Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 196984, 13 pages
doi:10.1155/2012/196984

Research Article

A Signature-Based Power Model for MPSoC on FPGA

Roberta Piscitelli and Andy D. Pimentel

Computer Systems Architecture Group, Informatics Institute, University of Amsterdam, 1098 XH Amsterdam, The Netherlands

Correspondence should be addressed to Roberta Piscitelli, r.piscitelli@uva.nl

Received 13 August 2011; Accepted 2 November 2011

Academic Editor: Luigi Raffo

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, for example, 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.

1. 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 [1].

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 cosimulation 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 [2], which aims at efficient design space exploration of embedded multimedia system architectures.

Until now, the Sesame framework has mainly been focused on the system-level performance analysis of multimedia MPSoC architectures. So, it did not include system-level power modeling and estimation capabilities. In [3], 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 microarchitecture 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 [4, 5]. 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.

2. 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 cosimulations for rapid performance evaluations. Based on these evaluations, the designer can further refine (parts of) the design, experiment with 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 [2], 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 cosimulation, 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 [6] 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 cosimulation 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

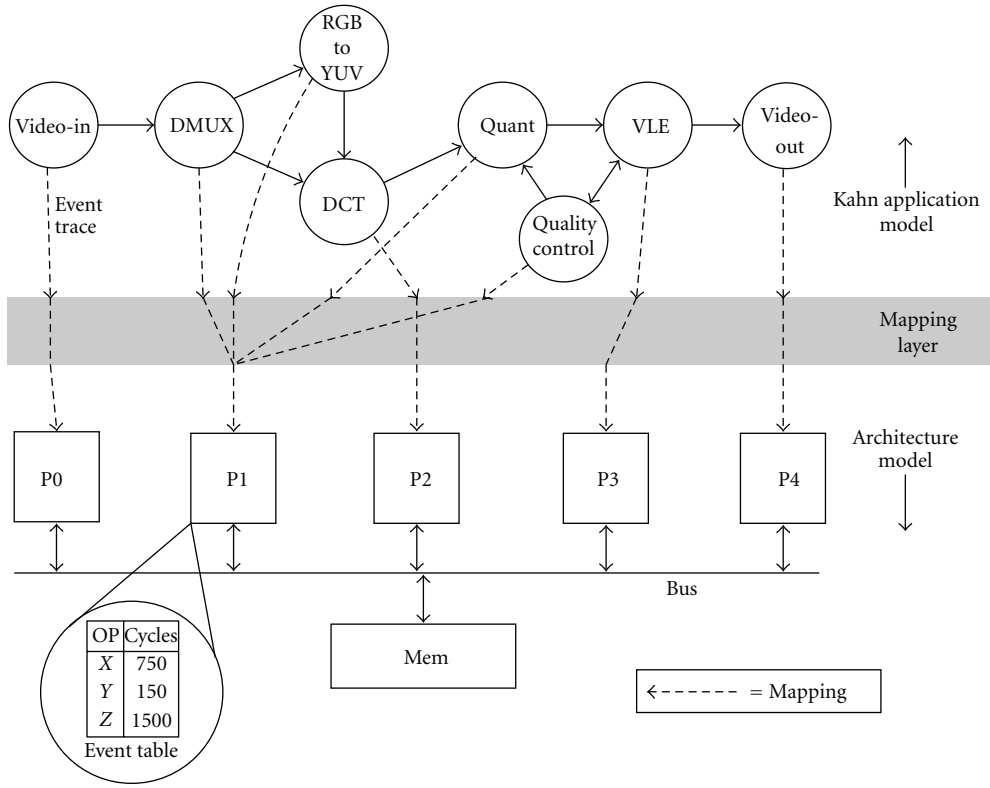

Figure 1: The Sesame system-level simulation environment.

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 microarchitecture model and thus being able to perform microarchitectural DSE.

3. 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 metadata about an application event.

3.1. Computational Events Signatures. A computational signature describes the complexity of computational events in a (micro-)architecture-independent fashion using an abstract instruction set (AIS) [3]. 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 RISC 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 here in after) 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 for example, 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 [7] 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. Taking the signature generation for the MicroBlaze processor as an example in Figure 2, application files are loaded into mb-gdb, which is the GNU C debugger for MicroBlaze. Mb-gdb is used to send instructions of the loaded executable files to the MicroBlaze instruction-set simulator and to perform cycle-accurate simulation of the execution of the software programs, as in [8].

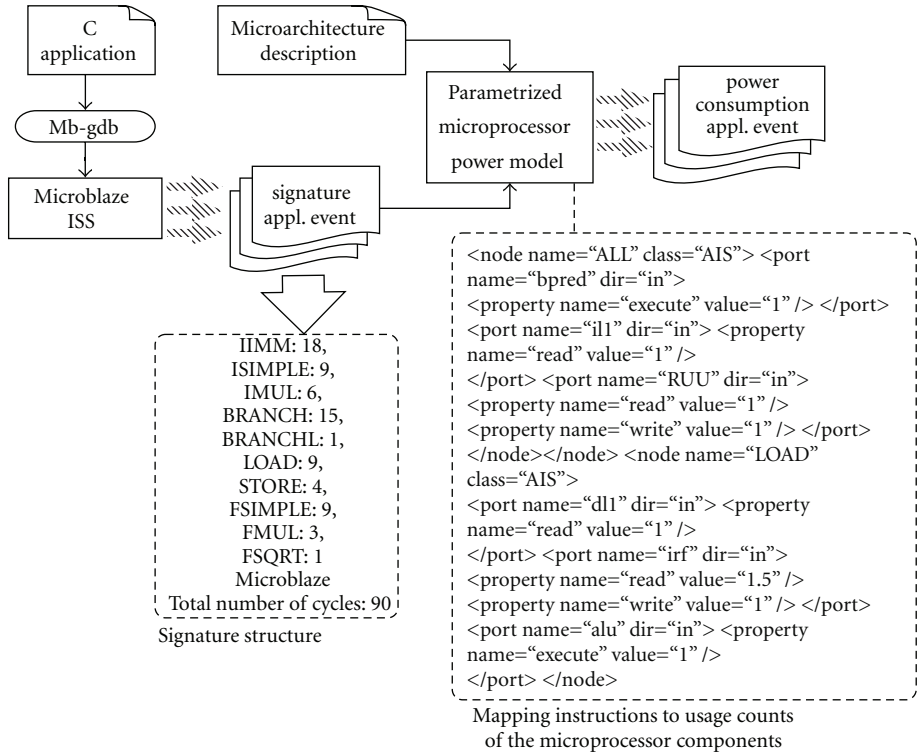

Figure 2: Computational event signature generation for MicroBlaze.

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 an 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 microarchitectural information, such as cache missrates, and branch misprediction rates. 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 microarchitecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described.

The microprocessor power model also uses a microarchitecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described. An example fragment of this mapping description is shown in Figure 2. It specifies that for every AIS instruction (indicated by the ALL tag), the instruction cache (il1) is read, the register update unit (RUU) is read and written, and branch prediction is performed. Furthermore, it specifies that for the AIS instruction LOAD, the ALU is used (to calculate the address), the level-1 data cache (dl1) is accessed, and the integer register file (irf) is read and written. With respect to the latter, it takes register and immediate addressing modes into account by assuming 1.5 read operations to the irf on average. In addition, the microarchitecture description file also contains the parameters for our power model, such as, the dimensions and organization of memory structures (caches, register file, etc.) in the microprocessor, clock frequency, and so on. Clearly, this microarchitecture description allows for easily extending the AIS and facilitates the modeling of different microarchitecture implementations.

3.2. 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 4. 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.

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 about the usage of the channel and the switching factor, which is related to the data distribution. In our transaction-level architecture models, we use the assumption that the communications performed by the KPN application model are not interleaved at the architecture level. For example, if a pixel block is transferred

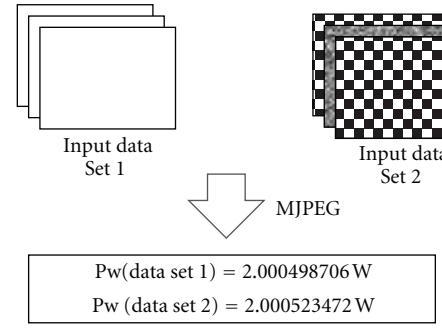

Figure 3: Measured power consumption of MJPEG application mapped on one MicroBlaze using two different input sets.

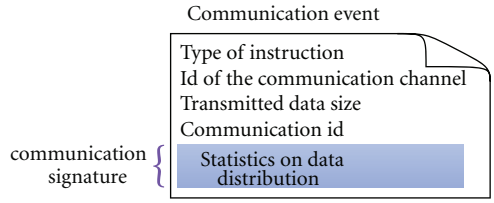

Figure 4: Structure of communication events.

between two KPN processes, then the architecture model simulates the (bus/network) transactions of the consecutive data words in the pixel block, without interleaving these transactions with other ones. In Figure 3, we show the impact of power on a MJPEG application using input sets with different data distribution. In the first input data set picture, the correlation between pixel blocks is very high, and consequently the average Hamming distance of the data will be zero. This results in lower power values with respect to the second input data set picture, which presents a higher Hamming distance distribution.

3.3. Signature-Based, System-Level Power Estimation. In Figure 5, the entire signature-based power modeling framework is illustrated. First the event traces are generated, together with the communication signatures.

The Kahn application model is used to generate the event traces, which represent the workload that is imposed on the underlying MPSoC architecture model. During this stage, 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. The communication signature generation is mapping dependent: communication patterns change with different mappings.

In addition, the computational signatures are generated (Figure 5, left side). In particular, 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. After that the computational event signatures are generated, the power consequences of trace events generated by the application model, are computed. As will be explained in the next section, the microprocessor power model uses a microarchitecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described.

Table 1: Different possibilities of reusing signatures in DSE.

Comp. signatures	Comm. signatures
μ-architectural exploration	μ-architectural exploration
Mapping exploration (limited)	Architectural exploration

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. 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).

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 1). For example, once computational signatures are generated for application events, it is possible to explore different microarchitectures 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 homogeneous systems. Communication signatures can be reused for both microarchitectural and architectural exploration.

4. 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, for example, traditional ISS-based power models, while still incorporating an explicit microarchitecture model and thus being able to perform microarchitectural DSE. The power models are based on FPGA technology, since we have incorporated these models in our system-level MPSoC synthesis framework Daedalus [5], 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

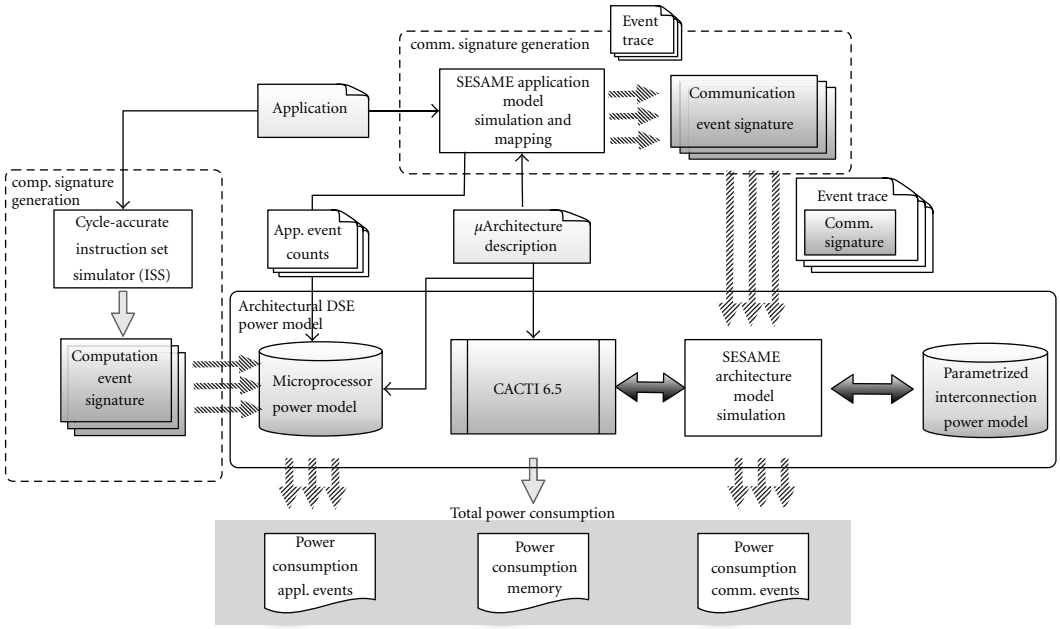

Figure 5: System-level power estimation framework.

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.

4.1. Interconnection Power Model. In this section, we derive architectural level parameterized, and 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 [5], 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:

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

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 arbiter (i.e., the number of *read* and *write* events that involve the arbiter) 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:

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

where C_{LUT}, n_{LUTs}, f, and 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 precalibration of the model, and

$$\text{access\_rate} = \frac{T_{\text{reads}} + T_{\text{writes}}}{T_{\text{total\_exec}}}. \tag{3}$$

Here, T_{reads} and T_{writes} are the total times spend on the execution of read and write transactions, respectively, and $T_{\text{total\_exec}}$ 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:

$$P_{\text{chan}} = \beta \times V_{\text{dd}}^2 \times f \times C_{\text{chan}} \times n_{\text{LUTs}} \times \text{Hamm\_dist}(e), \tag{4}$$

where β, C_{chan}, f, V_{dd}, and 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.

4.2. 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 [9] 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:

$$\text{signal\_rate} = \gamma \times \text{Hamm\_dist}(e), \tag{5}$$

where the γ is again a scaling factor obtained through precalibration of the model.

4.3. Microprocessor Power Model. The microprocessor model that underlies our power model is based on [3]. 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 subcomponents 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) [3].

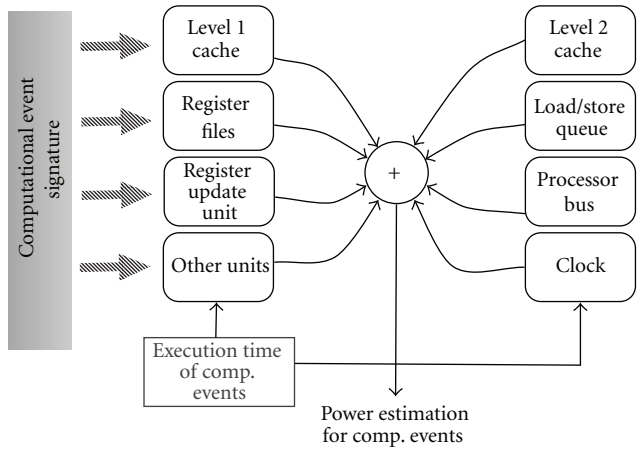

Figure 6: Different components in the microprocessor power model.

The power consumption of a computational application event is calculated by accumulating the power consumption of each of the components that constitute the microprocessor power model, as shown in Figure 6. 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 for example, the ALU (see other units in Figure 6), 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, 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.

5. 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 [4, 5]. This allows direct validation and calibration of our power model.

5.1. The Daedalus Design Flow. Daedalus offers a fully integrated tool flow in which design space exploration (DSE), system-level synthesis, application mapping, and system prototyping of MPSoCs are highly automated, which allows

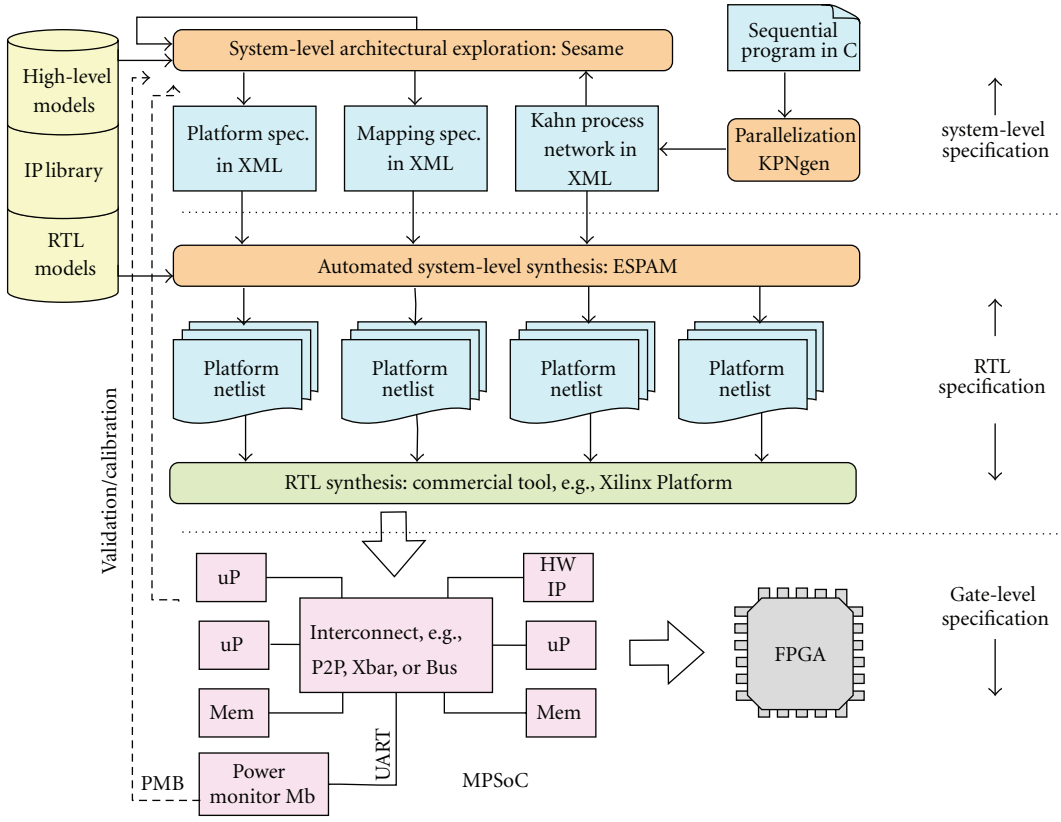

Figure 7: The Daedalus design and validation tool flow.

a direct validation and calibration of our power model. In Figure 1, the conceptual design flow of the Daedalus framework is depicted.

A key assumption in Daedalus is that the MPSoCs are constructed from a library of predefined and preverified IP components. These components include a variety of programmable and dedicated processors, memories, and interconnects, thereby allowing the implementation of a wide range of MPSoC platforms. So, this means that Daedalus aims at composable MPSoC design, in which MPSoCs are strictly composed of IP library components. Daedalus consists of three core tools.

Starting from a sequential multimedia application specification in C, the KPNgen tool [6] allows for automatically converting the sequential application into a parallel Kahn process network (KPN) specification. Here, the sequential input specifications are restricted to so-called static affine-nested loop programs, which is an important class of programs in, for example, the scientific and multimedia application domains. The generated or handcrafted KPNs (the latter in the case that, e.g., the input specification did not entirely meet the requirements of the KPNgen tool) are subsequently used by the Sesame modeling and simulation environment [2, 10] to perform system-level architectural design space exploration. To this end, Sesame uses (high-level) architecture model components from the IP component library (see the left part of Figure 7). As discussed before, Sesame allows for quickly evaluating the performance of different application to architecture mappings, HW/SW partitionings, and target platform architectures. Such exploration should result in a number of promising candidate system designs, of which their specifications (system-level platform description, application-architecture mapping description, and application description) acting as an input to the ESPAM tool [4, 5]. This tool uses these system-level input specifications, together with RTL versions of the components from the IP library, to automatically generate synthesizable VHDL that implements the candidate MPSoC platform architecture. In addition, it also generates the C code for those application processes that are mapped onto programmable cores. Using commercial synthesis tools and com pilers, this implementation can be readily mapped onto an FPGA for prototyping. Such prototyping also allows calibrating and validating Sesames system-level models, and as a consequence, improving the trustworthiness of these models.

5.2. Experimental Results. 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 and also a point-to-point network. The softcores on the FPGA device used in the framework do not use caches at this moment. This is considered to be future work. The validation environment

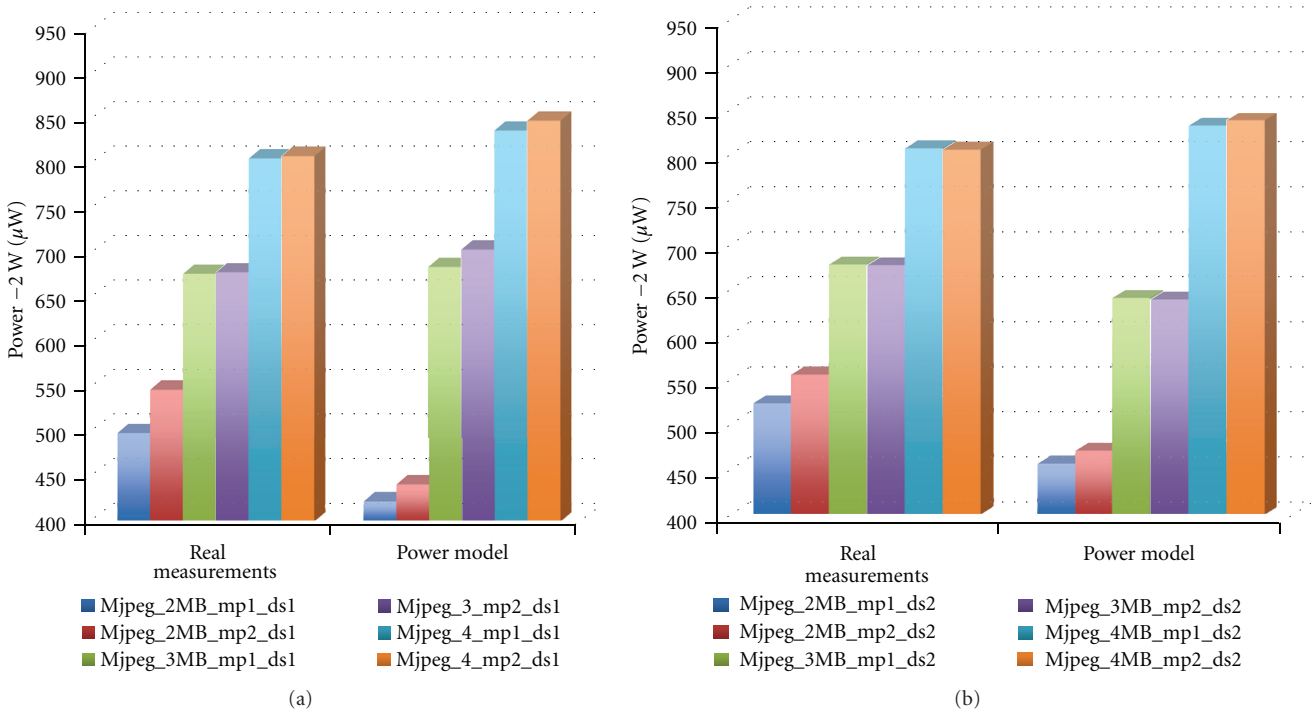

FIGURE 8: Mjpeg application with input set ds1 (a) and input set ds2 (b).

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 [11]. In order to do this, it runs a software driver which implements the PMBus protocol [11]. The extra MicroBlaze "polls" the power values in the Virtex-6 FPGA internal measurement registers and prints out the read values through the UART to the pc, as shown in Figure 7. In this way, we have a fully automated system to register the power values of an architecture running a particular application with a given mapping. 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 [12]. 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, for example, using a simulator [13].

The results of the validation experiments are shown in Figures 8, 9, 10, and 11. In the experiments, we compare the total power consumption, which is both leakage and dynamic power. 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 8, 9, 10, and 11 apply the following notation: $\text{app}_{\text{name}}\_\text{n}_{\text{proc}}\_\text{mapping}_{\text{type}}$, 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 $\text{mapping}_{\text{type}}$ 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. With respect to our previous work [14], we extend the analysis to a point-to-point architecture based on FIFOs. That is we tested the power model on two different communication architecture configurations: the first one is crossbar-based, while the second one is a point-to-point network based on FIFOs. The power values in Figures 8, 9, 10, and 11 are scaled by a factor of 2 W for the sake of improved visibility. Most charts show a very little difference between the good and bad configurations (mp1 versus mp2) for a number of processors greater than 2; this is explained by the fact that a design with a larger number of processors implies a higher use of the communication channels. Given an application with m tasks and n processors, if $m \gg n$,

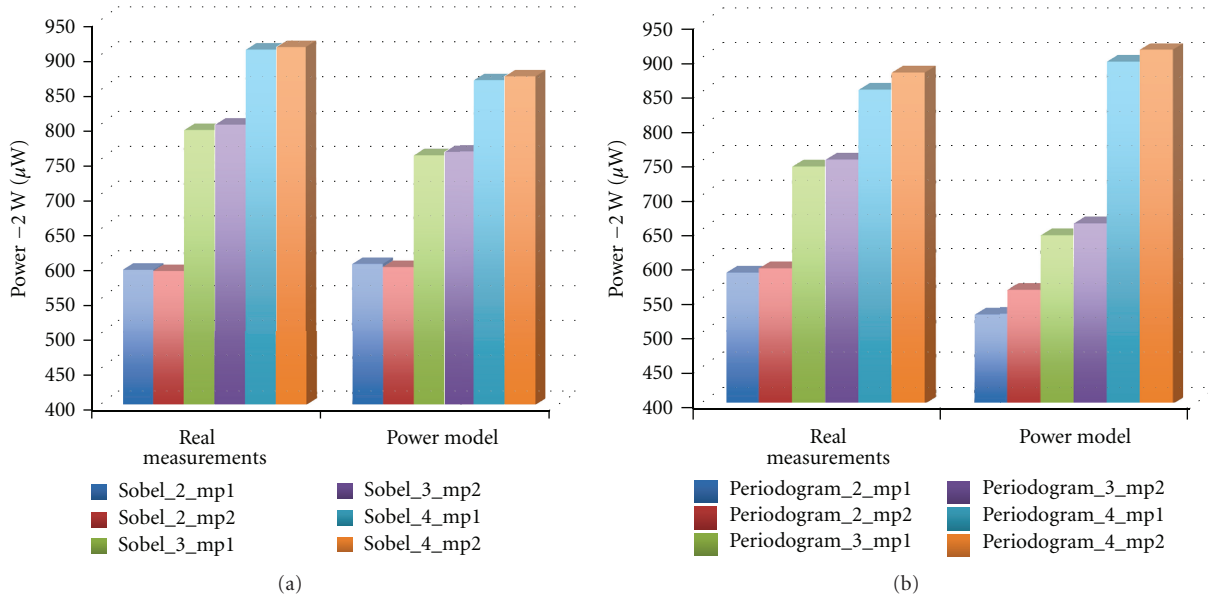

Figure 9: Sobel filter (a) and Periodogram application (b).

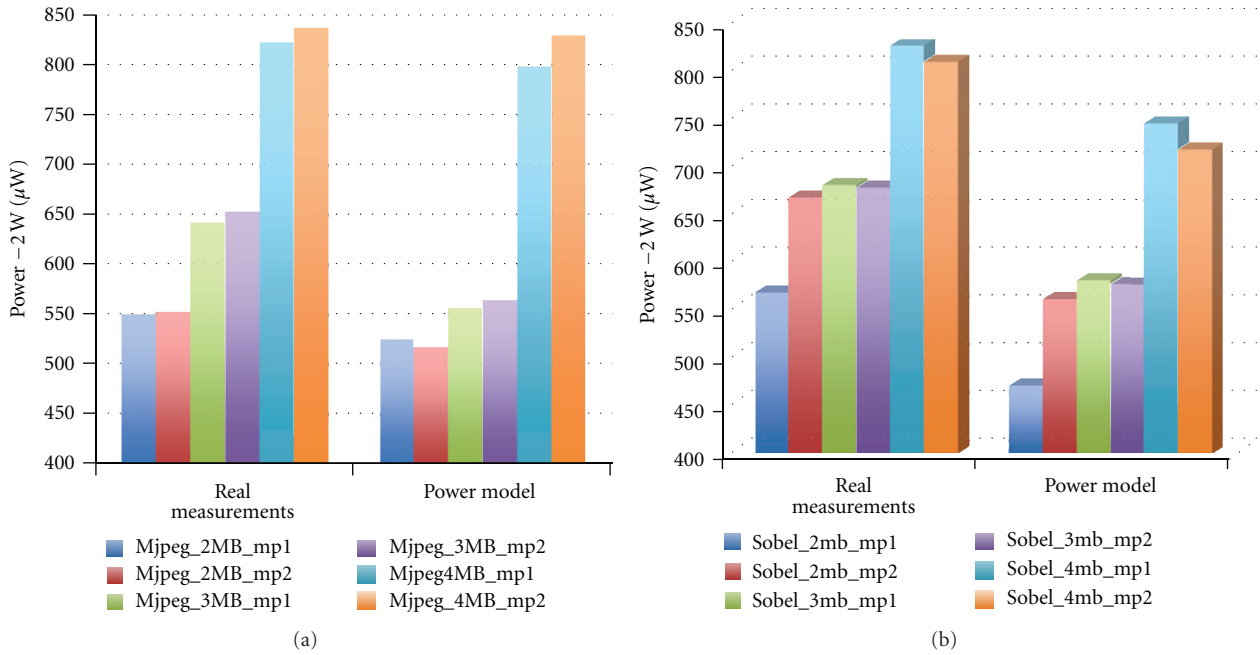

Figure 10: Mjpeg (a) and Sobel (b) applications in a point-to-point FIFO architecture.

then this implies that a good mapping can be beneficial for reducing tasks communication. However, in the case of $m = n$, the tasks mapping cannot avoid substantial communications.

The results in Figures 8, 9, 10, and 11 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% for the crossbar networks and an average error of our power estimations of around 10%, with a standard deviation of 6% for the point-to-point networks. 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

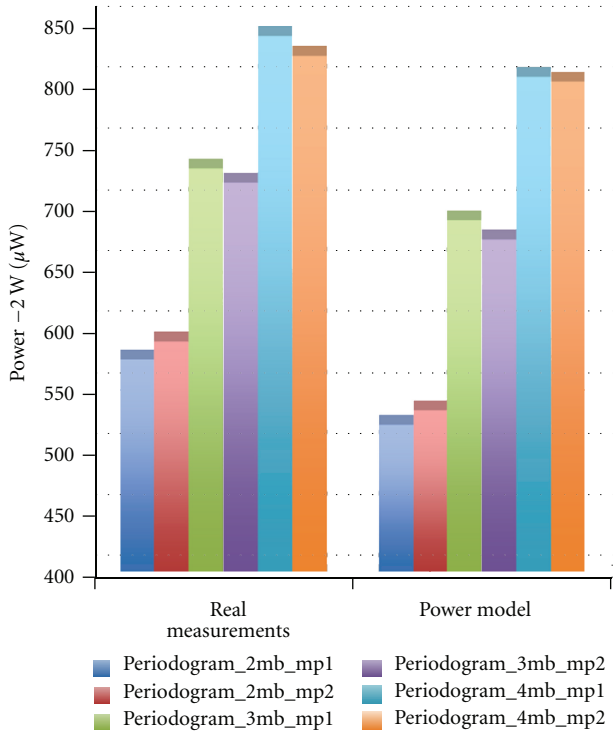

Figure 11: Periodogram application in a point-to-point FIFO architecture.

for a correct candidate architecture generation and selection during the process of design space exploration.

Since every design point evaluation takes only 0.16 seconds on average, 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.

6. Related Work

There exists a fairly large body of related work on system-level power modeling of MPSoCs. For example, in [15] developed a SoC power estimation method based on a SystemC TLM modeling strategy. It adopts multiaccuracy models, supporting the switch between different models at run time according to the desired accuracy level. The authors validate their model using the STBus NoC and an analytical power model of this NoC. An MPEG4 application was tested, achieving up to 82% speed-up compared to TLM BCA (bus cycle-accurate) simulation.

Atitallah et al. [16] uses a stack of abstract models. The higher-abstraction model, named Timed Programmer View (PVT), omits details related to the computation and communication resources. Such an abstract model enables designers to select a set of solutions to be explored at lower abstraction levels. The second model, CABA (cycle-accurate bit-accurate), is used for power estimation and platform configuration.

In [17], a system-level cycle-based framework to model and design heterogeneous MPSoC (called GRAPES) is presented. C++/SystemC-based IP system modules can be wrapped to act as plugins, which are managed by the simulation kernel in a TLM fashion. Those modules are managed by the GRAPES kernel, which is the core of the simulation framework. To estimate power during a simulation, they add a dedicated port to each component, which communicates with the corresponding power model. This feature permits to characterize each component with a set of activity monitors (inside the component module) necessary for the power estimation.

Reference [18] 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, comprising the Intel XScale-embedded microprocessor, its WMMX SIMD co processor, L1 caches, SDRAM, and the on-board address and data buses.

In [19], 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. The authors validate the framework using an ARM926EJ-S CPU and the AMBA AXI 3.0 as NoC. speed-up compared to a gate-level simulation is on average 100 times faster. Previous work was not addressing high-level power modeling of MPSoCs on FPGA.

In [20], an efficient hybrid system level (HSL) power estimation methodology for FPGA-based MPSoC is proposed. Within this methodology, the functional level power analysis (FLPA) is extended up to set up generic power models for the different parts of the system. Then, a simulation framework is developed at the transactional level to evaluate accurately the activities used in the related power models. With respect to this work, our processor model can easily model different kinds of RISC processors by simply introducing new units.

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 [21–24]. Many of the above approaches calibrate the high-level models with parameters extracted from RTL implementations, using low-level simulators for the architectural components. In [21], a rate-based power estimation method is presented. In the first phase, it considers data volume, estimating the average power in function of the total transmitted data: in the second phase, it calibrates the model through definition of the consumed power for each transition rate. In particular, the calibration uses RTL model of the NoC, while the latter uses an actor-oriented model. After the calibration, a power dissipation table is generated for each injection rate and router element. Using linear approximation, they determine the power dissipation for each injection rate. In [22], an energy estimation model based on the traffic flow in the NoCs building blocks (routers and interconnection wires) is presented. The authors represents the amount of energy consumed in the transmission of a data bit throughout the NoC (in its routers and interconnection wires). In [23], a NoC power and performance analysis with different traffic models, using analytical models, is presented. The authors

target a NoC with a mesh topology. The employed traffic models are: uniform, local, hot-spot, and matrix transpose. Results were compared to Synopsys Power Compiler and ModelSim, showing an error of 2% for power estimation and 3% for throughput. In [24], a methodology for accurate analysis of power consumption of message-passing primitives in a MPSoC is proposed, and, in particular, an energy model which allows to model the traffic-dependent nature of energy consumption through the use of a single, abstract parameter, namely, the size of the message exchanged. The ISS performs cycle-accurate simulation of the cores, while the rest of the system is described in SystemC at signal level. In [25], the authors employ a framework that takes as input message flows and derives a power profile of the network fabric. The authors map the CPU data path as a graph and the application as a set of messages that flow in this graph. Those mapped CPUs are connected into the network fabric, mapping the entire MPSoC as a network. The authors make use of a network power estimation tool, called LUNA, to evaluate the power dissipation of the entire MPSoC.

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., [15, 21–23, 25]) or validation by means of measurements on fixed target platforms (e.g., [18, 19]). Consequently, in general, related system-level MPSoC modeling efforts do also not target FPGA technology in their system-level power models.

7. 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, for example, 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 good power estimates in terms of DSE. As future work, we plan to perform additional experiments (e.g., using different memory hierarchies and different processor components) as well as to deploy the power model in real system-level DSE experiments.

Acknowledgments

This work has been partially supported by the MADNESS STREP-FP7 European Project and the NWO EASY 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] A. B. Kahng, L. Bin, L. S. Peh, and K. Samadi, "ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '09)*, pp. 423–428, April 2009.

[2] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, vol. 55, no. 2, pp. 99–111, 2006.

[3] P. Stralen and A. D. Pimentel, "A high-level microprocessor power modeling technique based on event signatures," in *Proceedings of the IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '07)*, 2007.

[4] M. Thompson, H. Nikolov, T. Stefanov et al., "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs," in *Proceedings of the 5th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '07)*, pp. 9–14, October 2007.

[5] H. Nikolov, M. Thompson, T. Stefanov et al., "Daedalus: Toward composable multimedia MP-SoC design," in *Proceedings of the 45th Design Automation Conference (DAC '08)*, pp. 574–579, June 2008.

[6] G. Kahn, "The semantics of a simple language for parallel programming," in *Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed., vol. 74, pp. 471–475, North-Holland Puplishing Company, New York, NY, USA, 1974.

[7] T. Austin, E. Larson, and D. Ernest, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 12–67, 2002.

[8] J. Ou and V. K. Prasanna, "Rapid energy estimation for hardware-software codesign using FPGAs," *EURASIP Journal on Embedded Systems*, vol. 2006, Article ID 98045, 11 pages, 2006.

[9] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture (MICRO '07)*, pp. 3–14, December 2007.

[10] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 82123, 11 pages, 2007.

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

[12] P. K. Huang, M. Hashemi, and S. Ghiasi, "System-level performance estimation for application-specific MPSoC interconnect synthesis," in *Proceedings of the Symposium on Application Specific Processors (SASP '08)*, pp. 95–100, June 2008.

[13] J. Becker, M. Huebner, and M. Ullmann, "Power estimation and power measurement of xilinx virtex fpgas: Trade-offs and limitations," in *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI '07)*, p. 283, IEEE Computer Society, Washington, DC, USA, 2003.

[14] R. Piscitelli and A. D. Pimentel, "A high-level power model for mpsoc on fpga," in *Proceedings of the 18th Reconfigurable Architectures Workshop (RAW '11)*, 2011.

[15] D. Sciuto, G. Beltrame, and C. Silvano, "Multi-accuracy power and performance transaction-level modeling," in *Proceedings*

of the Conference on Design, Automation and Test in Europe (DATE '08), 2008.

[16] J. L. Dekeyser, R. B. Atitallah, and S. Niar, "MPSoC power estimation framework at transaction level modeling," in *Proceedings of the 19th International Conference on Microelectronics (ICM '07)*, pp. 245–248, December 2007.

[17] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, "A modular approach to model heterogeneous MPSoC at cycle level," in *Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD '08)*, pp. 158–164, September 2008.

[18] A. Varma, E. Debes, I. Kozintsev, P. Klein, and B. Jacob, "Accurate and fast system-level power modeling: An XScale-based case study," *Transactions on Embedded Computing Systems*, vol. 7, no. 3, article 25, 2008.

[19] 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)*, pp. 551–558, IEEE Press, January 2006.

[20] S. Niar, E. Senn, S. K. Rethinagiri, R. B. Atitallah, and J. L. Dekeyser, "Hybrid system level power consumption estimation for fpga-based mpsoc," in *Proceedings of the 29th IEEE International Conference on Computer Design (ICCD '11)*, October 2011.

[21] L. Ost, G. Guindani, L. Indrusiak, S. Maatta, and F. Moraes, "Using abstract power estimation models for design space exploration in NoCbased MPSoC," *IEEE Design and Test of Computers*, vol. 28, no. 2, pp. 16–29, 2011.

[22] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based noc architectures under performance constraints," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '03)*, pp. 233–239, New York, NY, USA, 2003.

[23] 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 *Proceedings of the 21st International Conference on VLSI Design (VLSID '08)*, 2008.

[24] M. Loghi, L. Benini, and M. Poncino, "Power macromodeling of MPSoC message passing primitives," *ACM Transactions in Embedded Computing Systems*, vol. 6, no. 4, article 31, 2007.

[25] N. Eisley, V. Soteriou, and L. Peh, "High-level power analysis for multi-core chips," in *Proceedings of the International conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*, pp. 389–400, New York, NY, USA, 2006.