An MDE Approach for Energy Consumption Estimation in MPSoC Design

Chiraz Trabelsi, Samy Meftali, Rabie Ben Atitallah, Abderrazak Jemai, Jean Luc Dekeyser, Smail Niar

An MDE Approach for Energy Consumption Estimation in MPSoC Design

Chiraz Trabelsi
INRIA Lille Nord Europe
France
chiraz.trabelsi@inria.fr

Samy Meftali
INRIA Lille Nord Europe
France
meftali@lifl.fr

Rabie Ben Atitallah
INRIA Lille Nord Europe
France
benatita@lifl.fr

Abderrazak Jemai
LIP2 Laboratory Faculty of Science
of Tunis, Tunisia
Abderrazak.Jemai@insat.rnu.tn

Jean Luc Dekeyser
INRIA Lille Nord Europe
France
dekeyser@lifl.fr

Smail Niar
INRIA Lille Nord Europe
France
niar@lifl.fr

ABSTRACT

Energy Consumption is a leading criterion to take into account in the design of multiprocessor systems on chip (MPSoC). In this paper, we present a solution to estimate the energy consumption early in MPSoC design in order to find a good performance/energy trade-off in the design flow. This solution is based on the injection of consumption estimators between the hardware components during the co-simulation of a system at the CABA (Cycle Accurate Bit Accurate) level. These estimators are designed using a design framework and the corresponding SystemC code is automatically generated thanks to a model driven approach. Our solution offers an energy estimation framework without changing the IP(Intellectual Property)source codes, using standalone estimation modules, which allows their reuse. The accuracy of this approach is checked by integrating the consumption estimation in the simulation of significant applications.

1. INTRODUCTION

The next generation of MPSoC are expected to accommodate multiple processors on the same chip, which increases the performance but increases at the same time the power consumption. As technology is moving quickly into the nano-meter domain, the energy dissipated by MPSoC is more and more significant. So, it becomes harder to guarantee sufficient battery life time. Thus, balancing performance and power becomes a major design challenge. Therefore, the ITRS roadmap lists power reduction as the second most important design challenge[3].

To crack the power problem while maintaining acceptable productivity requires power methodologies that support IP reuse and abstraction. They have also to be automatically integrated in simulation models. It is interesting also for power methodologies to be non-intrusive in the original system's simulation code. In fact, without this last constraint, it is hard to imagine using multi provider IP libraries that are absolutely necessary in modern systems design.

The objective of this work is to provide designers by a complete and systematic performance and power estimation methodology for MPSoC design. This methodology has three major characteristics:

- Automatic: to reach time-to-market constraints, the power consumption estimation methodology has to be automatic, fast and not tedious for users,
- Allowing reuse and not intrusive: the solution must support heterogeneous and multi providers IP reuse. It is also necessary to keep the original system's code "clean", so to be non-intrusive. This is particularly important because, generally, designers do not have source code of IPs provided by foreign companies.
- Easy integration in a complete design flow: to be efficient, a power consumption methodology must be easily used in complete and automatic design flows.

This paper is organized as follows: in the next section, we explain and classify the main approaches of the literature. Details of the proposed methodology are given in section 3. In section 4 a significant case study is presented and the simulation results are analyzed. Section 5 concludes this article and gives some of the perspectives of this work.

2. RELATED WORK

Although the accuracy of consumption estimation tools at low levels [4], the simulation requires a high amount of time. This limits represent an obstacle to apply this approach to complex systems. 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, 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 pertinent activities and the occurrences of these activities during the simulation. The consumption of the main internal units is estimated using power macro-models, produced during lower-level simulations. The power model is integrated into the simulator. During the simulation, the consumption is calculated in every cycle, which permits accurate estimates. The contributions of the internal unit activities are calculated and added together during the simulation. Among tools described with this approach, we find SimplePower[12] and MPARM[5]. The main drawback of such tools is that they use intrusive approaches. A solution for this problem is to use standalone estimators connected between components. Among tools using this approach, we cite UNISIM[6], which is a simulation framework that offers cycle accurate energy estimation. The estimation is based on shadow modules. A shadow module is connected to the module that is being monitored. Whenever an operation is correctly executed by the monitored module, the shadow module is notified. This approach has many similar points with ours: both use separate modules in order to estimate power consumption, both use the inputs of the monitored modules for power estimation and both are adaptable for many hardware configurations. The disadvantage of shadow approach is that it uses a specific communication protocol between hardware components and between these components and the estimation modules. Besides, 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 have to be changed in order to support this functionality.

An interesting approach to reduce the design time of MPSoC is the Model Driven Engineering approach. MDE revolves around three focal concepts. Models, Metamodels and Model Transformations. 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) [9]; 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 levels models and help to keep the different models synchronized. At each intermediate level, implementation details are added to the MTs. A MT is a compilation process that transforms a source model into a target model and allows to move from an abstract model to a more detailed model. Usually, the initial high level models contain only domain specific concepts, while technological concepts are introduced seamlessly in the intermediate levels. The source and target models each conform to their respective metamodels thus respecting exogenous transformations [7]. A model transformation 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.

The solution proposed in this paper makes a trade-off between the speed of simulation using a high level of abstraction and an acceptable accuracy compared with lower levels. Besides, it is non-intrusive, 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. CONSUMPTION ESTIMATION AT THE CABA LEVEL

3.1 The Non-Intrusive Approach

As we said in the related work section, at the CABA level, the total energy consumption of a system is obtained by adding the consumptions of system components together. To provide accurate estimation, two kinds of consumption

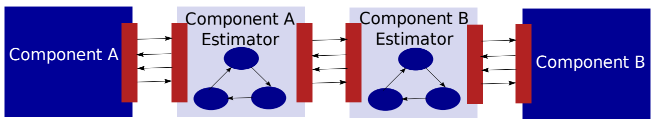


Figure 1: Consumption Estimators

are considered: dynamic consumption related to component activity (e.g. read/write operations), and static consumption related to leakage currents. For a long time, dynamic power consumption has been considered more significant then static consumption. However,this point of view changed with the advent of new sub-micron technologies, for which both types of consumption have their degree of importance. The energy consumption of a hardware component follows this equation:

$$E = \sum_i N_i * C_i$$

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 cost of a unit of the activity i or of a cycle of inactivity. In a previous work [1], we developed energy models for the main components of the SoCLib library: the processor, the cache memory, the shared SRAM memory and the interconnection network. We determined each component pertinent activities and we measured their unit costs using low level tools. To determine the occurrences of each activity, a counter was inserted in the source code of the components for each activity kind. Each counter is incremented during the simulation if the corresponding activity occurs during the current cycle. 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 SoCLib 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.

This approach gives fast accurate results but has the inconvenience to modify the source code of IPs, which is supposed to be maintained clean. Besides, generally, designers do not have this code if the IPs are provided by foreign companies. So, how can we determine the occurrences of the activities without being intrusive? The solution is to detect these activities through the signals that the components exchange. For this reason, we connect a number of consumption estimation modules between the components in order to determine their activities from the requests and responses that they exchange. Figure 1 shows that to determine the consumption of two linked components, we need a consumption estimator for each one. We don't use only one estimator to estimate the consumption of both components because separating estimators allows their reuse with different architectures.

The components of SoCLib library have VCI[11] interfaces besides a specific protocol for the communication between processors and caches. We chose the OCP[8] protocol for the interfaces of our estimators. This protocol allows

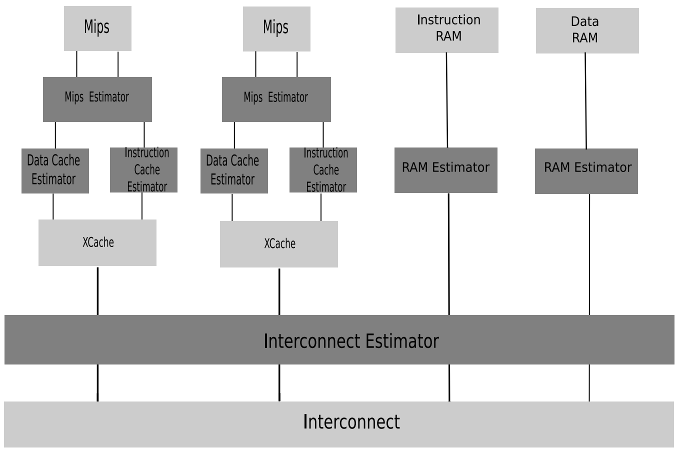


Figure 2: Example of simulated architectures with consumption estimators

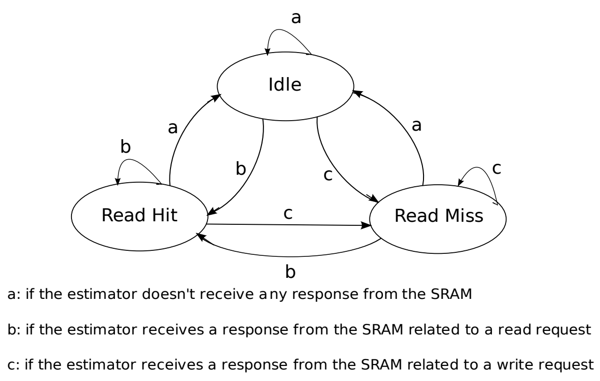

Idle
Read Hit
Read Miss
a
b
c
a: if the estimator doesn't receive any response from the SRAM
b: if the estimator receives a response from the SRAM related to a read request
c: if the estimator receives a response from the SRAM related to a write request

Figure 3: The SRAM Estimator's FSM

any type of communication and so it can be used for estimators connected between processors and caches as well as between the other VCI components. This solution permits the use of a single standard protocol allowing the integration of the estimators in different architectures. Therefore, we used wrappers to make the evaluated components compliant with the OCP protocol so we can connect estimators to them.

3.2 Consumption estimators for SoCLib

In this study, we developed consumption estimators for certain SoCLib components: the MIPSR3000, the cache memory, the SRAM shared memory and the interconnect. Figure 2 shows an example of a SoCLib architecture with integrated consumption estimators. The description of an estimator at the CABA level is based on a power state machine. The power state machine contains relevant power states of a component and transitions between them. A power state is related to an activity of the 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 these signals, it is possible to determine the current activity of a component. At every cycle, there is a transition in the power state machine. In each transition, there is an energy consumption related to the target state and so the corresponding occurrence counter is incremented.

For 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[5]. Thus, the SRAM Estimator's FSM is composed of three states: Read, write and Idle as shows Figure 3. 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 doesn't send any response, that means that it's inactive, which corresponds to the Idle state in the FSM. So, the SRAM total energy follows this equation:

$$E_{SRAM} = n_{read}.E_{read} + n_{write}.E_{write} + n_{idle}.E_{idle}$$

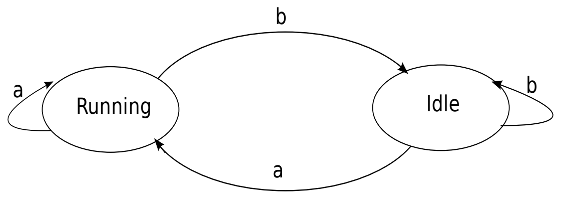
where n_{read}, n_{write} and n_{idle} are respectively the number of occurrences of a read access, a write access and a cycle of inactivity of the SRAM. These occurrences are given by the counters associated to each state in the FSM. E_{read}, E_{write} and E_{idle} are respectively the costs of a read 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 and were determined using the ELDO simulator[1].

Figure 4 shows the FSM of an estimator of a MIPSR3000 processor. The FSM of this estimator has two states: a state of activity where the MIPS executes an instruction and a state of inactivity where it waits for cache responses or does nothing. This estimator is connected between the processor and the cache. The detection of the MIPS activities is based on the responses that it receives from the instruction and data caches. If the MIPS estimator receives a cache miss it deduces that the MIPS will be inactive because of this miss, so it increments the idle counter. This counter is incremented also when the MIPS estimator doesn't receive any response from the caches, in this case the processor is not executing any instruction. If the MIPS estimator receives a valid response from the cache, and both cache miss signals are set to false, it deduces that the MIPS is executing an instruction and increments the running counter.

The processor's energy consumption in the active state depends on the instruction to be executed. The set of all the instructions energies will constitute the processor energy model, and the cost of each instruction can be determined from low level measurements. This approach has been used by several other authors [5] [10]. For instance Sinha [10] demonstrated that for the StrongARM SA-1100 processor, the maximum current variation between instructions during the execution of a program is only 8%. The Hitachi HS-4 processor behaves in the same way [10]. Consequently, for a processor with a relatively simple architecture, like the MIPS R3000, a consumption model that considers only the average current per instruction is sufficient. Thus, we can consider an average running energy cost. So, the energy consumed by the MIPS follows this equation:

$$E_{MIPS} = n_{running}.E_{running} + n_{idle}.E_{idle}$$

To estimate the energy consumption of more complex processors, we have to take into account the consumption of



a: if the mips estimator receives a response from the data or the instruction cache and there is no cache miss in both caches

b: if the mips estimator receives a cache miss signal or there is no response from both caches

Figure 4: The MIPS Estimator's FSM

each instruction. The power state machines were also used to represent the consumption of the instruction and data cache, and the interconnect.

3.3 The MDE Approach

Our work aims to provide MPSoC designers with an environment that allows them to generate automatically consumption estimators from estimator models. To modify an estimator, they are not obliged to write long code lines, they only need to enter a few modifications in the estimator model and relaunch the generation process to obtain the desired result. Thus, Model Driven Engineering is a solution to make power consumption estimation a fast and not tedious work for users in order to respect the time-to-market constraints. The integration of the consumption estimation in the design flow of the Gaspard framework[2], which is based on MDE too, allows thus fast architectures exploration. To implement this integration, we used the same MDE tools as Gaspard, namely Papyrus for graphical modeling, QVTO for model transformations and JET for code generation.

To facilitate the modeling of the estimators for the users, we have designed a UML profile so they can have a graphical view of the estimator structure, FSM and deployment. Here we proposed a new profile because there are notions that we need and that we don't find in standard UML profiles such as MARTE that Gaspard uses to model MPSoC. In fact, what we need here is to model the internal structure of a component (the estimator) to be generated later, while MARTE permits to model architectures composed of components that already exist. Among the notions that we don't find in MARTE is the State Machine that is a main concept in our modelling.

The model elaborated using the proposed profile must be transformed to a target model compliant to the consumption estimation metamodel. The code of the estimators is then generated using the target model. Figure 5 shows the consumption estimation metamodel. It is based on an "EstimationModel" that 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 interfaces or any other type of interfaces. Provided that the "ConsumptionEstimator" will be connected between hardware components, it has to transmit signals between them. This work is done by transmitting signals between each two associated interfaces, this information is given by the association "isAssociatedTo". The interfaces have "InterfaceTypes" such as an OCP master or an OCP slave. To each "InterfaceType" 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 numbers of words and lines in the cache. In certain cases, the estimator has to save some data in variables 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's 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's 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. 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 an "effect". An "Activity" manipulates several "EstimatorElements". The source code path of an activity is given by the "CodeFile" metaclass . That 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.

Thanks to the profile, the MPSoC designer can elaborate easily the estimator models. Figure 6 shows the state machine diagram corresponding to the instruction cache estimator's FSM. Here, we have omitted to display the guards and effects of transactions on the connectors because they are long expressions and would make the figure illegible. This information can be viewed in the properties view when clicking on the connectors. Figure 6 shows the guard and effect of a transition between the IDLE state and the Read Miss state(RUPDT), which corresponds to the update of the cache content after a read miss.

After the Model To Model transformation. The target model is transformed into a SystemC code. 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, MDE approach saves us a long coding time.

4. SIMULATION RESULTS

Gaspard 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 TLM[2]. For the moment, the code generation of a whole architecture at the CABA level is not supported yet but it is not very different from the code generation at the TLM level. The integration of the consumption estimators in the architecture in a UML model either in CABA or TLM levels can be done by indicating that a component will be monitored by an estimator in the deployment diagram. For Example, the deployment diagram can

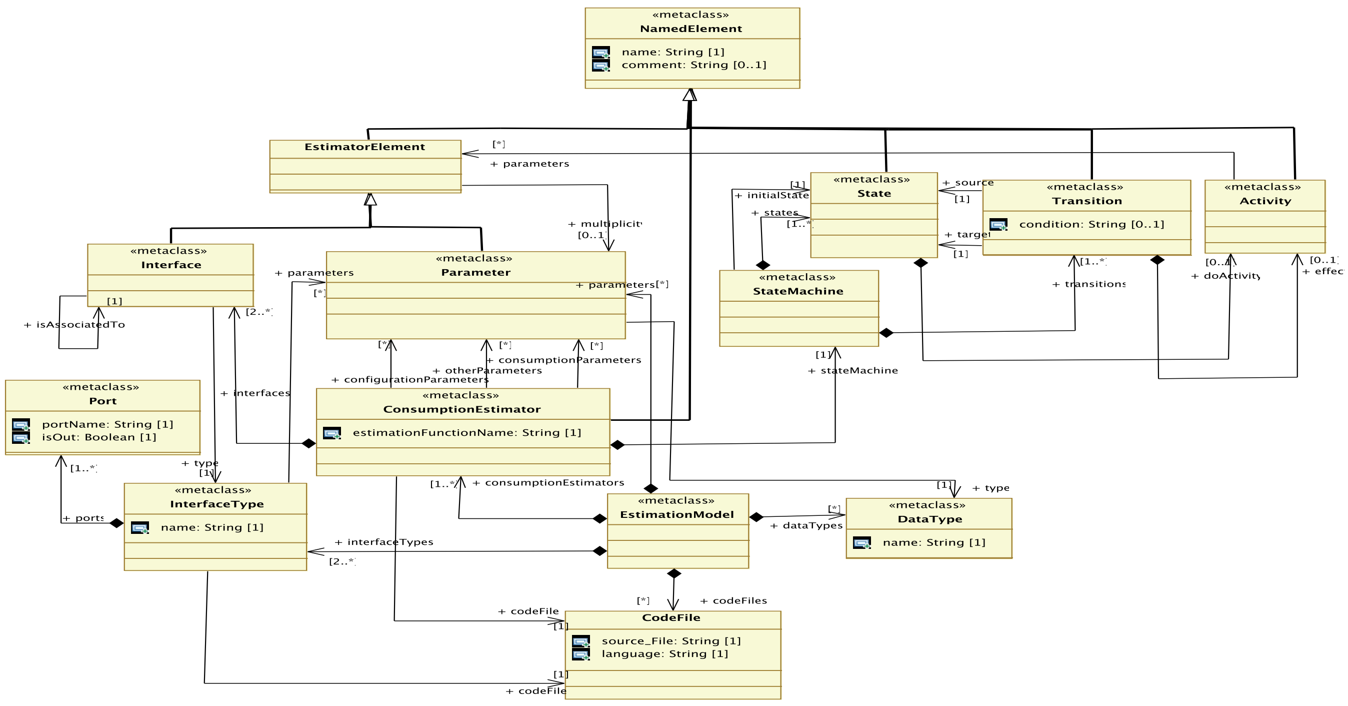
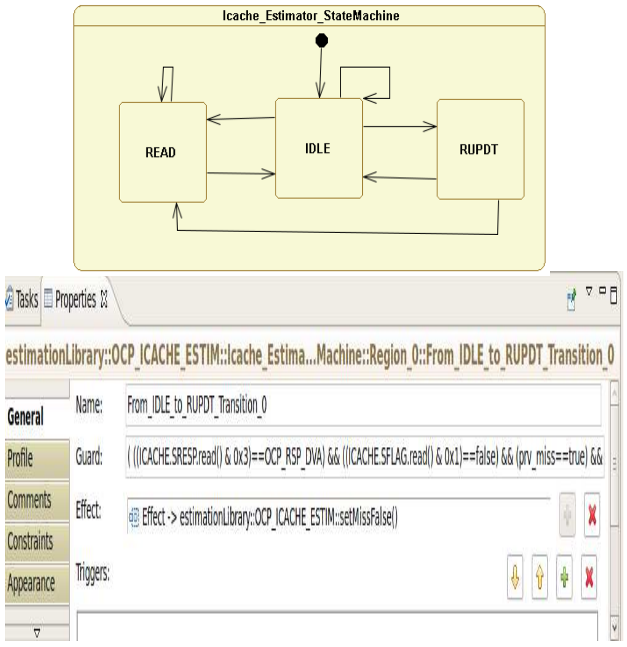


Figure 5: Consumption Estimation Metamodel

indicate that the RAM memory that is used 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 normally connected. That is what we will implement in future works. For now, to validate our approach we have to use it in an architecture described at the CABA level. Provided that the connections between the components and the estimators cannot be done yet at the CABA level using Model Driven Engineering, we used instead some script shells that generate different architectures giving as parameters the number of processors, the cache size, etc. So, the previously developed energy models were integrated into the SoCLib architectural simulator in order to profit from a fast architectural exploration environment for MPSoC design. To validate our approach, we used a Visiophony application for the UMTS network. For this application, we chose a minimal resolution using the QCIF format (144*176 pixels). The coding standard chosen was the H.263, adapted for Visiophony and Videoconference applications. For this paper, we evaluated only the DCT task to validate our approach. Figure 2 shows an example of the architectures used in the simulations. The hardware components used in these architectures are MIPSR3000 processors, 16KB SRAM memories and micro-networks. 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. To prove the accuracy of our approach, we have to compare its consumption results with those of the intrusive approach followed in our previous work [1]. Therefore, we executed the DCT task on different architectures varying the number of processors and the cache size, and we did the same simulations using the previous and the new approaches for consumption estimation.

The results of the simulations showed that the non-intrusive approach gave estimation values which are very close from those of the intrusive one(the consumption results of the previous approach can be found in[1]) with a difference that does not exceed 0.3% as shows Figure 7. The difference between the results given by the two approaches shows that there are some internal details that we cannot detect only through the interfaces. In fact, in this case, the difference is due to the fact that the consumption of the request FIFO queue of the cache is not considered in the new approach but in the previous one. This consumption corresponds to the consumption of a memory which is the FIFO here. As we have seen, to determine the consumption of a memory, we have to determine the number of the accesses to that memory and its size. The number of the requests written into the FIFO can be given by the read misses, the uncacheable reads and the write counters of the cache estimator, but the information that we don't have here is the maximum size of the FIFO. This can be only known by accessing the source code of the cache IP, which is against the principles of the non-intrusive approach. 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 to get accurate results because it took into account the main pertinent and the most consuming activities of the monitored hardware components.



estimationLibrary::OCP_ICACHE_ESTIM::Icache_Estima...Machine::Region_0::From_IDLE_to_RUPDT_Transition_0

Name: From_IDLE_to_RUPDT_Transition_0

Guard: (((ICACHE.SRESP.read() & 0x3)==OCP_RSP_DVA) && ((ICACHE.SFLAG.read() & 0x1)==false) && (prv_miss==true) &&

Effect: Effect -> estimationLibrary::OCP_ICACHE_ESTIM::setMissFalse()

Triggers:

Figure 6: MIPS Estimator's FSM model

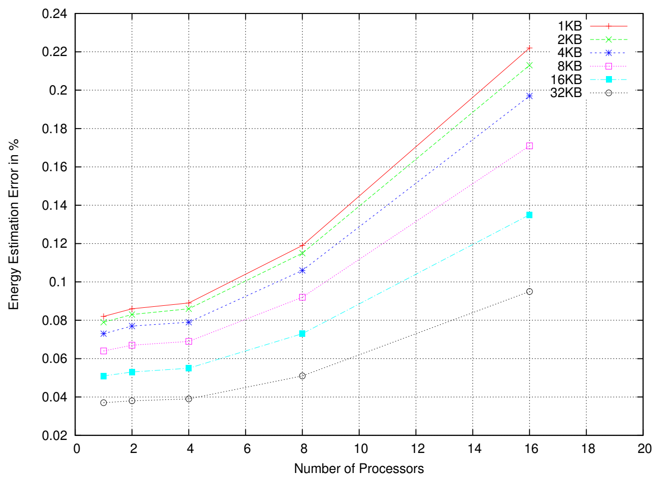


Figure 7: Variation in energy consumption estimation error in terms of cache size and number of processors

5. CONCLUSION

In this study, we enhanced our previous work in energy estimation, which used an approach that is intrusive in IP code sources in order to detect the activities of a hardware component and determine their consumption costs. In this paper, we have presented a non-intrusive approach using standalone estimators for energy consumption. These modules can be connected between hardware components in order to estimate their consumption. The implementation and the test of this approach followed these steps: designing consumption estimators at the CABA level using the Gaspard framework, generating automatically these estimators thanks to MDE tools, and integrating them in the SoCLib simulator to validate their estimation results. The components studied here are simple and their activities are fully detectable through the signals that they exchange. We plan to adapt the same approach for more complex architectures in order to obtain an acceptable accuracy and to validate our approach with different applications. Besides, we intent to apply the same approach for higher abstraction levels such as TLM.

6. REFERENCES

[1] R. B. Atitallah, S. Niar, A. Greiner, S. Meftali, and J. Dekeyser. Estimating energy consumption for an mpsoc architectural exploration. In *ARCS*, Frankfurt, Germany, 2006.

[2] Gamatié and al. A model driven design framework for high performance embedded systems. Technical report, INRIA, 2008.

[3] ITRS. Itrs 2007 edition. http://www.itrs.net/.

[4] R. P. Llopis and K. Goossens. The petrol approach to high-level power estimation. In *International Symposium on Low Power Electronics and Design*, USA, 1998.

[5] M. Loghi, M. Poncino, and L. Benini. Cycle-accurate power analysis for multiprocessor systems-on-a-chip. In *GLSVLSI*, Boston, Massachusetts, USA, 2004.

[6] 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.

[7] T. Mens and P. V. Gorp. P. a taxonomy of model transformation. *Proceedings of the International Workshop on Graph and Model Transformation, GraMoT 2005*, 152:125–142, 2006.

[8] OCP. *OCP International Partnership.* http://www.ocpip.org/home.

[9] S. Sendall and W. Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20(5):42–45, 2003.

[10] A. Sinha and A. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Proceedings of the 38th DAC Conference*, 2000.

[11] VCI. *Legacy Documents of the VSI Alliance.* http://vsi.org/.

[12] W. Ye, N. Vijaykrishnan, M. Kandemir, , and M. Irwin. The design and use of simplepower: A cycle-accurate energy estimation tool. In *DAC*, Los Angelos, California, 2000.