A Model-Driven Approach for Hybrid Power Estimation in Embedded Systems Design

Chiraz Trabelsi, Rabie Ben Atitallah, Samy Meftali, Jean-Luc Dekeyser, Abderrazek Jemai

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2011, Article ID 569031, 15 pages
doi:10.1155/2011/569031

Review Article

A Model-Driven Approach for Hybrid Power Estimation in Embedded Systems Design

Chiraz Trabelsi,[1] Rabie Ben Atitallah,[2] Samy Meftali,[1] Jean-Luc Dekeyser,[1] and Abderrazek Jemai[3,4]

[1] *INRIA Lille Nord Europe-LIFL-USTL-CNRS, 40 Avenue Halley, 59650 Villeneuve d'Ascq, France*
[2] *LAMIH, University of Valenciennes, Le Mont Houy, 59313 Valenciennes, France*
[3] *LIP2 Laboratory, Faculty of Science of Tunis, 2092 Manar 2 Tunis, Tunisia*
[4] *Institut National des Sciences Appliquées et de Technologie (INSAT), B.P. 676, 1080 Tunis Cedex, Tunisia*

Correspondence should be addressed to Chiraz Trabelsi, chiraz.trabelsi@inria.fr

Received 15 December 2010; Accepted 21 February 2011

Academic Editor: Tulika Mitra

As technology scales for increased circuit density and performance, the management of power consumption in system-on-chip (SoC) is becoming critical. Today, having the appropriate electronic system level (ESL) tools for power estimation in the design flow is mandatory. The main challenge for the design of such dedicated tools is to achieve a better tradeoff between accuracy and speed. This paper presents a consumption estimation approach allowing taking the consumption criterion into account early in the design flow during the system cosimulation. The originality of this approach is that it allows the power estimation for both white-box intellectual properties (IPs) using annotated power models and black-box IPs using standalone power estimators. In order to obtain accurate power estimates, our simulations were performed at the cycle-accurate bit-accurate (CABA) level, using SystemC. To make our approach fast and not tedious for users, the simulated architectures, including standalone power estimators, were generated automatically using a model driven engineering (MDE) approach. Both annotated power models and standalone power estimators can be used together to estimate the consumption of the same architecture, which makes them complementary. The simulation results showed that the power estimates given by both estimation techniques for a hardware component are very close, with a difference that does not exceed 0.3%. This proves that, even when the IP code is not accessible or not modifiable, our approach allows obtaining quite accurate power estimates that early in the design flow thanks to the automation offered by the MDE approach.

1. Introduction

While the increasing integration of systems-on-chip (SoC) permits to increase their computation performances, the underlying power dissipation has become a dominant concern. Therefore, power consumption becomes a major criterion to take into account during design space exploration. An important design challenge is to find a tradeoff between performance and power consumption early in the design flow in order to satisfy time-to-market constraints. Cracking the power problem while maintaining acceptable design productivity requires estimation methods that support abstraction and automation.

Low-level energy estimation methods take into account many details of the simulated SoC, which leads to very slow simulations that increase the design time significantly, especially for complex systems. Despite the accuracy of such methods, their slowness represents an obstacle to productivity. Therefore, more abstract estimation techniques are required.

The cycle-accurate bit-accurate (CABA) level [1] is an abstraction level for a system description that is higher than the register transfer level (RTL). It allows obtaining faster simulations than those performed using RTL. Usually, to move from the RTL to the CABA level, hardware implementation details are hidden from the processing part of the system while preserving system behavior at the clock cycle level. The bit-accurate implies that a communication protocol is used between components at the bit level. At the CABA level, the behavior of the system can be simulated

cycle by cycle, which permits obtaining quite accurate power estimates. Thus, this abstraction level allows for a tradeoff between simulation speed and accuracy. Therefore, we chose this abstraction level for our simulations.

Due to the tremendous amount of hardware resources available in SoCs, design tools and methodologies are required to decrease the design complexity. Implementing these systems directly at a low level such as RTL can lead to errors. Therefore, an efficient design methodology, such as model driven engineering (MDE) [2], is needed in order to make the SoC design easy and not tedious, by making the low-level technical details transparent to designers. In MDE, models become a means of productivity. The graphical nature of MDE offered by the unified modeling language (UML) makes the comprehensibility of a system easier and allows users to model their systems at a high abstraction level, reuse, modify, and extend their models. Using the automation offered by MDE, the whole code necessary for the simulation of an SoC can be generated automatically from models describing the system. In order to use the MDE for a high-level description of a system in a specific domain such as embedded systems, UML profiles are used. A UML profile is a set of stereotypes that add specific information to a UML model in order to describe a system related to a specific domain. Several UML profiles target embedded systems design such as the modeling and analysis of real-time and embedded systems (MARTE) [3] profile. MARTE is a standard profile promoted by the object management group (OMG).

Gaspard2 [4] is a SoC codesign framework that is based on MDE to describe both the architecture and application parts of a system at a high abstraction level. Gaspard2 uses the MARTE profile for embedded systems modeling. It targets many technologies such as VHDL and SystemC using model transformations. The generated SystemC code for a modeled system using Gaspard2 is used for co-simulation in order to determine the system performance in terms of execution time. But, until now, the energy estimation has not been fully integrated into Gaspard2. Our contribution in this framework is to integrate power estimation at the modeling level of Gaspard2 as well as at the simulation level, allowing of automation the energy estimation in the Gaspard2 design flow.

An accurate power estimation method is based on a characterization phase using low-level tools in order to determine the consumption of the different activities of a hardware component accurately. The obtained power model is then used during simulations to estimate the consumption of the related component. During simulation, the simulator detects whether an activity has occurred for a given component and adds its consumption cost to the total consumption of the component. The most important challenge here is how to detect these activities especially if the intellectual property (IP) description codes are not accessible.

The main contribution of this paper is to present a hybrid energy estimation approach for SoC, in which the consumption of both white-box IPs and black-box IPs can be estimated. Based on model-driven engineering, this approach allows to take the consumption criterion into account early in the design flow, during the co-simulation of SoC. In a previous work [5], we presented an annotated power model estimation technique for white-box IPs where counters are introduced into the code of the IPs. A counter is incremented whenever its related activity occurs. This technique was used in this present work, along with the standalone power estimator technique used for black-box IPs. The standalone power estimation modules were generated using MDE and connected between the components in order to detect their activities through the signals that they exchange. To test this approach, systems containing white-box IPs and black-box IPs and their related estimation modules were modeled in the Gaspard2 framework. Using the MDE model transformations, the code required for simulation can be generated automatically. Finally, consumption estimates can be obtained during simulations.

The rest of this paper is organized as follows. Section 2 gives a summary of the related works. An overview of MDE and the Gaspard2 framework is provided in Section 3. Section 4 illustrates our hybrid approach for energy estimation. Section 5 describes the MDE approach used to implement our estimation modules and their integration in the Gaspard2 framework. This paper ends with simulation results in Section 6.

2. Related Work

There are many research efforts devoted to power consumption estimation in SoC design. They operate at different abstraction levels and have different estimation techniques.

At the layout level, the consumption estimation depends on the electric currents through the transistors, which requires a transistor-level description of the SoC. Optimizing the consumption can be done by transistor resizing and layout rearrangement depending on the obtained estimates. Among tools operating at this level, we can mention SPICE [6]. Although this approach is very accurate, it requires a high amount of processed data and simulation time. These limits represent an obstacle to apply this approach to complex systems. At gate level, consumption estimation is based on power models of technology cells. The power consumed by a cell is correlated to its input data. Furthermore, it depends on many other parameters such as supply voltage and frequency. Thus, to optimize the power consumption of a whole system, many experiments must be done in order to optimize consumption parameters. Power Gate [7] of Synopsys is among tools operating at this level. The accuracy of the estimation of such tools is still high, but, similar to the layout level, the amount of data and the simulation time are still limiting. For a 10-million-gate design, even the fastest software-based logic simulator may not exceed a few cycles per second [8]. Compared to the Gate and layout (transistor) levels, RTL brings, respectively, an acceleration of 10x and 100x. At the RTL level, systems are described using more abstract blocks such as adders and multipliers, which accelerates the simulation. Two approaches can be used for consumption estimation. The first one is based on

probabilistic estimation [9]. This approach has the advantage of accelerating the simulation but suffers from its low accuracy. The second approach, which is the most used, is based on event simulation. This technique relies on cycle accurate simulation and the application of macro models. The macro-models are derived from the processed data [10]. Petrol [11] is among the tools following this approach.

The simulation of a complete SoC at the three previous levels suffers from slowness. With the increasing complexity of designs, these estimation approaches become inadequate as they produce estimation results late in the design flow. More abstract estimation techniques are required to enable early design decisions. To achieve this goal, several studies have proposed evaluating power consumption at higher abstraction levels such as the CABA level [1, 12], on which this work is based. At this level, the behavior of components is simulated cycle by cycle using an architectural level simulator. An analytic power model is used to estimate consumption for each platform component. The power model of a component is based on the power costs of its pertinent activities and the occurrences of these activities during the simulation. The consumption of pertinent activities is estimated using power macro-models, produced during lower-level simulations. The power model of each component is then integrated into the simulator in order to estimate the consumption of a system in every cycle, which permits quite accurate estimates. Among tools operating at this abstraction level, we find SimplePower [13] and MPARM [14]. The main drawback of such tools is the use of intrusive approaches. Such approaches add some code lines to the monitored component IPs in order to determine their consumption. The problem here is that these approaches are not automated, because the consumption information has to be added manually into the IP source codes. A solution to this problem is to use standalone estimators connected between the monitored components. The consumption estimation is handled by these standalone modules, and the components IPs are not modified. Among tools using this approach, we cite UNISIM [15], which is a simulation framework that offers cycle accurate energy estimation based on shadow modules. A shadow module is connected to the module that is being monitored and uses the input of this latter to estimate its consumption. This approach is adaptable for many hardware configurations. The disadvantage of the shadow approach is that it uses a specific communication protocol between hardware components. In addition, in the UNISIM framework, the monitored modules need to notify the estimation modules whenever an operation has been performed. This means that the IP source codes of the monitored modules have to be changed in order to support this functionality, which means that this method is not completely nonintrusive.

At a higher level of abstraction, we find the TLM (transaction-level modeling) [16, 17], which permits faster simulation but less accurate estimates. In TLM, a set of abstraction levels simplifying the description of inter-module communication is defined. Consequently simulation time is reduced by increasing communication operation granularity (the communication is insured via channels instead of signals). In [18, 19], authors present a characterization method for generating power models within TLM, adopted to peripheral components. The pertinent activities are identified at several levels and granularities. The characterization phase of the activities is performed at the gate level and helps to deduce power of coarse-grain activities at higher level. However, this method cannot be applied to different kinds of components, such as processors, or interconnect networks. In [20], the authors present a transaction-level power estimation approach that uses characterization from a low-level as well as an analytical modeling method. This approach is applied in the Gaspard2 framework [4] allowing a fast MPSoC (multiprocessor SoC) design at the timed programmer view (PVT) level, one of the TLM levels. The major disadvantage of this approach is that it is intrusive.

At the functional level, Tiwari et al. [21] proposed the first method of the consumption estimation of a software program by introducing the concept of instruction level-power analysis (ILPA). This method consists accumulating the energy dissipated by every instruction of a program. The drawback of this method is that it requires a real experimental environment containing the processor to model and to execute the instructions in order to measure the consumed current. Thus, it requires many analyses and measures and much time to develop a power model for a target processor. Among the tools based on this approach, we find JouleTrack [22] that estimates only the consumption of simple processors. Several extensions of the ILPA have been proposed, such as the functional-level power analysis (FLPA) [23] that decreases the time necessary for the power model development. This method performs power estimation only at the assembly-level with accuracy from 4% for simple cases to 10% when both parallelism and pipeline stalls are effectively considered. Among the tools using this approach, we cite SoftExplorer [24] that covers the power analysis of simple and complex processors.

MDE was also used for power estimation in several works. In [25, 26], an analytical method for power estimation is proposed. The power estimates are obtained by an estimation tool called SPEU (system properties estimation with UML). This method is based on describing an application using UML diagrams and the UML-SPT profile [27]. Using model transformations and the SPEU tool, analytical estimates can be obtained using the cost specified in the models such as the costs associated with the services of a processor. Since the energy estimates are obtained by an analytical method and do not rely on simulations, this method allows for a fast design space exploration. However, in order to obtain fast estimates, the used components have to be precharacterized in terms of energy consumption.

A similar approach for power estimation is presented in [28, 29]. It uses the architecture analysis and design language (AADL) [30] to describe embedded application and operating systems. The power estimates are obtained using the consumption analysis toolbox (CAT). This tool is populated by power models built using the FLPA methodology extended by considering the overhead due to operating system services. Using the CAT tool, software parameters can be extracted from the AADL models in order to determine

the consumption of the application using the power models already integrated in CAT. This approach gives relatively precise results.

In [31], an extension to the MARTE profile, with a dynamic power management (DPM) profile, is proposed. This approach considers an embedded application as a set of use cases and links a power mode to each use case. In a power mode, the components of a systems may have different power states. This approach can be used for fast system energy dissipation estimation without accurate functional description or realization of the system. However, this approach is still at the conceptual level, and no analysis tool has been developed yet to evaluate energy dissipation.

Despite the speed of the model-driven power estimation approaches presented earlier, their lack of precision may have an important impact on the final estimation accuracy. This is due to the fact that they are based on the code analysis or a rapid profiling, which makes it difficult to determine some parameters with precision. This is the case, for instance, in cache miss rates in complex processors, which may have a nonnegligible impact on the final power estimation.

The solution proposed in this paper makes a tradeoff between the speed of simulation using a high abstraction level, and an acceptable accuracy compared with lower levels. Besides, it is nonintrusive, which is interesting especially if the source code of the IPs is not accessible. Furthermore, the consumption estimators are automatically generated using a model driven engineering approach, which permits a gain in the design time.

3. Model Driven Engineering and the Gaspard2 Design Framework

MDE revolves around three focal concepts. models, metamodels and model transformations. A model is an abstract representation of some reality and has two key elements: concepts and relations. Concepts represent "things", and relations are the "links" between these things in reality. A model can be observed from different abstract points of view (views in MDE). The abstraction mechanism avoids dealing with details and eases reusability. A metamodel is a collection of concepts and relations for describing a model using a model description language and defines syntax of a model. This relation is analogous to a text and its language grammar. Each model is said to conform to its metamodel at a higher definition level. Finally, MDE allows to separate the concerns in different models, allowing reutilization of these models and keeping them human readable.

The MDE development process starts from a high abstraction level and finishes at a targeted model, by flowing through intermediate levels of abstraction via model transformations (MTs) [32]; by which, concrete results such as an executable model (or code) can be produced. MTs carry out refinements moving from high abstraction levels to low-level models and help to keep the different models synchronized. At each intermediate level, implementation details are added to the MTs. An MT is a compilation process that transforms source models into target models and allows moving from

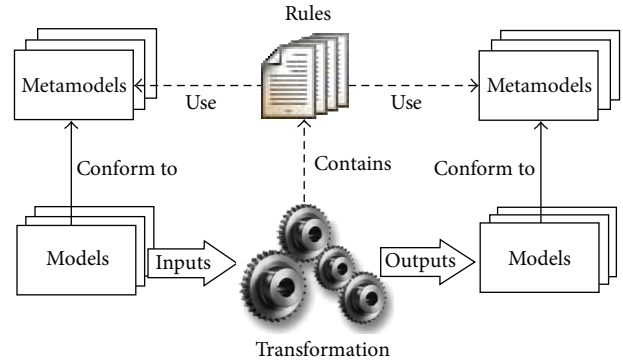

Figure 1: An overview of model transformations.

an abstract model to a more detailed model. In the case of an exogenous MT [33], as shown in Figure 1, the source and target models conform to two different metamodels while the endogenous transformation deals with two models conforming to the same metamodel. Usually, the initial high-level models contain only domain-specific concepts, whereas technological concepts are introduced seamlessly in the intermediate levels. An MT is based on a set of rules (either declarative or imperative) that help to identify concepts in a source metamodel in order to create enriched concepts in the target metamodel. It can be extended by adding or/and modifying rules in order to obtain a new MT targeting a different model. The advantage of this approach is that it allows defining several model transformations from the same abstraction level but targeted to different lower levels, offering opportunities to target different technology platforms. The model transformations can be either unidirectional (only source model can be modified; targeted model is regenerated automatically) or bidirectional (target model is also modifiable, requiring the source model to be modified in a synchronized way) in nature. In the second case, this could lead to a model synchronization issue [34]. For model transformations, OMG has proposed the metaobject Facility (MOF) standard for metamodel expression and query/view/transformation (QVT) [35] for transformation specifications.

Gaspard2 is an MDE-oriented SoC codesign framework based on the Modeling and Analysis for real-time and embedded systems (MARTE) standard. The applications targeted by Gaspard2 are control and flow-oriented ISP applications such as multimedia and high-performance applications. In Gaspard2, the UML component concept is used defining an application or an architecture component, and the MARTE FlowPort concept is used to define all ports in both the application and the architecture. To bridge the gap between high-level modeling using MARTE and execution platforms, Gaspard2 uses two main concepts: deployment and model transformation.

The design flow in Gaspard2 follows several steps: system modeling and deployment, model transformations and code generation. The left side of Figure 2 shows the design flow targeting the SystemC platform, which we used in our work. Gaspard2 also targets other platforms such as

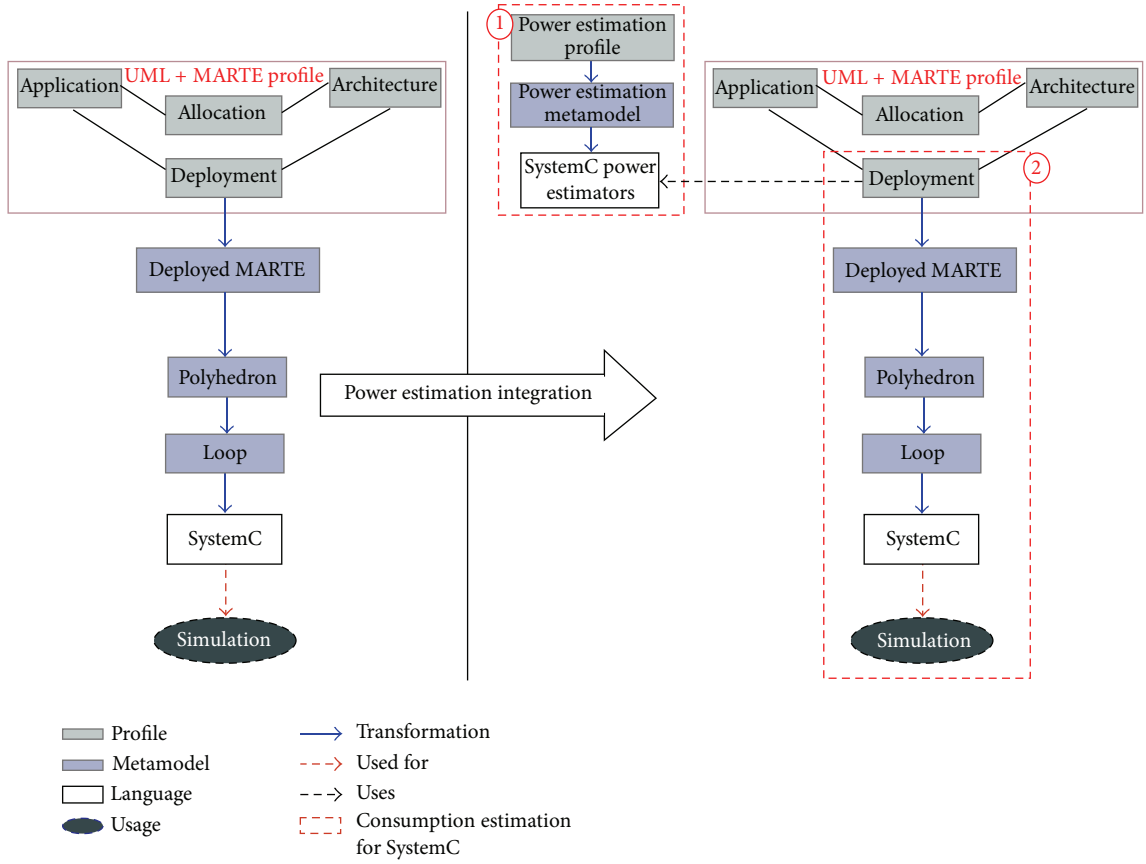

Figure 2: Power estimation integration in the Gaspard2 framework.

Fortran and VHDL. In Gaspard2, there is a separation between the architecture and the application models as shown in the left side of Figure 2. An application model describes the SW and/or HW components of an application. To link the application and the architecture models, an allocation model is used. At the deployment level, every elementary component (application or architecture, component) of a system is linked to an existing code hence, facilitating intellectual property (IP) reuse. Each elementary component can have several implementations (e.g., SystemC, VHDL). The deployment model provides IP information for model transformations targeting different domains (formal verification, simulations, high-performance computing or synthesis). After the deployment phase, model transformations (transformation chains) permit adding some details to the input model in order to get closer to the targeted technologies. At the end of a transformation chain, we have a model with technical details allowing the code generation related to the targeted technology. Model-to-model transformations are implemented with QVTO [36] and model-to-text transformations (code generation) with Acceleo [37].

Our contribution in Gaspard2 is to integrate the power estimation concepts in this framework. The integration follows two steps as shown in the right side of Figure 2. The first step is to generate power estimation modules for the SystemC IPs in the Gaspard2 IP library. For this, we developed a new profile and a new metamodel dedicated to power estimation. The profile allows describing the behavior of the power estimators and determining the architectural parameters that will be used in the consumption estimation later, when the estimators will be integrated into a whole SoC architecture. Using model transformations, the SystemC code of the estimators is generated automatically, in order to obtain new IPs dedicated to power estimation. These IPs are integrated in the IP library of Gaspard2. They are ready to be used for the consumption estimation of the hardware components used in the Gaspard2 system designs. The second step is to integrate the power estimation in SoC models in Gaspard2 in order to automate the power estimation for these systems. For this, we extended the deployment profile of Gaspard2. This extension allows linking the existing estimators in the library to the hardware components of a modeled SoC. The MDE approach of Gaspard2 then allows generating the SystemC code of the whole SoC with the integrated estimators. This code can then be integrated into the SystemC simulator, and the energy estimates can be displayed during the simulation.

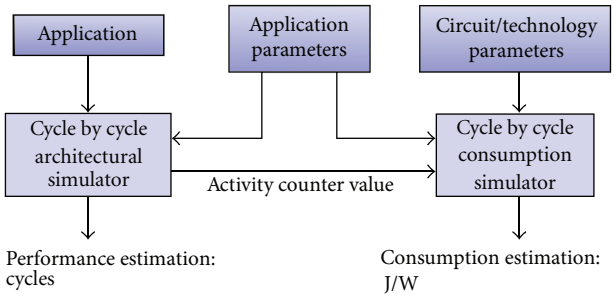

Figure 3: Consumption estimation at the CABA level.

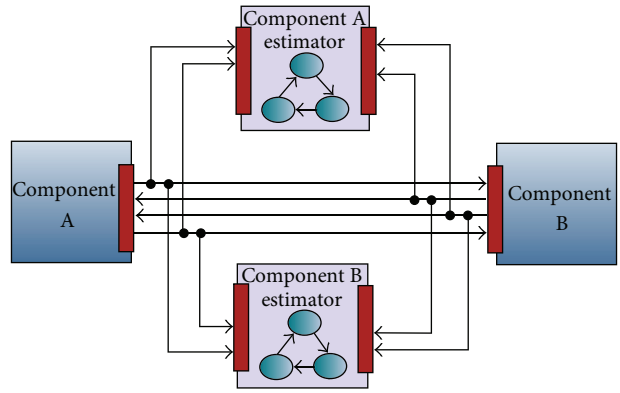

Figure 4: Power consumption estimators.

4. A Hybrid Consumption Estimation Approach

At the CABA level, the total energy consumption of a system is obtained by adding the consumptions of its components together. To provide accurate estimation, two kinds of consumption are considered: dynamic consumption related to component activity (e.g., read/write operations), and static consumption related to leakage currents when the component is inactive. For a long time, dynamic power consumption has been considered more significant than static consumption. However, this point of view changed with the advent of new submicron technologies, for which both types of consumption have their degree of importance. The consumption models used in this work are simple, since the related components are simple. Our consumption models are based on the following formula:

$$E = \sum_i N_i * C_i, \tag{1}$$

where N_i is the number of times the activity i is realized or the number of cycles the component is inactive, and C_i is the unit cost of the activity i or of a cycle of inactivity. Using this formula, we obtained the power models of the main components of the SoCLib library [38]: the processor, the cache memory, the shared SRAM memory and the interconnection network. These consumption models are detailed in [5]. They were obtained after we had determined each component pertinent activities and measured their unit costs using low-level tools. The same power models are used in this present work.

To determine the occurrences of each activity, activity counters are used. Each counter is incremented during the simulation if the corresponding activity occurs. This approach gives the consumption of the whole architecture in every cycle. The values of the activity counters are transmitted to the energy consumption models integrated into the simulator, to accumulate the energy dissipation of the architecture. The consumption simulator contains an energy model for each hardware component, which depends on its technology parameters. Figure 3 illustrates this approach.

The use of the counters depends on whether the IP's code is accessible or not. For white-box IPs, the counters can be inserted into their source code while for black-box ones, provided that their source code is not accessible, the activity occurrences are detected using standalone modules connected to these IPs. Our hybrid approach combines these two techniques in order to estimate the consumption of a system composed of both white-box and black-box IPs. In the following subsections, we detail consumption estimation techniques for white-box and black-box IPs.

4.1. Energy Estimation for White-Box IPs. In order to determine the occurrences of activities, a counter is inserted in the source code of the components for each activity type (e.g., memory read, memory write). A counter is incremented whenever its related activity is performed. Thus, we have to look for the code portions describing each activity and modify them by adding statements incrementing the corresponding counters. Despite the accuracy of this approach, due to the detection of all the consuming activities corresponding to a power model of a component, its drawback is that sometimes it is tedious to look for all the activities in an IP code especially with a great number of code lines. Besides, an IP is supposed to be unmodified, otherwise it is not the same IP anymore. In order to mend this problem, standalone energy estimation modules were implemented. These modules are also very useful especially when the code of the IPs is not accessible.

4.2. Energy Estimation for Black-Box IPs. Our solution to determine the occurrences of the activities without being intrusive in the IP codes is to detect these activities through the signals that the components exchange. Figure 4 shows that to determine the consumption of two linked components, we need a consumption estimator for each one. Here, we do not use only one estimator to estimate the consumption of both components because separating estimators allows their reuse with different architectures.

The description of an estimator at the CABA level is based on a power state machine as shows Figure 5. The power state machine contains the relevant power states of a component and the transitions between them. A power state is related to an activity of the monitored component or corresponds to a static consumption. The transitions are conditioned by the values of the signals that the monitored component exchange with others. Depending on

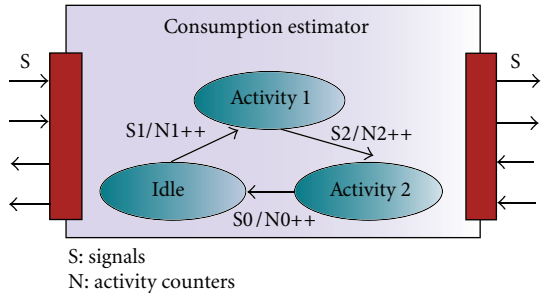

Figure 5: Power estimator structure.

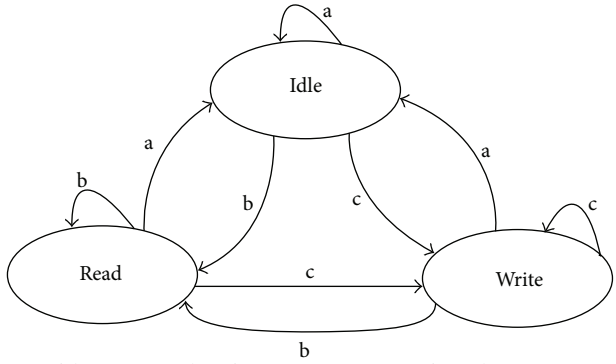

Figure 6: The SRAM estimator's FSM.

these signals, it is possible to determine the current activity of a component, so a transition to the corresponding power state is carried out. In each transition, there is an energy consumption related to the target state, so the related activity counter is incremented.

Let us take the example of an SRAM memory, three main activities consume energy: *Read*, *Write*, and *Idle*. These activities correspond to the read access mode, the write access mode, and the waiting state. This approach is similar to the approach proposed by Loghi et al. [14]. Thus, the SRAM estimator's finite-state machine (FSM) is composed of three states: *Read*, *write*, and *Idle* as Figure 6 shows. Connected between the SRAM and the interconnect, the SRAM estimator intercepts the requests that the SRAM receives and the responses that it sends. If the SRAM estimator receives a response from the SRAM corresponding to a read request, there will be a transition to the *Read* state. If the SRAM does not send any response, that means that it is inactive, which corresponds to the Idle state in the FSM. So, the SRAM total energy follows this equation:

$$E_{\text{SRAM}} = n_{\text{read}} \cdot E_{\text{read}} + n_{\text{write}} \cdot E_{\text{write}} + n_{\text{idle}} \cdot E_{\text{idle}}, \quad (2)$$

where $n_{\text{read}}, n_{\text{write}}$, and n_{idle} are, respectively, the number of occurrences of a read access, a write access and a cycle of inactivity of the SRAM. These occurrences are given by the counters associated to each state in the FSM. $E_{\text{read}}, E_{\text{write}}$, and E_{idle} are, respectively, the costs of a read access, a write access and a cycle of inactivity. These costs depend on the number of words and the number of bits per word in the SRAM.

We chose, for instance, the OCP (open core protocol) [39] for the interfaces of our estimators. This protocol allows describing any type of communication. This solution permits the use of a single standard protocol for the estimators, which allows them to be connected to components with different interface standards. Therefore, we used wrappers to make the evaluated components compliant with the OCP so we can connect estimators to them. Here, OCP is only an example of a communication protocol. Our approach can be applied to other protocols.

5. An MDE Approach for Power Consumption Estimation

In order to facilitate the energy estimation for SoC designers, the estimation modules can be generated automatically using an MDE approach. This approach has many benefits, for example, to modify the code related to an estimator, users are not obliged to write a long code, they only need to enter some modifications in the estimator model and relaunch the code generation process to obtain the desired result. Thus, model driven engineering is a solution to make power consumption estimation a fast and nontedious work for users in order to respect the time-to-market constraints. The integration of the consumption estimation in the design flow of the Gaspard2 framework [4] thus allows fast architectures exploration. To implement this integration, we used the same MDE tools as Gaspard2, namely, Papyrus [40] for graphical modeling, QVTO [36] for model transformations, and Acceleo [37] for code generation.

The integration of power estimation in the Gaspard2 framework follows two steps. The first step is to model power estimators for several hardware components of the Gaspard2 library and to generate SystemC code for these estimators. The second step is to use the generated estimators in the SoC models. This is done by linking the estimators to the hardware components of an SoC at the deployment level of Gaspard2. Using the transformation chain shown in Figure 2, the SystemC code corresponding to an SoC with its integrated power estimator that can be generated and used for power estimation during simulation. The two steps of our approach are detailed in the following sections.

5.1. Fist Step: Estimators Generation. In order to obtain the SystemC code of the estimators, the SoC designer has to follow a transformation chain as Figure 7 shows. First, the estimation modules are modeled using UML. To facilitate the modeling of the estimators for users, we have designed a UML profile dedicated to power estimation, so that they have a graphical view of the estimator's structure, FSM and deployment. The MARTE profile used for SoC modeling in Gaspard2 cannot be used for power estimators description

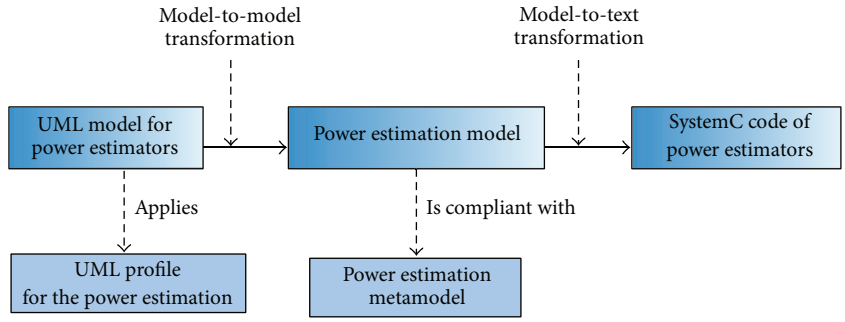

Figure 7: The estimators generation process.

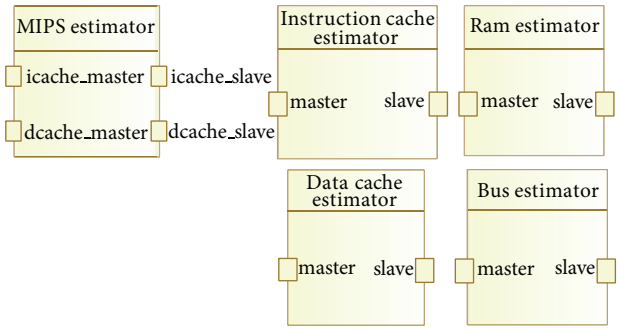

Figure 8: Power estimation Library.

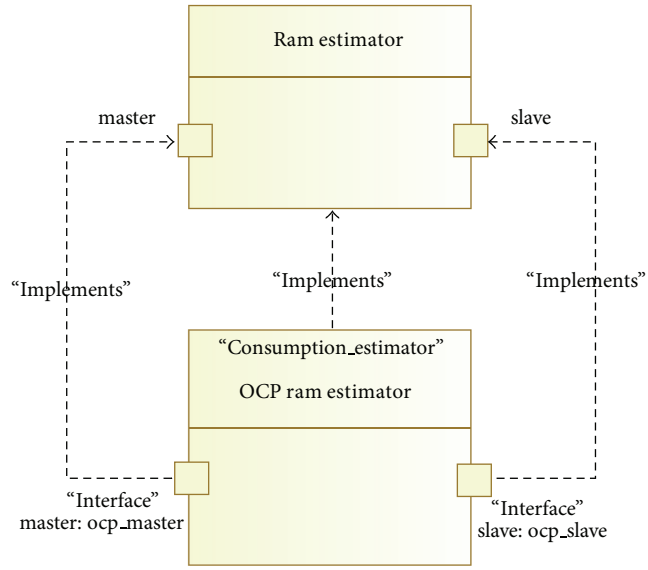

Figure 9: Power estimator implementation.

because there are concepts such as state machines, that we need and that we do not find in MARTE. Second, the estimators model is transformed into a model compliant with a power estimation metamodel. This metamodel is also dedicated to power estimation. It is a metamodel different from the metamodels used in Gaspard2 since it describes the internal structure of an elementary component (the power estimator) in order to generate its code while, in Gaspard2 the codes of the elementary components already exist in its library. Finally, the power model obtained by the model transformation is transformed into a SystemC code using a model-to-text transformation. These three substeps are detailed in the rest of this subsection. Using the power estimation profile, the SoC designer can elaborate the estimator models easily. Three types of UML diagrams are used here: composite, state machine, and deployment diagrams. Figure 8 shows an example of a model representing a library of estimators using a composite diagram. In this figure, we modeled the estimators for the MIPS processor, the instruction and data caches, the shared memory, and the interconnect. An estimator is represented by a UML component. The ports of this component correspond to the interfaces of the estimator. An estimator can be implemented by indicating its configuration and consumption parameters and the description of its interfaces, using the *Consumption-Estimator* and *Interface* stereotypes, respectively, as Figure 9 shows.

The UML state machine diagrams are used in order to describe the behavior of the estimator, which corresponds to the power machine of the monitored hardware component. A deployment diagram is used to give the code files related to the interfaces. Here, the code files describing the OCP interfaces already exist in the Gaspard2 library. So, we only need to make a link to those files. The code of the internal operations of the estimators is already in the library. The paths to these code files are specified in the deployment diagram so that the generated SystemC estimators include them in their headers. Thus, for the internal operations, the estimator only has to call the functions related to those operations.

The elaborated model can be then transformed into a model corresponding to the power estimation metamodel. The power estimation metamodel is composed of two parts: one part describes the structure of the estimators, and the other part describes their behavior using the FSM. Figures 10 and 11 show, respectively, the structure and the FSM parts of the consumption estimation metamodel. The structure part indicates that the estimation metamodel is based on an *EstimationModel*. The *EstimationModel* may contain one or several *ConsumptionEstimators*, which allows to have more than one estimator in the same model. A *ConsumptionEstimator* has *Interfaces* which may be OCP

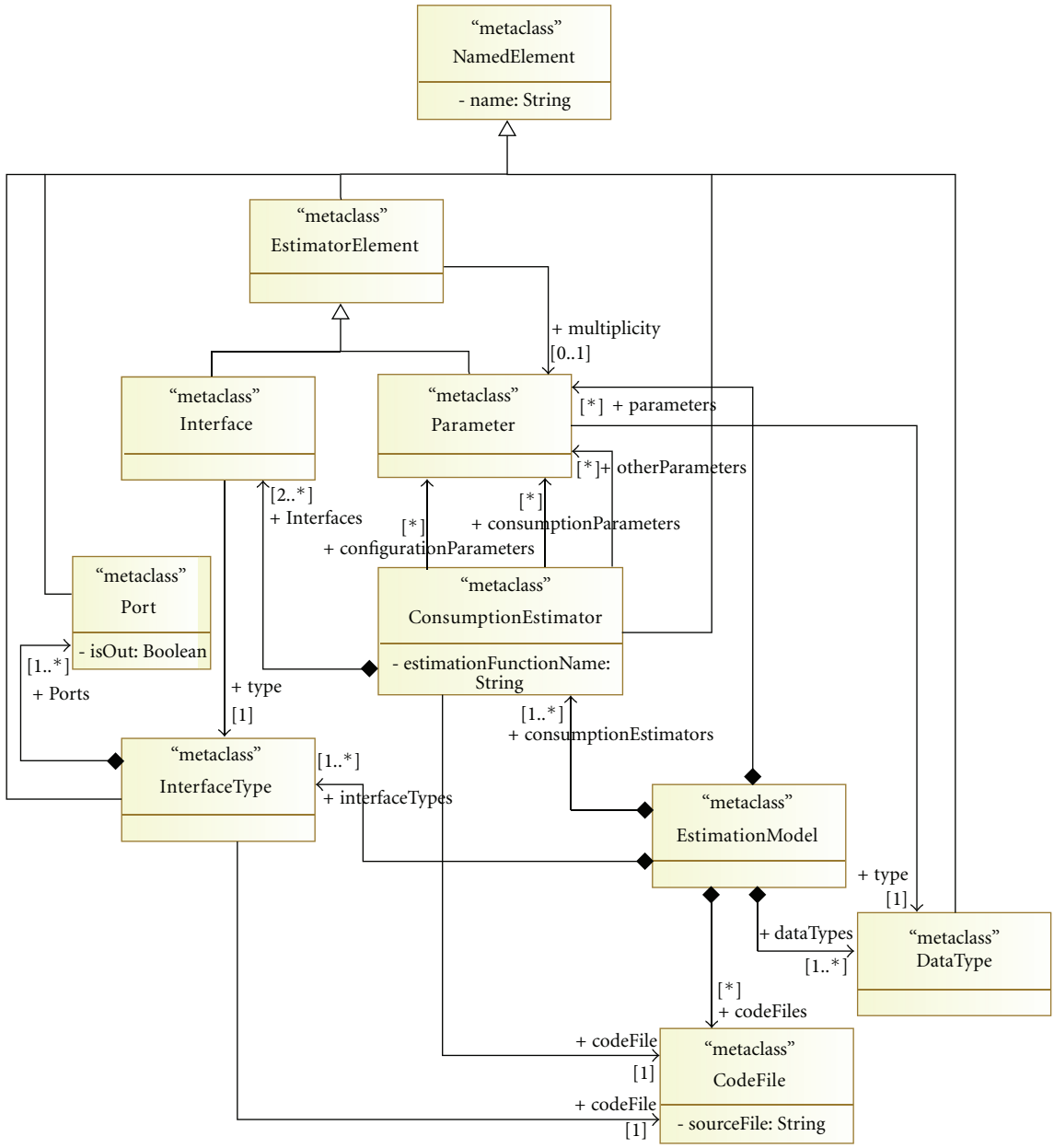

Figure 10: Energy estimation metamodel—the consumption estimator structure.

interfaces or any other type of interfaces. The interfaces have *InterfaceTypes* such as an OCP master or an OCP slave. To each *InterfaceType*, there are associated input and output ports. To take into account the configuration of the monitored component, the estimator uses some *Parameters* (*configurationParameters*) which may also serve in the energy consumption estimation (*consumptionParameters*) such as the number and size of cache blocks. In certain cases, the estimator has to save some data in registers in order to use them as a condition for next transitions. For example, the cache estimator needs to know if there was a cache miss previously, that is the use of the association *otherParameters*. Saving the values of these parameters is an *activity* which accompanies a transition or is a *doActivity* of a state in the estimator FSM. A *Parameter* has a *DataType* and a *multiplicity* in order to indicate if it is an array. The size of the array is also a *Parameter* of the *ConsumptionEstimator*. The path to the *InterfaceType* source code is given by the *CodeFile* metaclass.

The FSM of the estimator is inspired from the state machine of the UML metamodel. It contains the main concepts of this metamodel. It also contains power estimation information using the EstimatorElement metaclass. This metaclass represents the estimator parameters and interfaces, as shown in Figure 10, which are used in the activities of the state machine. An estimator has a *StateMachine* which contains *States* and *Transitions* between *States*. A *State* may have a *doActivity*. A *Transition* may have a *condition* and

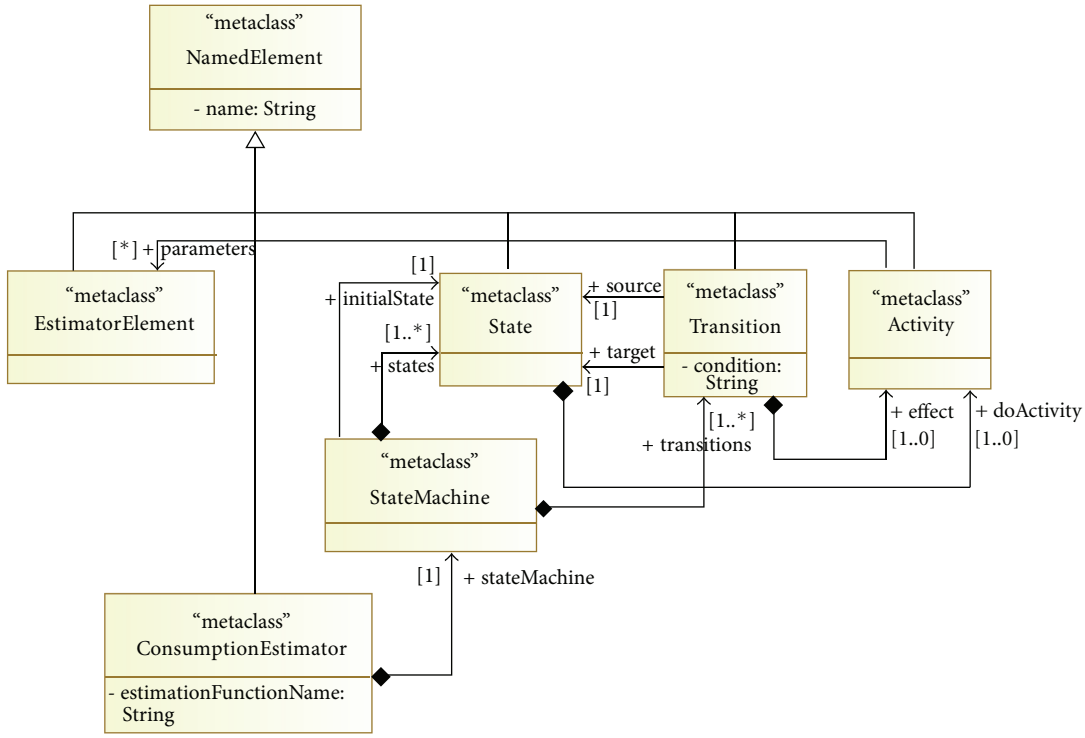

Figure 11: Energy estimation metamodel—the consumption estimator FSM.

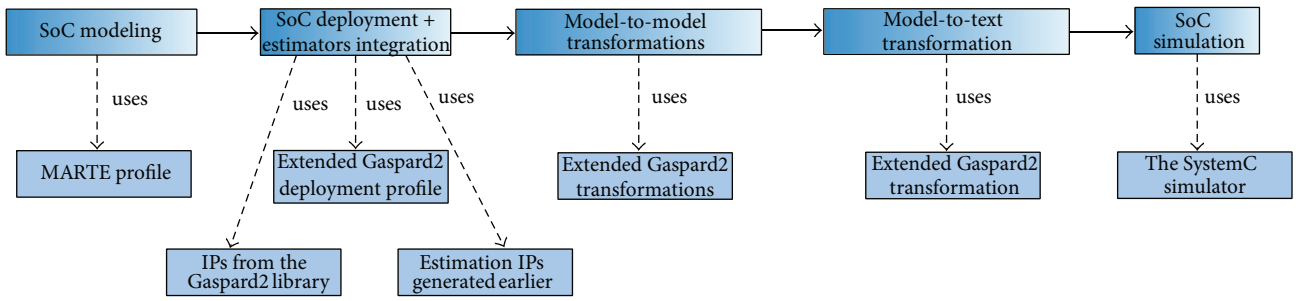

Figure 12: Integration of power estimators in SoC models.

an *effect*. An *Activity* manipulates several *EstimatorElements*. The source code path of an activity is given by the *CodeFile* metaclass. This supposes that all the activities of an estimator are saved in the same file with the function that calculates the consumption of the monitored component and whose name is given by the property *estimationFunctionName* of the *ConsumptionEstimator* metaclass.

After the model-to-model transformation targeting an estimation model that is compliant to the metamodel described above, the model-to-text transformation transforms this model into a SystemC code using Acceleo [37]. With this process, we have generated estimators with 200 to 300 code lines, compared with 10 to 120 lines inserted into IPs with the intrusive approach. This difference is because of the code necessary to manage the FSM of the estimator and especially the conditions of the transitions. Here, the MDE approach saves us a long coding time.

5.2. Second Step: Integration of Power Estimators in SoC Models. This step is a set of several substeps as described in Figure 12. The first sub-step is to model an SoC in the Gaspard2 framework. Gaspard2 is an environment that permits to model a whole MPSoC architecture using UML and then generate the simulation code in various abstraction level languages such as VHDL and SystemC [4]. Gaspard2 uses the MARTE profile for system modeling. Figure 13 shows an example of a system modeled by Gaspard2. This system contains the components that we will monitor (the processors, the cache memories, the shared memories, and the interconnect). These components are modeled using some stereotypes from the MARTE profile, such as *HW_Processor*, *HW_Cache*, and *HW_RAM*. At the deployment level of Gaspard2, the existing IPs are linked to the modeled architectural components. At this level, we use stereotypes such as *VirtualIP* and *HardwareIP* from

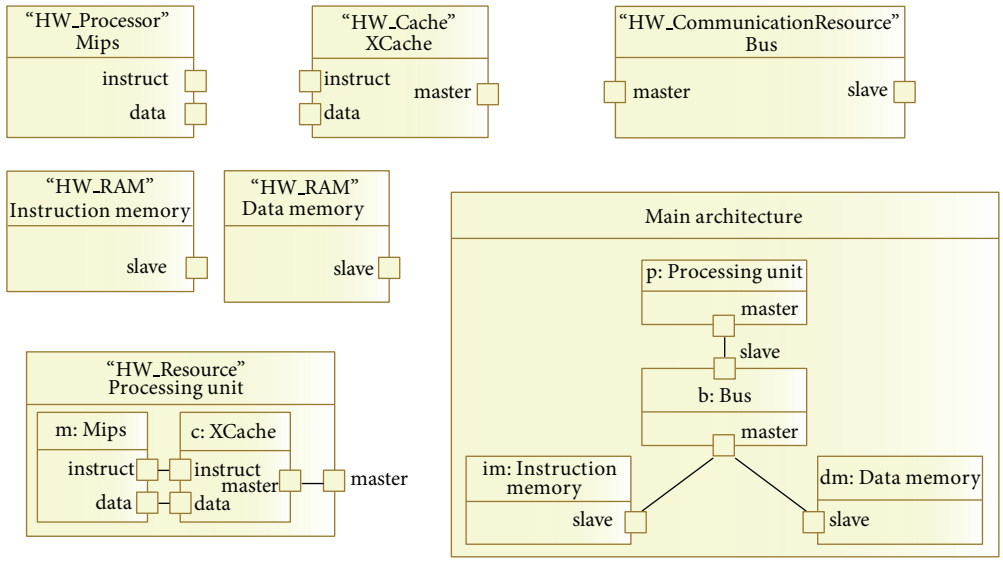

Figure 13: An architecture model in Gaspard2.

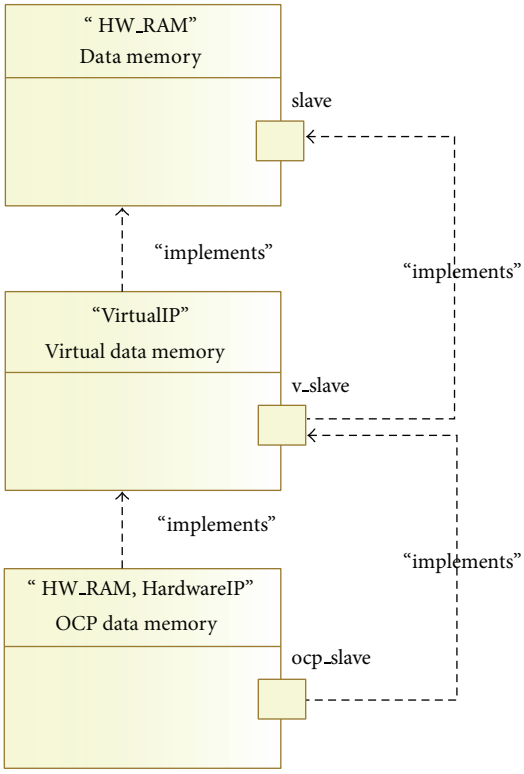

Figure 14: Architecture deployment in Gaspard2.

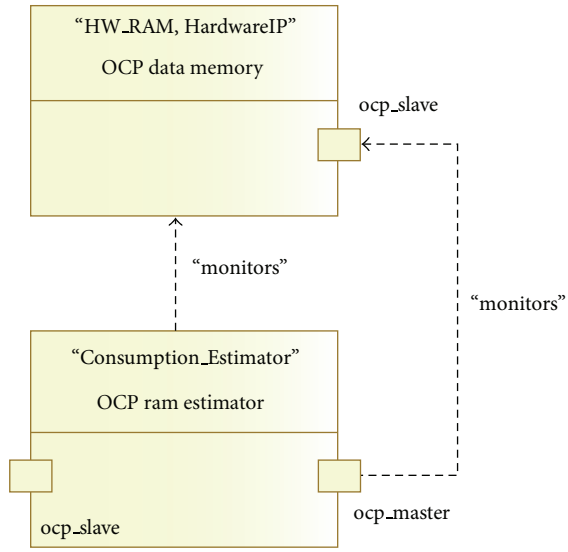

Figure 15: Example of a memory consumption estimator integration in a Gaspard2 SoC design.

the deployment profile of Gaspard2. Figure 14 shows the deployment of the memory component from the architecture described in Figure 13. The integration of the consumption estimation in an architecture is done at the deployment level. Therefore, we extended the deployment profile of Gaspard2 by adding the *Consumption Estimator* and the *monitors* stereotypes. The *Consumption Estimator* stereotype is applied to a component that represents an existing power estimator in the Gaspard2 library. These estimators are those we generated earlier. Figure 15 shows an example of the integration of the estimation modules in the deployment model in Gaspard2. Here, the deployment diagram shows that the RAM memory that is used in the system is OCP compliant, and that its estimator is connected to its OCP slave interface. So, at the generation of the code for the simulation, there will be an insertion of an estimator between the memory and the other component of the MPSoC to which the memory is connected. To do this, we introduced some modification in the transformation chain of Gaspard2 targeting the SystemC platform. At the end of the chain, we obtain the whole system code with

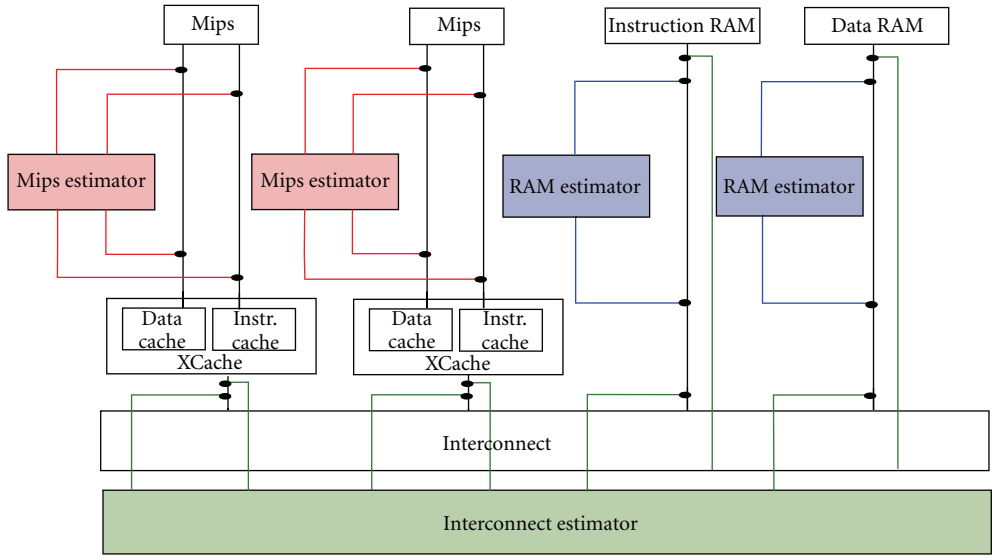

Figure 16: Example of simulated architectures with consumption estimators.

integrated consumption estimators in order to integrate power estimation during the system simulation.

6. Simulation Results

To validate our approach, we estimated the consumption of a JPEG compression application for images of 256 ∗ 256 pixels. The inputs of this software application are images in the BMP format, and the outputs are images in the JPEG format. The JPEG application was simulated on different architectures. The hardware components used in these architectures are MIPSR3000 processors, 16 KB SRAM memories, and micronetworks. The used caches contain actually two independent instruction and data caches, sharing the same interface. They are direct mapped caches. The data cache's writing policy is write-through.

All the IPs used here are white-box IPs (with an accessible code), but we can apply to them the same approach as black-box IPs. Thus, in order to insert the counters into the code, we use standalone estimators. This way, we can use both approaches for a simulated system at the same time, using a white-box approach for some IPs and a black-box one for the others. Figure 16 shows an example of the simulated architectures with 2 processors. In order to observe the communication between two components, the estimator needs two interfaces to observe the communication in both directions. In the example described in Figure 16, the white-box approach was used for the cache memories whereas the black-box approach was used for the rest of the IPs.

In order to follow the evolution of the performance during the simulation time, the consumption of each component of an architecture was written into an XML file that serves as a data source for a reporting engine that generates a consumption report. The report gives the percentage of the consumption of each component at different simulation

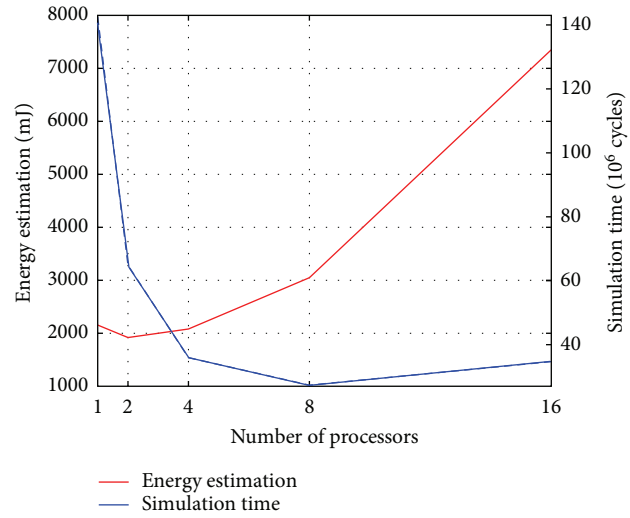

Figure 17: Performance and energy variation in terms of the number of processors.

cycles and information about the current cycle. It can also give details about a cycle chosen by the user and indicate the cycle where a consumption threshold fixed by the user was exceeded. This report gives a graphical view of the performance of the simulated architecture, which facilitates the design space exploration.

To evaluate the impact of the number of processors on the performance and the total consumption of the system, we executed the JPEG application using systems with 1 up to 16 processors. The size of the instruction and data cache was set to 4 KB, and the MIPS frequency was set at 50 MHz. All the processors execute the same JPEG application but on different image macroblocs. Figure 17 reports the execution time in cycles and the total energy consumption in mJ.

Table 1: Simulation time and energy consumption for combined estimation techniques.

Estimation approach				Simulation time (s)	Energy consumption (mJ)
Processor	Cache memory	Shared memory	Interconnect		
W	W	W	W	436	2080.4
W	W	W	B	550	2080.4
W	W	B	W	499	2080.4
W	W	B	B	578	2080.4
W	B	W	W	607	2077.4
W	B	W	B	731	2077.4
W	B	B	W	632	2077.4
W	B	B	B	756	2077.4
B	W	W	W	514	2080.4
B	W	W	B	598	2080.4
B	W	B	W	541	2080.4
B	W	B	B	627	2080.4
B	B	W	W	661	2077.4
B	B	W	B	781	2077.4
B	B	B	W	679	2077.4
B	B	B	B	828	2077.4

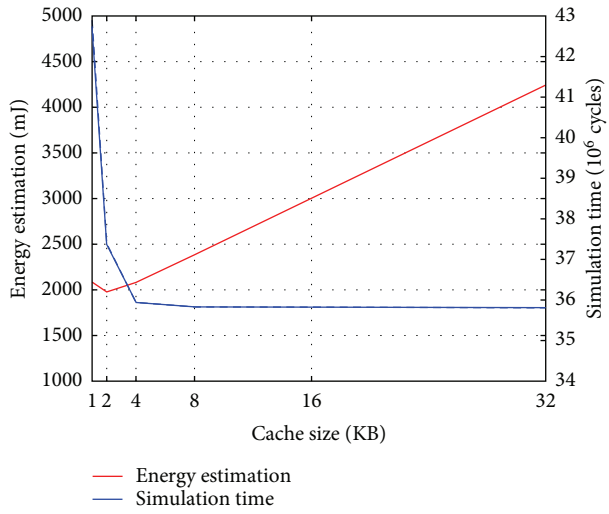

Figure 18: Variations in performance and energy consumption in terms of cache size with 4 processors.

Given these results, it seems that adding processors to the system decreases execution time, which improves system performance. This variation is not linear because the processors share resources, and, sometimes, they cannot reach the same target simultaneously, which necessitates waiting cycles and diminishes system performance. In terms of energy consumption, up to a certain number of processors, the total system energy consumption decreases as the number of execution cycles is reduced, and then it tends to stabilize as the system performance improves. But increasing the number of processors over a certain limit tends to be ineffective, as it just adds new conflicts at the interconnect, leading to more waiting cycles, which alters overall performances, especially in terms of power consumption.

To examine the impact of varying instruction and data cache sizes on the performance and energy consumption of the whole system, we used a 4-processor configuration. We executed the JPEG-parallelized algorithm using instruction and data caches of increasing size: from 1 KB up to 32 KB. Our results are presented in Figure 18.

The increase in the cache size increases the overall system energy consumption significantly. In general, larger caches improve system performance; however, this depends on the size of the task or of the data to be handled. In our example, the move from 4 KB to 8 KB, for instance, improved performance by 0.3% but also increased energy consumption by 14%. Between the 8 KB and 16 KB caches, the performance did not change, but energy consumption increased by 78%.

To prove that both white-box and black-box approaches can be used at the same time for a system, we simulated a system composed of 4 processors with a cache size of 4 KB. For each simulation, we varied the estimation approach for the different components. Since we have here 4 types of components (the processor, the cache memory, the shared memory, and the interconnect), we obtained 16 different simulations as Table 1 shows. The simulation results show that the energy estimates are very close for all combinations, with a difference that does not exceed 0.15%. This difference is due to the consumption of the cache FIFOs that was considered in the intrusive approach but not in the nonintrusive one, since this information is not detectable through the signals between the component of an architecture. However, provided that the consumption of the FIFO is very small, it can be neglected, and we can say that using standalone estimation modules permitted getting accurate results because it took into account the main pertinent and the most consuming activities of the monitored hardware components.

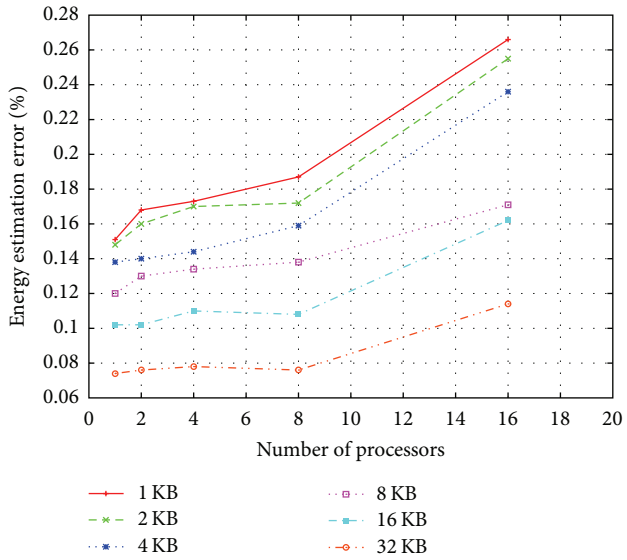

Figure 19: Variation in energy consumption estimation error in terms of cache size and number of processors.

Since the black-box approach uses additional modules added to the simulated system, this approach increases the simulation time as Table 1 shows. This increase does not exceed 90%, which is obtained when the black-box approach is applied to all the components of a system. The combination of both approaches for the same system gives faster simulations. The increase in the simulation time can be considered as acceptable because the simulation at the CABA level is fast, and having a doubled (the worst case) simulation time still allows to have fast simulations.

In order to test the efficiency of the black-box approach, we simulated systems with different numbers of processors and cache sizes. Two types of simulations were done. The first type uses the black-box approach for all the components of a system, and the second type uses the white-box one. The increase in simulation time between a system using a white-box approach and a system, with the same number of processors and cache size, using the black-box approach, was between 80% and 98%. If we consider the white-box approach as a reference, the estimation error of the black-box approach does not exceed 0.3% as Figure 19 shows.

7. Conclusion

This paper presents a hybrid approach for energy estimation for systems-on-chip (SoC). This approach is applicable for both white-box and black-box IPs. For white-box IPs, our approach inserts activity counters into the IP codes in order to detect the activity occurrences and determining the consumption of these activities, during the simulation of a system. For black-box IPs, estimation modules are connected between the IPs in order to detect their activities through the signals that they exchange during the simulation. These modules were generated using an MDE approach. The simulated systems were also generated automatically from high-level models in the Gaspard2 framework. In order to integrate the energy estimation in the design flow of this framework, we extended its deployment level. The simulations were implemented in SystemC at the CABA level. This high abstraction level allowed us to obtain fast simulations and quite accurate results due to an acceptable amount of architectural details that this abstraction level takes into account. In order to test the efficiency of our approach, we used a combination of the white-box and the black-box approaches for different system configurations. The simulation results showed that even if the code of an IP is not accessible, we can still obtain its consumption estimates with an acceptable accuracy. For this, we only need to detect, from the signals that it exchanges with the other components of a system, the activity that it is performing and associate its related consumption cost. In our future research, we plan to adapt our approach to more complex architectures including other components, such as other types of processors and interconnects. Measurements of the consumption from real implementations of the studied systems can also be considered, in our future works, in order to compare them with the results of the simulations.

References

[1] R. Buchmann and A. Greiner, "A fully static scheduling approach for fast cycle accurate systemc simulation of MPSoCs," in *Proceedings of the International Conference on Microelectronics (ICM '07)*, pp. 101–104, Cairo, Egypt, 2007.

[2] OMG, "Portal of the model driven engineering community," http://www.planetmde.org/.

[3] OMG, *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, 2009.

[4] DaRT, "Gaspard2 framework," 2009, http://www.gaspard2.org/.

[5] R. Ben Atitallah, S. Niar, A. Greiner, S. Meftali, and J. L. Dekeyser, "Estimating energy consumption for an MPSoC architectural exploration," in *Proceedings of the ARCS*, vol. 3894 of *Lecture Notes in Computer Science*, pp. 298–310, Frankfurt, Germany, 2006.

[6] M. Andersson and P. Kuivalainen, "Spice macromodel for power dmos transistors," in *Proceedings of the Nordic Semiconductor Meeting*, Finland, 1992.

[7] Synopsys, "Synopsys low power solutions for asic design flow," Tech. Rep., Synopsys, 1998, http://vada.skku.ac.kr/ClassInfo/ic/lowpower/.

[8] C. Rowen, "Reducing SoC simulation and development time," *IEEE Computer*, vol. 35, no. 12, pp. 29–34, 2002.

[9] J. Costa, J. Monteiro, L. M. Silveira, and S. Devadas, "A probabilistic approach for rt-level power modeling," in *Proceedings of the 16th IEEE International Conference on Electronics, Circuits and Systems*, Cyprus, 1999.

[10] Q. Wu, Q. Qiu, M. Pedram, and C. S. Ding, "Cycle-accurate macro-models for RT-level power analysis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 4, pp. 520–528, 1998.

[11] R. P. Llopis and K. Goossens, "Petrol approach to high-level power estimation," in *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 130–132, August 1998.

[12] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: exploring the multi-processor SoC design space with systemC," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 41, no. 2, pp. 169–182, 2005.

[13] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Design and use of SimplePower: a cycle-accurate energy estimation tool," in *Proceedings of the Design Automation Conference*, pp. 340–345, Los Angelos, Calif, USA, 2000.

[14] M. Loghi, M. Poncino, and L. Benini, "Cycle-accurate power analysis for multiprocessor systems-on-a-chip," in *Proceedings of the ACM Great Lakes Symposium on VLSI*, pp. 401–406, 2004.

[15] D. Ludovici, G. Keramidas, G. N. Gaydadjiev, and S. Kaxiras, "Integration of power saving techniques in the unisim simulation framework through the shadow module design paradigm," in *Rapid Simulation and Performance Evaluation*, 2009.

[16] A. Donlin, "Transaction level modeling: flows and use models," in *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '04)*, pp. 75–80, September 2004.

[17] Y. Lo, S. Abdi, and D. Gajski, "Transaction level model automation for multicore systems," in *Proceedings of the 38th DAC Conference, IGI Global*, 2009.

[18] I. Lee, H. Kim, P. Yang et al., "PowerViP: SoC power estimation framework at transaction level," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '06)*, vol. 2006, pp. 551–558, 2006.

[19] N. Dhanwada, I. C. Lin, and V. Narayanan, "A power estimation methodology for SystemC transaction level models," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (CODES+ISSS '05)*, pp. 142–147, NJ, USA, September 2005.

[20] R. B. Atitallah, S. Niar, and J.-L. Dekeyser, "MPSoC power estimation framework at transaction level modeling," in *Proceedings of the International Conference on Microelectronics (ICM '07)*, pp. 245–248, Cairo, Egypt, 2007.

[21] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 437–445, 1994.

[22] A. Sinha and A. P. Chandrakasan, "JouleTrack—a web based tool for software energy profiling," in *Proceedings of the 38th Design Automation Conference*, pp. 220–225, June 2001.

[23] N. Julien, J. Laurent, E. Senn, and E. Martin, "Power estimation of a c algorithm based on the functional-level power analysis of a digital signal processor," in *Proceedings of the 4th International Symposium on High Performance Computing (ISHPC '02)*, London, UK, 2002.

[24] E. Senn, J. Laurent, N. Julien, and E. Martin, "SoftExplorer: estimation, characterization, and optimization of the power and energy consumption at the algorithmic level," in *Proceedings of the IEEE PATMOS*, vol. 3254 of *Lecture Notes in Computer Science*, pp. 342–351, 2004.

[25] M. F. D. S. Oliveira, L. B. De Brisolara, L. Carro, and F. R. Wagner, "Early embedded software design space exploration using UML-based estimation," in *Proceedings of the 17th IEEE International Workshop on Rapid System Prototyping (RSP '06)*, pp. 24–32, June 2006.

[26] M. F. da S. Oliveira, E. W. BriÃo, F. A. Nascimento, and F. R. Wagner, "Model driven engineering for mpsoc design space exploration," in *Proceedings of the 20th Annual Conference on Integrated Circuits and Systems Design (SBCCI '07)*, Rio de Janeiro, Brazil, 2007.

[27] OMG, *Uml Profile for Schedulability, Performance, and Time*, 2002.

[28] E. Senn, J. Laurent, E. Juin, and J. P. Diguet, "Refining power consumption estimations in the component based AADL design flow," in *Proceedings of the Forum on Specification, Verification and Design Languages (FDL '08)*, pp. 173–178, September 2008.

[29] S. Dhouib, E. Senn, J.-P. Diguet, J. Laurent, and D. Blouin, "Model driven high-level power estimation of embedded operating systems communication services," in *Proceedings of the International Conference on Embedded Software and Systems (ICESS '09)*, pp. 475–481, May 2009.

[30] SAE, "The sae aadl standard info site," http://www.aadl.info/.

[31] T. Arpinen, E. Salminen, T. D. Hamalainen, and M. Hannikainen, "Extension to marte profile for modeling dynamic power management of embedded systems," in *Proceedings of the W6 1st Workshop on Model Based Engineering for Embedded Systems Design (M-BED '10)*, 2010.

[32] S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.

[33] T. Mens and P. V. Gorp, "A taxonomy of model transformation," in *Proceedings of the International Workshop on Graph and Model Transformation (GraMoT '05)*, vol. 152, pp. 125–142, 2006.

[34] P. Stevens, "A landscape of bidirectional model transformations," in *Proceedings of the GTTSE*, 2007.

[35] OMG, "MOF Query /Views/Transformations," 2005, http://www.omg.org/cgi-bin/doc?ptc/2005-11-01.

[36] QVTO, "Eclipse Model To Model (M2M) Project," http://www.eclipse.org/m2m/.

[37] Acceleo, "MDA Generator Home," http://www.acceleo.org/pages/home/en.

[38] SoCLib, "Soclib project: an integrated system-on-chip modeling and simulation platform," Tech. Rep., CNRS, 2003, http://www.soclib.fr/trac/dev.

[39] OCP, "OCP International Partnership," http://www.ocpip.org/.

[40] Papyrus, "Papyrus: Open Source Toolfor Graphical UML2 Modelling," http://www.papyrusuml.org/.