A High-Level Power Model for MPSoC on FPGA

Roberta Piscitelli and Andy D. Pimentel
Computer Systems Architecture group
Informatics Institute, University of Amsterdam, The Netherlands
Email: {r.piscitelli,a.d.pimentel}@uva.nl

***Abstract*—This paper presents a framework for high-level power estimation of multiprocessor systems-on-chip (MPSoC) architectures on FPGA. The technique is based on abstract execution profiles, called event signatures, and it operates at a higher level of abstraction than, e.g., commonly-used instruction-set simulator (ISS) based power estimation methods and should thus be capable of achieving good evaluation performance. As a consequence, the technique can be very useful in the context of early system-level design space exploration. We integrated the power estimation technique in a system-level MPSoC synthesis framework. Subsequently, using this framework, we designed a range of different candidate architectures which contain different numbers of Microblaze processors and compared our power estimation results to those from real measurements on a Virtex-6 FPGA board.**

I. Introduction

The complexity of modern embedded systems, which are increasingly based on MultiProcessor-SoC (MPSoC) architectures, has led to the emergence of system-level design. System-level design tries to cope with the design complexity by raising the abstraction level of the design process. Here, a key ingredient is the notion of high-level modeling and simulation in which the models allow for capturing the behavior of system components and their interactions at a high level of abstraction. These high-level models minimize the modeling effort and are optimized for execution speed. Consequently, they facilitate early architectural design space exploration (DSE).

An important element of system-level design is the high-level modeling for architectural power estimation. This allows to verify that power budgets are approximately met by the different parts of the design and the entire design, and evaluate the effect of various high-level optimizations, which have been shown to have much more significant impact on power than low-level optimizations [10].

The traditional practice for embedded systems evaluation often combines two types of simulators, one for simulating the programmable components running the software and one for the dedicated hardware parts. However, using such a hardware/software co-simulation environment during the early design stages has major drawbacks: (i) it requires too much effort to build them, (ii) they are often too slow for exhaustive explorations, and (iii) they are inflexible in quickly evaluating different hardware/software partitionings. To overcome these shortcomings, a number of high-level modeling and simulation environments have been proposed in recent years. An example is our Sesame system-level modeling and simulation environment [19], which aims at efficient design space exploration of embedded multimedia system architectures.

Until now, the Sesame framework has mainly focused on the system-level performance analysis of multimedia MPSoC architectures. So, it did not include system-level power modeling and estimation capabilities. In [20], we initiated a first step towards this end, however, by introducing the concept of *computational event signatures*, allowing for high-level power modeling of microprocessors (and their local memory hierarchy). This signature-based power modeling operates at a higher level of abstraction than commonly-used instruction-set simulator (ISS) based power models and is capable of achieving good evaluation performance. This is important since ISS-based power estimation generally is not suited for early DSE as it is too slow for evaluating a large design space: the evaluation of a single design point via ISS-based simulation with a realistic benchmark program may take in the order of seconds to hundreds of seconds. Moreover, unlike many other high-level power estimation techniques, the signature-based power modeling technique still incorporates an explicit micro-architecture model of a processor, and thus is able to perform micro-architectural DSE as well.

In this paper, we extend the aforementioned signature-based power modeling work, and we present a full system-level MPSoC power estimation framework based on the Sesame framework, in which the power consumption of all the system components is modeled using signature-based models. The MPSoC power model has been incorporated into Daedalus, which is a system-level design flow for the design of MPSoC based embedded multimedia systems [14], [17]. Daedalus offers a fully integrated tool-flow in which system-level synthesis and FPGA-based system prototyping of MPSoCs are highly automated. This allows us to quickly validate our high-level power models against real MPSoC implementations on FPGA.

In the next section, we briefly describe the Sesame framework. Section 3 introduces the concept of *event signatures* and explains how they are used in the power modeling of architectures. Section 4 gives an overview of our MPSoC power modeling framework and the different components used for modeling processors, memories and communication channels. Section 5 presents a number of experiments in which we compare the results from our models against real measurements of real MPSoC implementations on a Virtex-6 FPGA board. In Section 6, we describe related work, after which Section 7 concludes the paper.

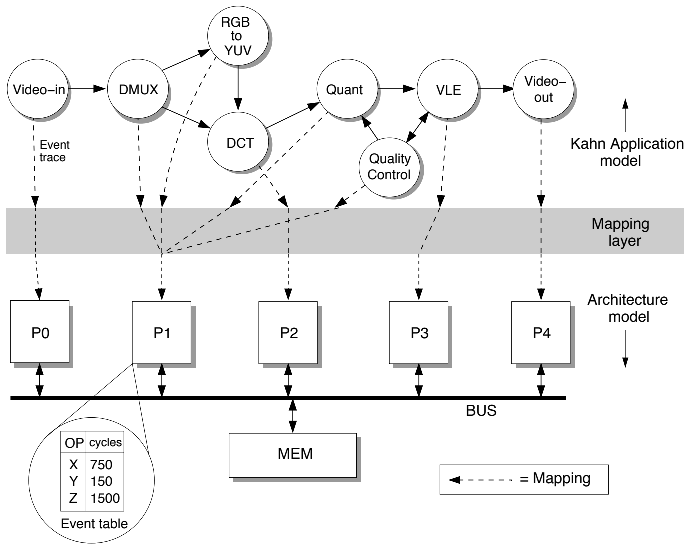


Fig. 1. The Sesame system-level simulation environment

II. THE SESAME ENVIRONMENT

Sesame is a modeling and simulation environment for the efficient design space exploration of heterogeneous embedded systems. Using Sesame, a designer can model embedded applications and MPSoC architectures at the system level, map the former onto the latter, and perform application-architecture co-simulations for rapid performance evaluations. Based on these evaluations, the designer can further refine (parts of) the design, experiment with a different hardware/software partitionings, perform simulations at multiple levels of abstraction, or even have mixed-level simulations where architecture model components operate at different levels of abstraction. To achieve this flexibility, the Sesame environment uses separate application and architectures models. According to the Y-chart approach [19], an application model – derived from a target application domain – describes the functional behavior of an application in an architecture-independent manner. This model correctly expresses the functional behavior, but is free from architectural issues, such as timing characteristics, resource utilization, or bandwidth constraints. Next, a platform architecture model – defined with the application domain in mind – defines architecture resources and captures their performance constraints. Finally, an explicit mapping step maps an application model onto an architecture model for co-simulation, after which the system performance can be evaluated quantitatively. The layered infrastructure of Sesame is illustrated in Figure 1.

For application modeling, Sesame uses the Kahn Process Network (KPN) model of computation [9] in which parallel processes implemented in a high-level language communicate with each other via unbounded FIFO channels. Hence, the KPN model unveils the inherent task-level parallelism available in the application and makes the communication explicit. Furthermore, the code of each Kahn process is instrumented with annotations describing the application's computational actions, which allows to capture the computational behavior of an application. The reading from and writing to FIFO channels represent the communication behavior of a process within the application model. When the Kahn model is executed, each process records its computational and communication actions, and generates a trace of *application events*. These application events are an abstract representation of the application behavior and are necessary for driving an architecture model. Application events are generally coarse grained, such as *read(channel id, pixel block)* or *execute(DCT)*.

An architecture model simulates the performance consequences of the computation and communication events generated by an application model. It solely accounts for architectural (performance) constraints and does not need to model functional behavior. This is possible because the functional behavior is already captured by the application model, which drives the architecture simulation. The timing consequences of application events are simulated by parameterizing each architecture model component with an event table containing operation latencies. The table entries could include, for example, the latency of an *execute(DCT)* event, or the latency of a memory access in the case of a memory component. With respect to communication, issues such as synchronization and contention on shared resources are also captured in the architecture model.

To realize trace-driven co-simulation of application and architecture models, Sesame has an intermediate mapping layer with two main functions. First, it controls the mapping of Kahn processes (i.e., their event traces) onto architecture model components by dispatching application events to the correct architecture model component. Second, it makes sure that no communication deadlocks occur when multiple Kahn processes are mapped onto a single architecture model component. In this case, the dispatch mechanism also provides various strategies for application event scheduling.

Extending the Sesame framework to also support power modeling of MPSoCs could be done fairly easily by adding power consumption numbers to the event tables. So, this means that a component in the architecture model not only accounts for the timing consequences of an incoming application event, but also accounts for the power that is consumed by the execution of this application event (which is specified in the event tables now). The power numbers that need to be stored in the event tables can, of course, be retrieved from lower-level power simulators or from (prototype) implementations of components. However, simply adding fixed power numbers to the event tables would be a rigid solution in terms of DSE: these numbers would only be valid for the specific implementation used for measuring the power numbers. Therefore, we propose a high-level power estimation method based on so-called event signatures that allows for more flexible power estimation in the scope of system-level DSE. As will be explained in the next sections, signature-based power estimation provides an abstraction of processor activity and communication in comparison to traditional ISS-based power models, while still incorporating an explicit micro-architecture model and thus being able to perform micro-architectural DSE.

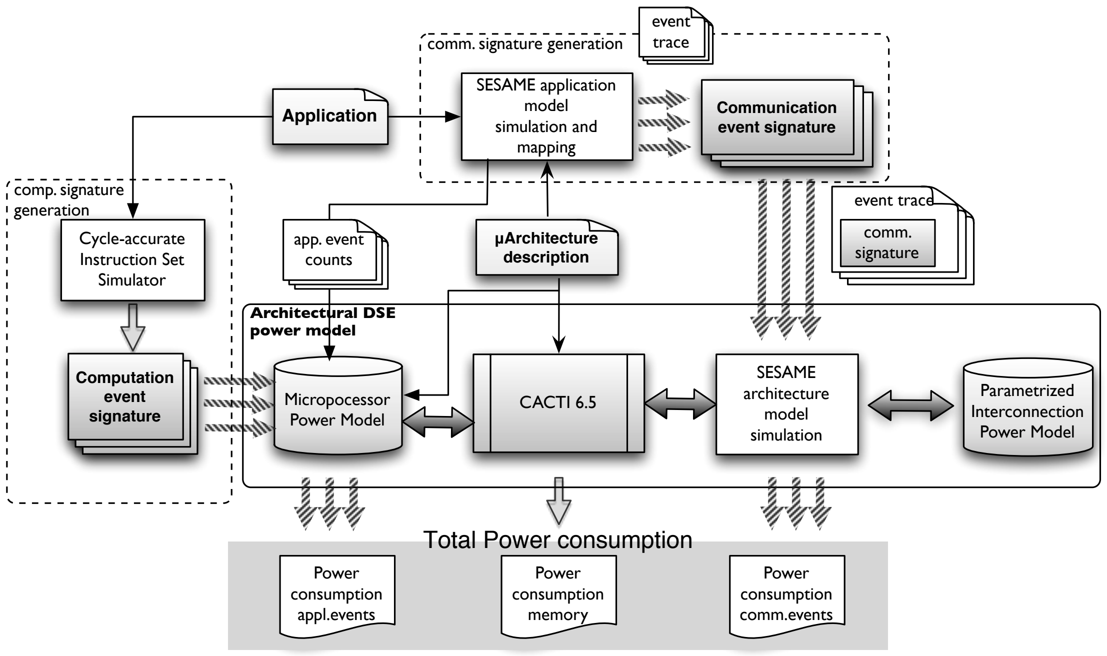

event trace
comm. signature generation
Application
SESAME application model simulation and mapping
Communication event signature
comp. signature generation
Cycle-accurate Instruction Set Simulator
app. event counts
µArchitecture description
event trace
comm. signature
Computation event signature
Architectural DSE power model
Micropocessor Power Model
CACTI 6.5
SESAME architecture model simulation
Parametrized Interconnection Power Model
Total Power consumption
Power consumption appl.events
Power consumption memory
Power consumption comm.events

Fig. 2. System-level Power estimation framework

III. Event signatures

An event signature is an abstract execution profile of an application event that describes the computational complexity of an application event (in the case of computational events) or provides information about the data that is communicated (in the case of communication events). Hence, it can be considered as meta-data about an application event.

A. Computational events signatures

A computational signature describes the complexity of computational events in a (micro-)architecture independent fashion using an Abstract Instruction Set (AIS) [20]. Currently, our AIS is based on a load-store architecture and consists of *instruction classes*, such as Simple Integer Arithmetic, Simple Integer Arithmetic Immediate, Integer Multiply, Branch, Load, and Store. The high level of abstraction of the AIS should allow for capturing the computational behavior of a wide range of processors with different instruction-set architectures. To construct the signatures, the real machine instructions of the application code represented by an application event (derived from an instruction set simulator as will be explained below) are first mapped onto the various AIS instruction classes, after which a compact execution profile is made. This means that the resulting signature is a vector containing the instruction counts of the different AIS instruction classes. Here, each index in this vector specifies the number of executed instructions of a certain AIS class in the application event. We note that the generation of signatures for each application event is a one-time effort, unless e.g. an algorithmic change is made to an application event's implementation.

To generate computational signatures, each Kahn application process is simulated using a particular Instruction Set Simulator (ISS), depending on the class of target processor the application will be mapped on. For example, we currently use ISSs from the SimpleScalar simulator suite [2] for the more complex multiple-issue processors, while we deploy the Microblaze cycle-accurate instruction-set simulator provided by Xilinx for the more simple soft cores. Using these ISSs, the event signatures are constructed – by mapping the executed machine instructions onto the AIS as explained above – for every computational application event that can be generated by the Kahn process in question. The event signatures act as input to our parameterized microprocessor power model, which will be described in more detail in the next section. For each signature, the ISS may also provide the power model with some additional micro-architectural information, such as cache miss-rates, branch misprediction rates, etc. In our case, only instruction and data cache miss-rates are used. As will be explained later on, the microprocessor power model subsequently uses a micro-architecture description file in which the mapping of AIS instructions to usage counts of micro-processor components is described.

B. Communication event signatures

In Sesame, the Kahn processes generate *read* and *write* communication events as a side effect of reading data from or writing data to ports. Hence, communication events are automatically generated. For the sake of power estimation, the communication events are also extended with a signature, as shown in Figure 3. A communication signature describes the complexity of transmitting data through a communication channel (e.g., FIFO, Memory Bus, PLB Bus) based on the dimension of the transmitted data and the statistical distribution of the contents of the data itself. For the latter, we use the *average Hamming distance* of the coarse-grained data communications. More specifically, we calculate the average Hamming distance of the data words within the data chunk communicated by a *read* or *write* event (which could be, e.g., a pixel block, or even an entire image frame), after which the result is again averaged with Hamming distance of the previous data transaction on the same communication channel. In this way, we can get information of the usage of the channel and the switching factor, which is related to the data distribution.

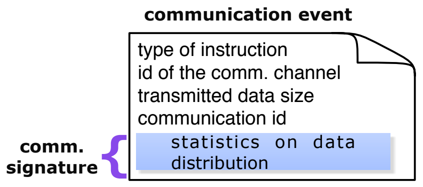


Fig. 3. Structure of communication events.

C. Signature-based, system-level power estimation

In Figure 2, the entire signature-based power modeling framework is illustrated. More specifically, a Kahn application model is mapped onto a given architecture model, and simulated with Sesame; during this stage, each process generates its own trace of application events, representing the workload that is imposed on the underlying MPSoC architecture model. The communication signature generation is mapping dependent: communication patterns change with different mappings. For every *read/write* event, the *average Hamming distance*, as explained in the previous subsection, is computed. This information is then integrated in the trace events, forming the communication signature. On the other side, the Kahn application processes for which a power estimation needs to be performed, are simulated using the ISS, constructing the event signatures (as explained in the previous section) for every computational application event that can be generated by the Kahn process in question. As will be explained in the next section, the microprocessor power model uses a micro-architecture description file in which the mapping of AIS instructions to usage counts of micro-processor components is described. The Sesame architecture model simulates the performance and power consequences of the computation and communication events generated by the application model. To this end, each architecture model component is parameterized with an event table containing the latencies of the application events it can execute (as explained in Section 2). Moreover, each architecture model component now also has an underlying signature-based power model. These models are activity-based for which the activity counts are derived from the different application events in the event traces as well as the signature information of the separate events. The total power consumption is then obtained by simply adding the average power contributions of microprocessor(s), memories and interconnect(s).

TABLE I
DIFFERENT POSSIBILITIES OF REUSING SIGNATURES IN DSE

comp. signatures	comm. signatures
μ-architectural exploration	μ-architectural exploration
mapping exploration (limited)	architectural exploration

The structure of the entire system-level power model is composed by separate and independent modules, which allow for the reuse of the different underlying component models as well as the generated signatures (as shown in Table I). For example, once computational signatures are generated for application events, it is possible to explore different micro-architectures executing the same application with the same mapping. Moreover, given the computational event signatures, it is also possible to do mapping exploration, limited to the case of homogenous systems. Regarding communication signatures, they can be reused for both micro-architectural and architectural exploration.

IV. POWER MODEL

We propose a high-level power estimation method based on the previously discussed event signatures that allows for flexible power estimation in the scope of system-level DSE. As will be explained in the subsequent subsections, signature-based power estimation provides an abstraction of processor (and communication) activity in comparison to e.g. traditional ISS-based power models, while still incorporating an explicit micro-architecture model and thus being able to perform micro-architectural DSE. The power models are based on FPGA technology, since we have incorporated these models in our system-level MPSoC synthesis framework Daedalus [17], which targets FPGA-based (prototype) implementations. The MPSoC power model is formed by three main building blocks, modeling the microprocessors, the memory hierarchy and the interconnections respectively. The model is based on the activity counts that can be derived from the application events and their signatures as described before, and on the power characteristics of the components themselves, measured in terms of LUTs used. In particular, we estimate through synthesis on FPGA the maximum number of LUTs used for each component. The resulting model is therefore a compositional power model, consisting of the various components (for which the models are described below) used in the MPSoC under study. In the remainder of this paper, we will focus on homogeneous systems, but the used techniques do allow the modeling and simulation of heterogeneous systems as well.

A. Interconnection Power model

In this section, we derive architectural-level parameterized, activity based power models for major network building blocks within our targeted MPSoCs. These include FIFO buffers, crossbar switches, buses and arbiters. The currently modeled building blocks – network components as well as processor and memory components – are all part of the IP library of our Daedalus synthesis framework [17], which allows the construction of a large variety of MPSoC systems. Consequently, all our modeled MPSoCs can actually be rapidly synthesized to and prototyped on FPGA, allowing us to easily validate our power models.

Our network power models are composed of models for the aforementioned network building blocks, for which each of them we have derived parameterized power equations. These equations are all based on the common power equation for CMOS circuits:

$$P_{interconnect} = V_{dd}^2 fC\alpha \tag{1}$$

where *f* is the clock frequency, V_{dd} the operating voltage, C the capacitance of the component and α is the average switching activity of the component respectively. The capacitance values for our component models are obtained through an estimation of the number of LUTs used for the component in question as well as the capacitance of a LUT itself. Here, we estimate the number of LUTs needed for every component through synthesis, after which the capacitance is obtained using the X-Power tool from Xilinx. The activity rate α is primarily based on the *read* and *write* events from the application event traces that involve the component in question. For example, for an arbiter component of a bus, the total time of read and write transactions to the bus (i.e., the number of *read* and *write* events that involve the bus) as a fraction of the total execution time is taken as the access rate (i.e., activity rate). Consequently, the power consumption of an arbiter is modeled as follows:

$$P_{arbiter} = \beta \times V_{dd}^2 \times f \times C_{LUT} \times n_{LUTs} \times access\_rate \tag{2}$$

where C_{LUT}, n_{LUTs}, f, V_{dd} are respectively the estimated capacitance of a LUT, the estimated number of LUTs needed to build the arbiter, the clock frequency and the operating voltage. β is a scaling factor obtained through pre-calibration of the model, and

$$access\_rate = \frac{T_{reads} + T_{writes}}{T_{total\_exec}}$$

Here, T_{reads} and T_{writes} are the total times spend on the execution of read and write transactions, respectively, and T_{total_exec} is the total execution time.

For communication channels like busses, not only the number of *read* and *write* events play a role to determine the activity factor, but also the data that is actually communicated. To this end, we consider the *Hamming Distance distribution* between the data transactions, as explained in the previous section on communication signatures. Thus, every communication trace event is carrying the statistical activity-based information of the channel from/to which the data is read/written. Consequently, for any activity (read/write of data) in the channel, the dynamic power of the interconnection is calculated according to technology parameters and the statistical distribution of the data transmitted. Hence, for every packet transmitted over the channel, the estimated power is computed in the following way:

$$P_{chan} = \beta \times V_{dd}^2 \times f \times C_{chan} \times n_{LUTs} \times Hamm\_dist(e) \tag{3}$$

where β, C_{chan}, f, V_{dd}, n_{LUTs} are again the scaling factor, estimated capacitance of the communication channel, clock frequency, the operating voltage, and number of LUTs needed to build the interconnection channel. The $Hamm\_dist(e)$ parameter is the average Hamming distance of the data transmitted in the *read/write* events. In our models, leakage power is calculated according to the estimated look-up tables needed to build a particular interconnection.

B. Memory Power model

For on-chip memory (level 1 and 2 caches, register file, etc.) and main memory, we use the analytical energy model developed in CACTI 6.5 [16] to determine the power consumption of read and write accesses to these structures. These power estimates include leakage power. The access rates for the processor-related memories, such as caches and register file, are derived from the computational signatures, as will be explained in the next subsection. Moreover, we use the cache missrate information provided by the ISS used to generate the computational signatures to derive the access counts for structures like the level-2 cache and the processor's load/store queue.

For the main memory and communication buffers, we calculate the access rate in the same fashion as for a network arbiter component as explained above: the communication application events are used to track the number of accesses to the memory. That is, the total time taken by read and write accesses (represented by the communication application events) to a memory as a fraction of the total execution time is taken as the access rate. Subsequently, the signal rate represents the switching probability of the signals. For every read/write event to the memory, the average Hamming distance contained in the communication event signature is extracted and the signal rate is calculated as follows:

$$signal\_rate = \gamma \times Hamm\_dist(e)$$

where the γ is again a scaling factor obtained through pre-calibration of the model.

C. Microprocessor Power model

The microprocessor model that underlies our power model is based on [20]. It assumes a dynamic pipelined machine, consisting of one arithmetic logical unit, one floating point unit, a multiplier and two levels of caches. However, this model can easily be extended to other processor models, by simply introducing new units. For the power model of the clock component, three sub-components are recognized: the clock distribution wiring, the clock buffering and the clocked node capacitance. We assume a H-tree based clock network using a distributed driver scheme (i.e. applying clock buffers).

The power consumption of a computational application event is calculated by accumulating the power consumption

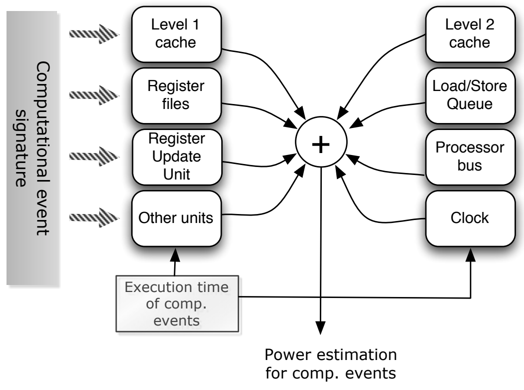


Fig. 4. Different components in the microprocessor power model

of each of the components that constitute the micro-processor power model, as shown in Figure 4. More specifically, the first step to calculate an application event's power consumption is to map its signature to usage counts of the various processor components. So, here it is determined how often e.g. the ALU (see Other Units in Figure 4), the register file and the level-1 instruction and data caches are accessed during the execution of an application event. The microprocessor power model uses an XML-based micro-architecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described. This micro-architecture description file also contains the parameters for our microprocessor power model, such as e.g. the dimensions and organization of memory structures (caches, register file, etc.) in the microprocessor, clock frequency, and so on. Clearly, this micro-architecture description allows for easily extending the AIS and facilitates the modeling of different micro-architecture implementations.

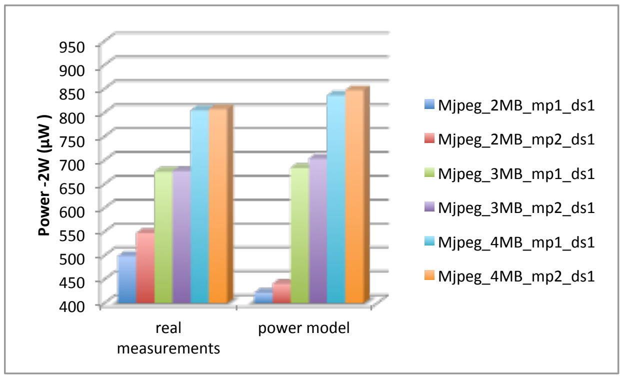
The above ingredients (the event signatures, additional micro-architectural information per signature such as cache statistics, and the micro-architecture description of the processor) subsequently allow the power model to produce power consumption estimates for each computational application event by accumulating the power consumption of the processor components used by the application event.

V. Validation

As mentioned before, we have integrated our power model into the Daedalus system-level design flow for the design of MPSoC based embedded multimedia systems [14], [17]. This allows for direct validation and calibration of our power model. By deploying Daedalus, we have designed several different candidate MPSoC configurations and compared our power estimates for these architectures with the real measurements. The studied MPSoCs contain different numbers of Microblaze processors that are interconnected using a crossbar network. The validation environment is formed by the architecture itself and an extra Microblaze. This extra Microblaze polls the power values in the internal measurement registers in our target Virtex-6 FPGA, and interfaces an I2C controller in the FPGA design with the I2C interface of the PMBus controller chip [1]. As we introduced an extra Microblaze in the design, the resulting power consumption of the system is scaled by a fixed factor, which is dependent on the measurement infrastructure. This is, however, not a problem since our primary aim is to provide high-fidelity rankings in terms of power behavior (which is key to early design space exploration) rather than obtaining near-perfect absolute power estimations [8]. Evidently, the additional power consumed by the extra Microblaze does not affect the fidelity of the rankings (i.e., the extra Microblaze exists in every MPSoC configuration), while the power measurements obtained are much more accurate compared to e.g. using a simulator [4].

The results of the validation experiments are shown in Figures 5 and 6. In these experiments, we mapped three different parallel multimedia applications onto the target MPSoCs: a Motion-JPEG encoder (Mjpeg), a Periodogram, which is an estimate of the spectral density of a signal, and a Sobel filter for edge detection in images. In addition, for each of the applications, we also investigated two different task mappings onto the target architectures. Here, we selected one "good" mapping, in terms of task communication, as well as a "poor" one for each application. That is, in the "good" mapping we minimize task communications, while in the "poor" one we maximize task communications. The experiments in Figures 5 and 6 apply the following notation: $app_{name}\_n_{proc}\_mapping_{type}$, where app_{name} is the application considered, n_{proc} indicates the number of processors used in the architecture (e.g., "3mb" indicates an MPSoC with 3 MicroBlaze processors), and $mapping_{type}$ refers to the type of mapping used. With respect to the latter, the tag *mp1* indicates the good mapping, while *mp2* refers to the poor mapping. For the Motion-JPEG application, we also considered two different data input sets: the first input set (*ds1*) is characterized by a high data-correlation, while the second input set (*ds2*) has a very low data correlation, in terms of measured average *Hamming distance distribution* of the input data. The power values in Figures 5 and 6 are scaled by a factor of 2W for the sake of improved visibility.

The results in Figures 5 and 6 show that our power model performs quite decently in terms of absolute accuracy. We observed an average error of our power estimations of around 7%, with a standard deviation of 5%. More important in the context of early design space exploration, however, is the fact that our power model appears to be very capable of estimating the right power consumption trends for the various MPSoC configurations, applications and mappings. We explicitly checked the fidelity of our estimations in terms of quality ranking of candidate architectures by ranking all design instances according to their consumed power for a specific application. Our estimates result in a ranking of the power values that is correct for every application we considered, therefore showing a high fidelity. This high-fidelity quality-ranking of candidate architectures thus allows for a correct candidate architecture generation and selection during the process of design space exploration.



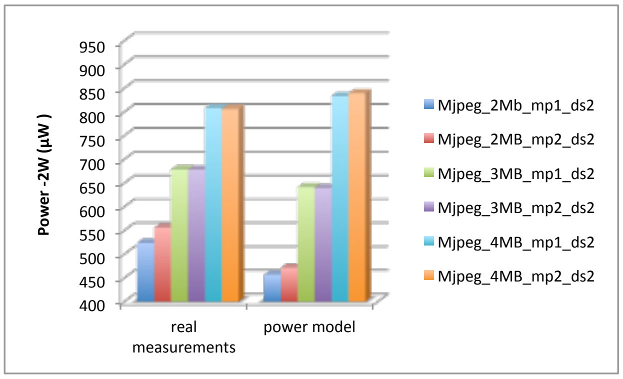
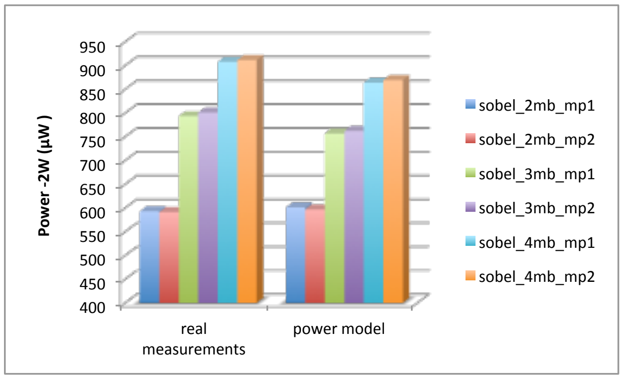


Fig. 5. Mjpeg application with input set ds1 (left) and input set ds2 (right)



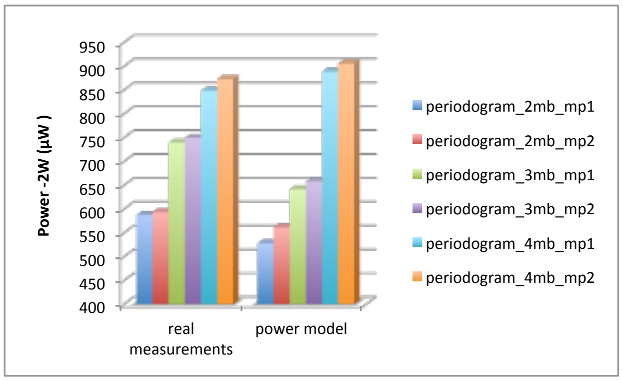


Fig. 6. Sobel filter (left) and Periodogram application (right)

Since every design point evaluation takes only few seconds, the presented power model offers remarkable potentials for quickly experimenting with different MPSoC architectures and exploring system-level design options during the very early stages of design.

VI. Related Work

There exists a fairly large body of related work on system-level power modeling of MPSoCs. For example, in [6], a SoC power estimation method has been presented that is based on a SystemC TLM modeling strategy. It adopts multi-accuracy models, supporting the switch between different models at run-time according to the desired accuracy level. Atitallah et. al. [3] use a stack of abstract models. The high-level Timed Programmer View model omits details related to the computation and communication resources and is used to prune the design space. The second Cycle-Accurate Bit-Accurate model is used for power estimation and platform configuration. In [15], a system-level cycle-based framework to model and design MPSoCs is presented. C++/SystemC based IP system modules can be wrapped to act as plug-ins, which are managed by the simulation kernel in a TLM fashion. To estimate power during a simulation, dedicated ports to each component are added, which communicate with the corresponding power model. [21] presents a simulation-based methodology for extending system performance modeling frameworks to also include power modeling. They demonstrate the use of this methodology with a case study of a real, complex embedded system. In [12], a power estimation framework for SoCs is presented, using power profiles to produce cycle accurate results. The SoC is divided in its building blocks (e.g. processors, memories, communication and peripherals) and the power estimation is based on the RTL analysis of each component.

Moreover, there also exist a considerable number of research efforts that only focus on the power modeling of the on-chip network of MPSoCs. Examples are [18], [7], [11], [13]. Many of the above approaches calibrate the high-level models with parameters extracted from RTL implementations, using low-level simulators for the architectural components. To the best of our knowledge, none of the existing efforts have incorporated the power models in a (highly automated) system-level MPSoC synthesis framework, allowing for accurate and flexible validation of the models. Instead, most existing works either use simulation-based validation (e.g. [6], [7], [11], [5], [18]), or validation by means of measurements on fixed target platforms (e.g. [21], [12]). Consequently, in general, related system-level MPSoC modeling efforts do also not target FPGA technology in their system-level power models.

VII. Conclusion

We presented a framework for high-level power estimation of multiprocessor systems-on-chip (MPSoC) architectures on

FPGA. The technique is based on abstract execution profiles called "event signatures", and it operates at a higher level of abstraction than, e.g., commonly-used instruction-set simulator (ISS) based power estimation methods and should thus be capable of achieving good evaluation performance. The model is based on the activity counts from the signatures, and from the power characteristics of the components themselves, measured in terms of LUTs used. The signature-based power modeling technique has been integrated in our Daedalus system-level MPSoC synthesis framework, which allows a direct validation and calibration of the power model. We compared the results from our signature-based power modeling to those from real measurements on a Virtex 6 FPGA board. These validation results indicate that our high-level power model achieves fairly accurate power estimates. As future work, we plan to perform more extensive validation experiments (e.g., using different interconnects and memory hierarchies) as well as to deploy the power model in real system-level DSE experiments.

Acknowledgment

This work has been partially supported by the MADNESS STREP-FP7 European Project. We would like to give special credits to Todor Stefanov, and Mohamed Bamakhrama for their support on implementing the Microblaze software driver for the PMBus controller.

References

[1] http://pmbus.org/specs.html.

[2] T. Austin, E. Larson, and D. Ernst. Simplescalar: An infrastructure for computer system modeling. *Computer*, 35:59–67, 2002.

[3] S. Dekeyser J.-L. B. Atitallah, R. Niar. MPSoC power estimation framework at transaction level modeling. *Microelectronics, 2007. ICM 2007.*, pages 245–248, 2007.

[4] J. Becker, M. Huebner, and M. Ullmann. Power estimation and power measurement of xilinx virtex fpgas: Trade-offs and limitations. In *SBCCI '03: Proceedings of the 16th symposium on Integrated circuits and systems design*, page 283, Washington, DC, USA, 2003. IEEE Computer Society.

[5] N. Eisley, V. Soteriou, and L. Peh. High-level power analysis for multi-core chips. In *CASES '06: Proc. of the 2006 int. conference on Compilers, architecture and synthesis for embedded systems*, pages 389–400, USA, 2006.

[6] D. Sciuto G. Beltrame and C. Silvano. Multi-accuracy power and performance transaction-level modeling. In *DATE '08: Proc. of the conference on Design, automation and test in Europe*, pages 788–791, USA, 2008.

[7] J. Hu and R. Marculescu. Energy-aware mapping for tile-based noc architectures under performance constraints. In *ASP-DAC '03: Proc. of the 2003 Asia and South Pacific Design Automation Conference*, pages 233–239, USA, 2003.

[8] P. Huang, M. Hashemi, and S. Ghiasi. System-level performance estimation for application-specific MPSoC interconnect synthesis. In *SASP '08: Proc. of the 2008 Symposium on Application Specific Processors*, pages 95–100, USA, 2008.

[9] G. Kahn. The semantics of a simple language for parallel programming. In *Proc. of the IFIP Congress 74*, 1974.

[10] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *DATE*, pages 423–428, 2009.

[11] S. Koohi, M. Mirza-Aghatabar, S Hessabi, and M. Pedram. High-level modeling approach for analyzing the effects of traffic models on power and throughput in mesh-based nocs. In *VLSID '08: Proc. of the 21st Int. Conference on VLSI Design*, pages 415–420, USA, 2008.

[12] I. Lee, H. Kim, P. Yang, S. Yoo, E. Chung, K. Choi, J. Kong, and S. Eo. Powervip: Soc power estimation framework at transaction level. In *Proc. of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 551–558, Piscataway, NJ, USA, 2006. IEEE Press.

[13] M. Loghi, L. Benini, and M. Poncino. Power macromodeling of MPSoC message passing primitives. *ACM Trans. Embed. Comput. Syst.*, 6(4):31, 2007.

[14] T. Stefanov A.D. Pimentel C. Erbas S. Polstra E.F. Deprettere M. Thompson, H. Nikolov. A framework for rapid system-level exploration, synthesis, and programming of multimedia MPSoCs. In *Proc. of the IEEE/ACM int. conference on Hardware/software codesign and system synthesis*, pages 9–14, USA, 2007.

[15] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. A modular approach to model heterogeneous MPSoC at Cycle Level. In *DSD '08: Proc. of the 2008 11th EUROMICRO Conf. on Digital System Design Architectures, Methods and Tools*, pages 158–164, USA, 2008.

[16] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *MICRO 40: Proc. of the 40th Annual IEEE/ACM Int. Symposium on Microarchitecture*, pages 3–14, USA, 2007.

[17] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere. Daedalus: Toward composable multimedia MPSoC design. In *Proc. of the 45th ACM/IEEE Int. Design Automation Conference (DAC '08)*, pages pp. 574–579, Anaheim, USA,, 2008.

[18] L. Ost, G. Guindani, L. Indrusiak, S. Maatta, and F. Moraes. Using abstract power estimation models for design space exploration in NoC-based MPSoC. *IEEE Design and Test of Computers*, 99(PrePrints), 2010.

[19] Andy D. Pimentel, Cagkan Erbas, and Simon Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Comput.*, 55(2):99–112, 2006.

[20] P. Stralen and A. D. Pimentel. A high-level microprocessor power modeling technique based on event signatures. In *Proc. of the IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '07)*, pages pp. 33–40, Salzburg, Austria, 2007.

[21] A. Varma, B. Jacob, E. Debes, I. Kozintsev, and P. Klein. Accurate and fast system-level power modeling: An xscale-based case study. *ACM Trans. Embed. Comput. Syst.*, 6(4):26, 2007.