ESL POWER AND PERFORMANCE ESTIMATION FOR HETEROGENEOUS MPSOCS USING SYSTEMC

Martin Streubühr[1], Rafael Rosales[1], Ralph Hasholzner[2], Christian Haubelt[3], and Jürgen Teich[1]

[1] University of Erlangen-Nuremberg, Erlangen, Germany
[2] Intel Mobile Communications GmbH, Neubiberg, Germany
[3] University of Rostock, Rostock, Germany
{martin.streubuehr, rafael.rosales, teich}@cs.fau.de
ralph.hasholzner@intel.com, christian.haubelt@uni-rostock.de

ABSTRACT

In this paper, we propose a top-down power and performance estimation methodology for heterogeneous multiprocessor systems-on-chip. The proposed approach is applicable at the Electronic System Level (ESL). Thus, power and performance estimation can be applied in very early design phases. By strictly separating the system functionality from its architecture, different design options can be assessed with minimal effort. Moreover, the simulation-based approach permits to evaluate the effects of dynamic power management, even at this level of abstraction. A case study from the multimedia domain is used to show the efficiency of the proposed methodology.

Index Terms— System-Level Modeling, Power and Performance Evaluation, Simulation, SystemC

1. INTRODUCTION

Multiprocessor systems-on-chips (MPSoCs) are becoming more and more the important implementation platform for computationally intensive applications, e.g., multimedia or network processing. As heterogeneous MPSoCs are mainly designed for a given set of applications, their advantage lies in low power dissipation due to a lower clock frequency but still high computational power by exploiting the parallelism provided by the architecture. However, power and performance estimation for such architectures is a challenging task. On the one hand, building prototypes for measurements is costly and time-consuming and analytical or statistical methods are often too pessimistic leading to over-constrained designs.

Power and performance numbers are crucial design parameters, especially in the design of mobile platforms. Thus, novel estimation techniques are required at high levels of abstraction, hence being available in early design phases. In recent years, virtual prototyping has been proposed in order to estimate software code performance with good accuracy and reasonable runtime. But for using virtual prototyping tools most of the software implementation already has to be done, limiting its usefulness in early design phases.

Many high level power estimation approaches are promoted in literature. For digital design, mostly bottom-up approaches are applied by incrementing the abstraction level from well known techniques. Typically these approaches require much implementation details like RTL models, ISS simulation models, or transaction level models. More abstract approaches typically are based on analytical models. E.g. modeling power state machines for hardware components allows for deriving minimum and maximum bounds for power dissipation by analyzing the state space. Other prominent analytical approaches are based on spreadsheet calculations often used in the design phase. Here, activity of components and applications is abstracted to characterizing numbers in order to estimate power consumption by formula. Such approaches abstract many dynamic effects to average numbers and are applicable when limiting the analysis to few use cases.

Our work applies a top-down approach by combining model-based design and spreadsheet power models: Similar configuration data as spreadsheet models is added to a model-based power and performance simulation. Thereby, dynamic effects like power state switching (frequency and voltage scaling) can be simulated. Furthermore, dynamic behavior of applications can be modeled by functional SystemC simulation models by applying model-based design. These models can be very abstract covering simple activity profiles or may include more detailed functional code. For the later case, dynamic behavior like data-dependent control flow can be modeled without adding to much implementation details.

We do not expect very accurate performance estimations by applying the presented approach. But our approach targets the design phase and is expected to provide relative figures for comparing different design options. For more precise estimations, very accurate techniques exist but require a lot of implementation details. In summary our approach has the following advantages:

- Estimations can be done in very early design phases, even if no functional source code of the application is available.
- Different design options can be assessed with minimal effort, i.e., changing a single configuration file only.
- Effects from dynamic power management are taken into account.
- Configurable simulation models allow for fast power and performance analysis.
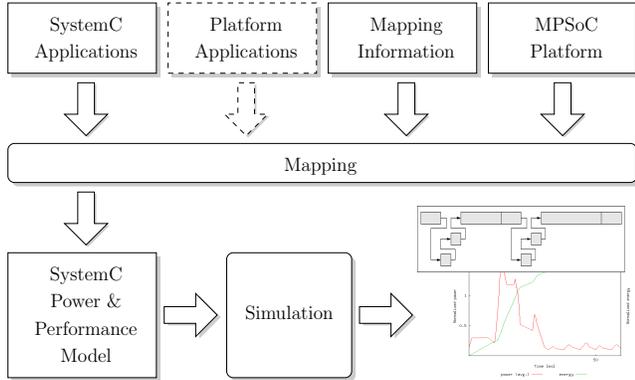
Fig. 1. Methodology overview: Given an MPSoC platform model, a SystemC model of the applications, and a mapping constraint file, a System-level Power and Performance Model is generated automatically. Power and performance estimation is done by simulation.

The proposed methodology is open to incremental refinement, giving the opportunity for integration into existing design flows.

Figure 1 shows an overview of the proposed methodology. Following the Y-chart approach [1], a special focus is put on the separation of functionality and architecture. The functionality is given by a set of applications modeled as actor-oriented design in SystemC [2] by the designer. Additional platform applications may be modeled also, e.g., modeling the software provided by the platform vendor. Especially, firmware threads like periodic wireless network monitoring daemons contribute to the MPSoC's resource utilization and, thus, influence the overall performance. In contrast to the functional model, the architecture model is composed by *Virtual Processing Components* written in SystemC [3]. These components represent the processing and interconnection resources of the heterogeneous MPSoC platform and provides a technical basis for implementing the functionality. Finally, a mapping constraint file assigns actors of the applications to the components of the MPSoC platform. Given this information, a System-level Power and Performance Model (SPPM) is generated automatically. As the SPPM is implemented using SystemC, power and performance numbers can be estimated by analyzing simulation traces. By changing the mapping constraint file, different design options can easily be explored.

The remainder of this paper is structured as follows: Related work is discussed in Section 2. In Section 3 the input for automatically generating the SPPM is discussed. In Section 4 the simulation-based power and performance estimation is described. Section 5 presents results from applying the proposed methodology to a multimedia application mapped onto state-of-the-art cell phone platforms. Conclusions are given in Section 6.

2. RELATED WORK

Approaches for power and performance modeling of embedded systems exist at several levels of abstraction, namely layout level, cycle-accurate register transfer level [4, 5], in-

struction level [6, 7, 8], cycle-accurate transaction level [9], approximately-timed transaction level [10, 11][1], application level [12], etc. Some of them also consider effects of dynamic power management (Dynamic Voltage and Frequency Scaling), as reported in [13]. Often a bottom-up abstraction is done, where abstract power and performance models are derived from existing detailed models or real implementations. Due to incremental abstractions, a hierarchy of power models [14] is defined, trading accuracy and efficiency in power estimation.

At the Electronic System Level, the following modeling approaches have been explored: Spreadsheet models, e.g., in [15], where power dissipation is obtained from data sheets and cumulated for all components. Power state machine models [16], where power dissipation of components are modeled for steady states and state transitions. In this approach, functionality of components is abstracted as much as possible. Power estimation using power state machines can be simulation-based [16] or analytical [17] by building the product power state machine of all components in a system. Such an analytical model can provide a maximum and minimum bound of power dissipation. Simulation-based approaches typically estimate power dissipation for use cases or workload scenarios. Especially SystemC-based simulation approaches are reported in [18, 19]. API power hooks are used in [18] for the integration of power models to SystemC performance models. An aspect-oriented methodology that avoids API calls is used in [19] to integrate power models in functional SystemC models.

Activity driven or rate-based power models, where the power model is driven by activity data from a performance model can be found in [20, 21]. The former makes use of a system simulator and combines the simulation perfomance counters and operating conditions, such as voltage and frequencies, for power estimation. The later proposes the use of "event signatures", that encapsulate machine instructions plus some microarchitecture data derived from a instruction set simulation and abstracted away to be fed into the power model for better estimations.

Beside power modeling, many new approaches in performance modeling have been proposed, e.g., [22, 23, 3]. They show many similarities to our proposed performance estimation methodology, but do not consider power estimation.

Our approach focuses on modeling of power dissipation at the Electronic System Level as well, but in contrast, our approach allows for power and performance evaluation of architecture variants by separating functionality and architecture. The application model is mapped to platform models, which include power models. Via configuration files, different variants can be evaluated by only changing the mappings between functionality and architecture, while no further changes in the functional model are required. Sophisticated scheduling and arbitration policies allows for modeling of software tasks running on processors and communication across shared media. The abstraction level for our power and performance simulation corresponds to approximately-timed transaction level, as in [10, 11]. Being a top-down approach, it differentiates from other activity driven approaches where abstractions are introduced in a bottom-up fashion. Resource contention can

---

[1]The presented approaches refer to the *programmers view with timing* term which corresponds to the approximately-timed coding style in the actual TLM 2.0 standard.

be handled for communication and computation. Furthermore, we adopted the notation of power state machines [16] by using a generic power state machine for each component of the MPSoC platform. The power dissipation for components in the architecture model can be configured from data sheets as in [15] or even from more detailed models or existing platforms. Finally, we support modeling of dynamic power switching as reported in [13]. For rapid design space exploration, different design options can easily be evaluated by configuring the mapping of the application model to the architecture model as reported in [3]. As a consequence, our system-level approaches can be used for a power-aware top-down design of embedded systems.

3. MODELING

In this section, we discuss the required input to generate the System-level Power and Performance Model (SPPM) in order to perform a power and performance estimation. As we use SystemC as the back-bone simulator, this input is required to elaborate the simulation model, consisting of a platform model, an application model, mapping constraints, and transaction routings.

3.1. Characterizing the MPSoC platform

The MPSoC platform is modeled by a set of components representing microcontrollers, DSPs, hardware accelerators, memories, and buses. Each component is characterized by a scheduler required to resolve resource contention and three different *execution modes*, namely IDLE, STALL, and EXEC. With each execution mode a power dissipation is associated. IDLE represents the mode where the component has no assigned actor to be executed, i.e. the running list of a potential scheduler is empty. In the EXEC mode the component is assumed to execute an actor ready for execution, whereas in the STALL mode a ready actor is assigned to the component but required input data still has to be transported to this particular component or a result has to be written back to memory, i.e., a memory access has been initiated. Note that for components which cannot initiate memory transactions, the STALL mode can be omitted.

In order to model the influence of dynamic power management techniques, several so called *power states*, e.g., specified by different voltages and frequencies, can be associated with each component. Without loss of generality, we assume in the following that two different power states (FAST, SLOW) are associated with each component. Each power state is characterized with the three above mentioned execution modes. Moreover, a *transfer delay* is specified for each power state of each component. This transfer delay is the time needed to transport a single byte of data from one connected component to another.

Finally, if the component has specified more than one power state, a so called *governor* is needed to select the proper power state at runtime. In this paper, we just consider a very simple governor algorithm, selecting the power state according to a component's load. The load of a component is calculated over a user specified time window. The power state selection is controlled by a hysteresis with a lower and an upper threshold.
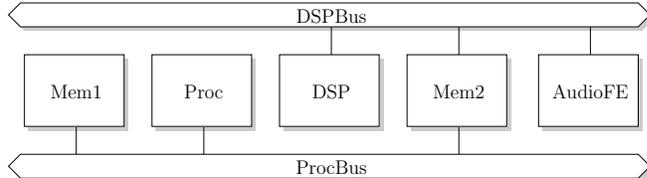

Fig. 2. An MPSoC platform that consists of a processor, a DSP, a hardware accelerator, and two memories and buses.

An example of an MPSoC platform is given in Figure 2. The platform characterization is done using an XML file. A part of this XML file is shown below:

```
<platform>
  <component name="Proc">
    <scheduler type="FCFS"/>
    <power value="FAST">
      <idle   value="0.04 mW"/>
      <stall value="0.27 mW"/>
      <exec   value="1.00 mW"/>
      <transfer value="0.83 ns"/>
    </power>
    <power value="SLOW">
      <idle   value="0.01 mW"/>
      <stall value="0.05 mW"/>
      <exec   value="0.17 mW"/>
      <transfer value="3.33 ns"/>
    </power>
    <governor>
      <window value="10 ms"/>
      <high    value="70%"/>
      <low     value="20%"/>
    </governor>
  </component>
  <component name="ProcBus">
    ...
</platform>
```

Within the `platform` tag, the processing components and the interconnection components of the MPSoC platform are defined. In the above listing only a single processor `Proc` is specified with a First Come First Served (FCFS) scheduling policy and two distinguished power states `FAST` and `SLOW`. For each component and each power state the three execution modes are configured with power dissipation values in mW. Furthermore, the transfer delay is specified per component and power state. Other components of the platform including hardware accelerator, buses, and memories are characterized in a similar way. The power dissipation numbers can be estimated by power estimation tools at lower levels [24, 12] or by measurements carried out on existing platforms. The measurements usually need to be done under multiple test scenarios designed for this purpose. The test scenarios should be designed such that the power consumption of a major component can be derived or extrapolated from the measurements performed at different frequencies and/or voltages.

3.2. Application Modeling

To allow power and performance estimation, the user has to specify the set of applications. In the proposed methodology, each application must be written in actor-oriented style using SystemC. The concepts of actor-oriented design [25, 26] guarantee that the functionality is defined in a way such that it
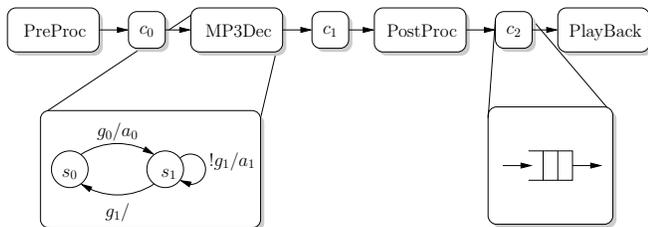
Fig. 3. The actor-oriented design models an MP3 decoding application. Actors are specified by finite state machines and actions. Communication between actors is done via queues with FIFO semantics.

is independent from the actual mapping to an MPSoC platform. Actors form atomic units with clearly defined execution and communication rules and each actor needs to be mapped to exactly one component in the MPSoC platform. Thus, actors define the granularity for potential hardware/software partitionings.

For the proposed methodology, a SystemC library [2] for actor oriented design is used. The library provides special channels for inter-actor communication and a C++ syntax to specify the behavior of actors in a unique way. Channels are queues with FIFO semantics allowing to read/write and consume/produce data in a first in first out discipline. Actors are restricted to communicate through dedicated channels only.

An example of an actor-oriented model of an MP3 decoding is given in Figure 3: The application is modeled by a chain of actors. The first actor (PreProc) does the preprocessing, i.e., parsing the MP3 input stream and extracting important information, e.g., compression rate. Audio samples provided by the preprocessing are decoded by the MP3 decoding actor (MP3Dec). The decoded audio samples are post processed by the PostProc actor performing for example equalizing and front end adaption. Finally, the PlayBack actor performs the actual playback of the audio samples.

The control behavior of an actor is specified using a finite state machine. Each state represents a start or end point for communication of the actor. State transitions are annotated by a so called *guard* and a so called *action* (separated by a /). This shows some similarities to the rule-based model of computation for SystemC proposed in [27] which is based on the idea of *guarded atomic actions* [28]. If a guard evaluates to true, the associated action is performed and the end state of the corresponding state transition is entered.

For example the finite state maschine of the MP3Dec actor is shown in Figure 3. The actor starts in state s_0. Guard g_0 checks if new input data is available. If g_0 is satisfied, action a_0 reads a frame of an encoded sample and the actor changes to state s_1. In this state, guard g_1 tests if the entire sample is decoded. If not, a chunk of data will be decoded by a_1. Otherwise, the actor will return to its initial state s_0. Note that the number of loops in state s_1 depends on the content and compression rate of the MP3 sample.

If the power and performance estimation should take different applications into account, these applications can be specified in a similar way. Next, it will be discussed how to map a given set of applications to an MPSoC platform.

3.3. Setting the Mapping Constraints

Mapping constraints are specified again within an XML configuration file. An example fraction of the XML file is shown below:

```
<mappings>
 <mapping actor="MP3Dec" component="Proc">
  <delay mode="FAST" name="g0" value="100 us"/>
  <delay mode="SLOW" name="g0" value="300 us"/>
  <delay mode="FAST" name="g1" value="100 us"/>
  <delay mode="SLOW" name="g1" value="300 us"/>
  ...
 </mapping>
 ...
</mappings>
<topology>
  <route src="c0" snk="MP3Dec">
    <hop name="Mem1"/>
    <hop name="ProcBus"/>
    <hop name="Proc"/>
  </route>
  ...
</topology>
```

Within this file, two pieces of information have to be provided, the actor and channel mapping, as well as the transaction routing. For each actor and channel the mapping specifies the component the actor or channel is bound to. Beside this, estimated computation times for executing an actor's guard or action on the specified component are annotated to the mapping information.

It can be seen that actor MP3Dec is bound onto component Proc. In this configuration, computing guard g_0 and g_1 requires each $100\mu s$ when assuming component Proc being in FAST power state. In SLOW power state, the computation times increase by a factor of three. Beside the guard computation times, the action computation times in each power state are specified too.

The transaction routing, on the other hand, defines the routes for each actor/channel and channel/actor connection. Typically, the actor is bound to a computation resource and the channel is mapped to a memory component, i.e., the route defines the components required for handling a memory access. For each token communicated between actors corresponding write and read operations to the connecting channel would occur. In the example fraction of the XML code, the input data from channel c_0 will be read by actor MP3Dec and is routed via the hops, Mem1, ProcBus, and Proc.

4. POWER AND PERFORMANCE ESTIMATION

In this section, the automatic generation of an SPPM and the actual estimation process is presented. Results of applying this new methodology to a case study where a multimedia application is mapped onto a state-of-the-art cell phone platform are given in Section 5.

The set of actor-oriented applications is given by a functional SystemC model (cf. Figure 4(a)). This actor-oriented application model is compiled and linked with a library providing so called *Virtual Processing Components* (VPC). Changing the application models typically requires a recompilation of the function model. The XML configuration file is read at SystemC's elaboration phase and is used to configure VPCs according to the platform characterization, mappings, and transaction routings (cf. Figure 4(b)). This step requires
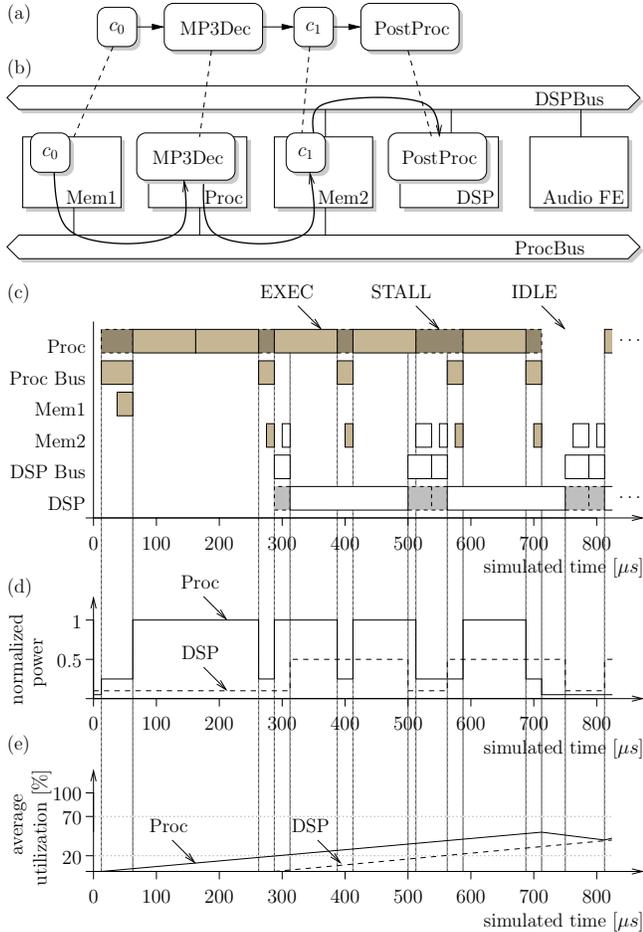
Fig. 4. (a) Application models are mapped to a platform model (b). (c) Simulation traces are obtained during simulation of the application. (d) Power traces can be computed for each component and the entire platform. (e) Power governors use component utilization to select appropriate power states.

no compilation at all, but changing the configuration file and starting the SystemC simulation again.

4.1. Performance Estimation

A VPC is a SystemC module with an event-based scheduling interface. VPCs represent components of the platform model at the granularity of processors, buses, hardware accelerators, etc. For each component in the platform characterization, a VPC is generated and a scheduler according to the configuration file is instantiated.

During elaboration of the SystemC application model, the VPC library is configured with the mapping constraints and, hence, the estimated computation times for each action and guard according to the given mapping. This information is used during simulation to assess the performance (execution time) of the system. Each time a guard or an action is activated during simulation, the guard or action is computed, respectively. Before consuming or producing any data on the actor's ports, the actor will notify the component the actor is

bound to. The actor's execution is blocked by this notification. The component will then add the actor to the ready list of the associated scheduler. The scheduler decides when an actor should be scheduled and, thus, computes when the computation time of the guard or action expires. As the interface between VPC and scheduler is event-based, preemptive and non-preemptive scheduling policies can be implemented.

An example of the combined functional and timing simulation is shown in Figure 4. In this figure, the execution of actor MP3Dec mapped to the Proc component and the actor PostProc mapped to the DSP component are visualized in simulated execution time.[2] In this example, the MP3Dec actor has enough input data for execution. Firing the actor in the application model causes a forwarding of activity events to the Proc component. Here, the MP3Dec can be executed several times without preemption, thereby, reading input data from the memory Mem1 and writing decoded data to memory Mem2. The delay from computation and communication is feed back to the application model. This results in delayed notification of new input data tokens on channel c_1 similar to an interrupt after writing the data. As a consequence, the actor PostProc is fired around $300\mu s$ causing the execution of its functionality and communication and execution delay simulation on component DSP. In the period $[510\mu s, 590\mu s]$ congestion occurs when MP3Dec and PostProc access the shared communication buffer on Mem2. During memory access, the accessing component enters the STALL execution mode. The entirely filled queue c_1 blocks the MP3Dec actor and causes the component Proc to be in the IDLE execution mode during the period $[710\mu s, 810\mu s]$ until PostProc consumes data.

Communication is handled similarly to computations (guards and actions) but requires tight synchronization of the VPCs. In a first step the size of communicated data is calculated for each channel connected to the corresponding actor. Afterwards, the delay on each component on the transaction route is computed using the associated transfer delays. The simulation of the communication delay, then, is performed analogously to the computation timing simulation using VPCs.

4.2. Power Estimation

From the generated simulation traces, it is possible to derive the power consumption of each component in the platform and, hence, for the entire platform (cf. Figure 4(d)). For this purpose, the periods of being in execution mode IDLE, STALL, or EXEC are traced and power consumption is compute accorrding to the selected power state. As an example, the Proc component in Figure 4 is in EXEC mode in the period $[60\mu s, 260\mu s]$, in STALL mode in the period of $[510\mu s, 590\mu s]$, and in IDLE state during the period of $[710\mu s, 810\mu s]$.

Together with the configured power consumption values, a so called *power trace* can be generated. The resulting power trace for the MP3 decoding assuming components DSP and Proc being in the FAST power state is shown in Figure 4(d).

Besides power tracing, the components can switch between the configured power states (e.g. SLOW, FAST) with

---

[2]The execution time would occur when running on the real MP-SoC platform. In contrast, the simulation time is required by the SystemC simulation running on a host workstation.

Table 1. The platform model is given by its components, abstract power states, and power input values. The power input is normalized to the power consumed by the processor in FAST power state and EXEC execution mode.

component	power state	normalized power input			transfer delay [ns]
		EXEC	IDLE	STALL	
Proc	FAST	1.00	0.04	0.27	0.83
	SLOW	0.17	0.01	0.05	3.33
DSP	FAST	0.47	0.08	0.08	3.33
	SLOW	0.10	0.02	0.02	6.66
ProcBus	FAST	0.24	0.04	0.04	1.66
	SLOW	0.08	0.01	0.01	3.33
DSPBus	FAST	0.12	0.02	0.02	3.33
	SLOW	0.04	0.01	0.01	6.66
Mem1	FAST	0.20	0.02	0.02	1.66
	SLOW	0.67	0.01	0.01	3.33
Mem2	FAST	0.12	0.03	0.03	1.66
	SLOW	0.04	0.01	0.01	3.33
AudioFE	FAST	0	0	0	0
	SLOW	0	0	0	0

Table 2. Four different variants of mapping the actors to the components are used in the case study.

actor	var1	var2	var3	var4
PreProc	Proc	Proc	Proc	DSP
MP3Dec	Proc	Proc	DSP	DSP
PostProc	Proc	DSP	DSP	DSP
PlayBack	AudioFE	AudioFE	AudioFE	AudioFE

Table 3. Different execution times are modeled, if the actors are mapped to the processor or to the DSP.

actor	actor execution time [us]			
	Proc		DSP	
	FAST	SLOW	FAST	SLOW
PreProc	100	400	200	400
MP3Dec	200	800	300	600
PostProc	400	1600	700	1400

the help of power governors. Therefore, we distinguish between local and global governors. Each component is assigned to a local governor responsible for monitoring the components average utilization. In our MP3 decoder example, the local governor implements a hysteresis to select one of the power states SLOW or FAST. A global governor corresponds to a power domain: For a fixed set of components, a global governor receives the selected power states from their local governors and forces a unique power state for all components in this set. Thus, the global governor is an abstraction of power management unit.

5. RESULTS

In this case study, we present results from applying our methodology to a state-of-the-art MPSoC platform for mobile devices. Here, an MP3 decoder is mapped to a given MPSoC platform. Four different mappings of the MP3 decoder to the platform are evaluated for power and performance. The actor-oriented model of the MP3 decoder given in Figure 3 is written in SystemC. The actor implementation has been simplified to model the typical data rate occurring during MP3 decoding. The platform given in Figure 2 is set up by XML configuration files using the power input values and transfer delays from Table 1. Here, the power consumption is normalized to the power consumption caused by the processor (Proc) in FAST power state and EXEC state. This case study focuses on modeling of the digital part of the MPSoC platform. Thus, the power consumption caused by the Audio Front End is assumed to be constant, hence it is not modeled here. Four mapping variants used in this test case are given in Table 2. From an initial single processor mapping in the first variant, the actors are remapped step-by-step to the DSP. Execution times for the actors depending on the mappings are given in Table 3. For each mapping variant, the PlayBack actor is mapped to the

Audio Front End and is activated at a constant period in order to guarantee continuous audio playback. Detecting a buffer underrun for the PlayBack actor implies violation of the performance constraints for the MP3 decoding.

The total energy consumption achieved by our high level power and performance simulation is given in Figure 5. Decoding of 22080 bytes of data was simulated in order to compare the energy consumption for the different variants. Note that the second variant has been used to normalize the absolute energy consumption for comparison. Here, the third variant has a minimal energy consumption of 79.2% compared to the reference variant (var2). The fourth variant potentially could save more energy, but due to higher usage of the FAST power state caused by high load on the DSP the total amount of 79.6% of the energy consumption of the reference variant is consumed in this test case. Individual energy consumptions for each component in the MPSoC platform are given in Figure 6. As expected, the energy consumption caused by the processor decreases, while the energy consumption for the DSP increases if more actors are remapped to the DSP. Energy consumption caused by buses and memory changes slightly, as the major part of the energy consumption is caused by the IDLE mode of these components.

An exemplary trace for power and energy consumption recorded during simulation of architecture variant 3 is shown in Figure 7. In the initialization phase, the buffers between actors are filled up. This results in a high load causing the power governors to switch from the SLOW power state to the FAST power state causing a higher power consumption, as can be seen in Figure 7. Once the buffers are filled, a back-pressure from the PlayBack actor causes a nearly constant average load due to periodic activation of the PlayBack actor. As a result, the platform switches back to the SLOW power state.
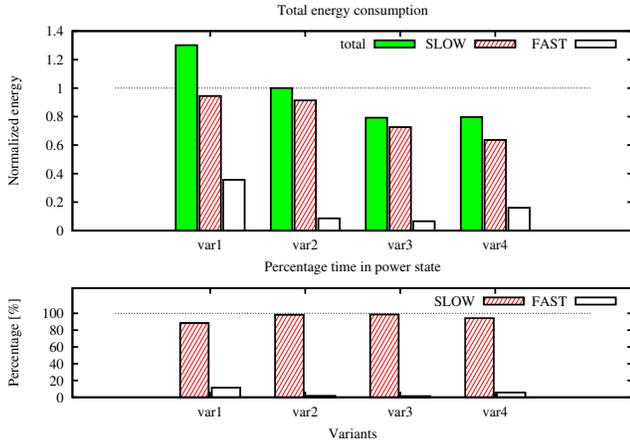
Fig. 5. The normalized energy consumption has been recorded during simulation of all architecture variants. The decoding of 22080 bytes of data has been simulated.
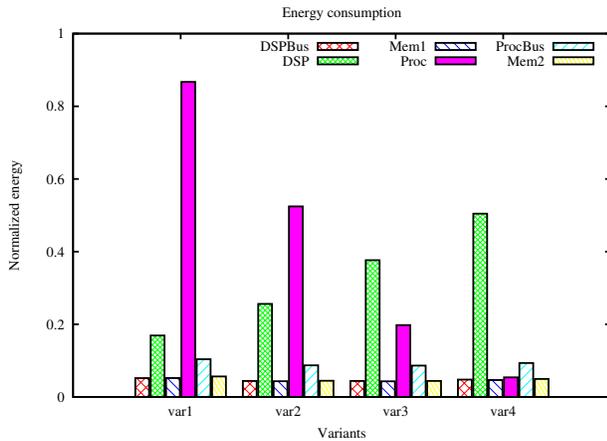

Fig. 7. The power and energy consumption have been traced during simulation of MP3 decoding. During the initialization phase, the governor selects the FAST power state due to high load.

Fig. 6. The energy consumption has been recorded during simulation of the four different architecture variants.

6. CONCLUSIONS

In this paper, a novel power and performance estimation methodology for heterogeneous multiprocessor systems-on-chip has been proposed. We generated system-level power and performance models based on real world measurements from existing platforms and the characterization of power states of all hardware components such as processors, buses, and memory devices. This allows for power and performance estimations at the Electronic System Level, where such results can lead to important design decisions at affordable simulation speed. Strict separation of the system functionality from its architecture allows assessing different design options with minimal effort. Moreover, the simulation-based approach enables to even evaluate the effects of dynamic power management.
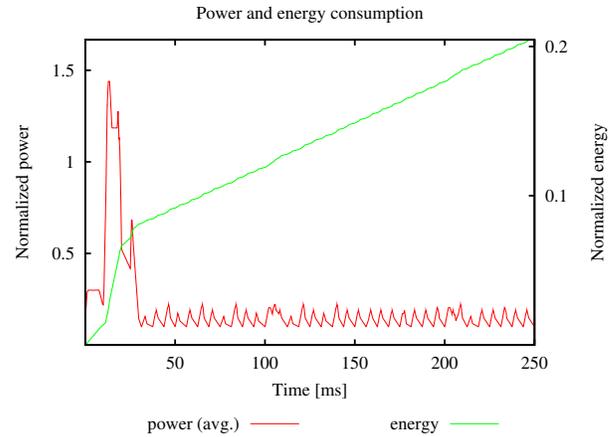
7. REFERENCES

[1] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter van der Wolf, "An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures," in *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, Zurich, Switzerland, July 1997, pp. 338–349.

[2] Joachim Falk, Christian Haubelt, and Jürgen Teich, "Efficient Representation and Simulation of Model-Based Designs in SystemC," in *Proc. of Forum on specification & Design Languages*, Darmstadt, Germany, Sept. 2006.

[3] Martin Streubühr, Jens Gladigau, Christian Haubelt, and Jürgen Teich, "Efficient Approximately-Timed Performance Modeling for Architectural Exploration of MP-SoCs," in *Advances in Design Methods from Modeling Languages for Embedded Systems and SoC's*, Dominique Borrione, Ed., vol. 63 of *Lecture Notes in Electrical Engineering*, pp. 59–72. Springer Netherlands, 2010.

[4] David Brooks, Vivek Tiwari, and Margaret Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proceedings of the International Symposium on Computer Architecture*, New York, NY, USA, 2000, pp. 83–94, ACM.

[5] Wu Ye, Narayanan Vijaykrishnan, Mahmut T. Kandemir, and Mary Jane Irwin, "The Design and use of SimplePower: A Cycle-Accurate Energy Estimation Tool," in *Proceedings of the Design Automation Conference*, 2000, pp. 340–345.

[6] Vivek Tiwari, Sharad Malik, and Andrew Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 437–445, 1994.

[7] Amit Sinha and Anantha P. Chandrakasan, "JouleTrack: A Web Based Tool for Software Energy Profiling," in *Proceedings of the Design Automation Conference*, New York, NY, USA, 2001, pp. 220–225, ACM.

[8] Tony D. Givargis, Frank Vahid, and Jörg Henkel, "Instruction-Based System-Level Power Evaluation of System-on-a-Chip Peripheral Cores," in *International Symposium on System Synthesis*, Los Alamitos, CA, USA, 2000, p. 163, IEEE Computer Society.

[9] Ikhwan Lee, Hyunsuk Kim, Peng Yang, Sungjoo Yoo, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, and Soo-Kwan Eo, "PowerViP: Soc Power Estimation Framework at Transaction Level," in *Proceedings of the Conference on Asia South Pacific Design*, Piscataway, NJ, USA, 2006, pp. 551–558, IEEE Press.

[10] Nagu Dhanwada, Ing-Chao Lin, and Vijay Narayanan, "A Power Estimation Methodology for SystemC Transaction Level Models," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, New York, NY, USA, 2005, pp. 142–147, ACM.

[11] Rabie Ben Atitallah, Smail Niar, and Jean-Luc Dekeyser, "MPSoC Power Estimation Framework at Transaction Level Modeling," in *Internatonal Conference on Microelectronics*, Dec. 2007, pp. 245–248.

[12] Robertas Damaševičius and Vytautas Štuikys, "Estimation of Power Consumption at Behavioral Modeling Level Using SystemC," *EURASIP Journal on Embedded Systems*, vol. 2007, pp. Article ID 68673, 8 pages, 2007, doi:10.1155/2007/68673.

[13] Hugo Lebreton and Pascal Vivet, "Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI '08)*, April 2008, pp. 463–466.

[14] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T. Chakradhar, "Power Monitors: A Framework for System-Level Power Estimation Using Heterogeneous Power Models," in *Proceedings of the International Conference on VLSI Design*, Los Alamitos, CA, USA, 2005, pp. 579–585, IEEE Computer Society.

[15] David Lidsky and Jan M. Rabaey, "Early Power Exploration - A World Wide Web Application," in *Proceedings of the Design Automation Conference*, New York, NY, USA, 1996, pp. 27–32, ACM.

[16] Luca Benini, Robin Hodgson, and Polly Siegel, "System-level Power Estimation And Optimization," in *International Symposium on Low Power Electronics and Design*, New York, NY, USA, 1998, pp. 173–178, ACM.

[17] Reinaldo A. Bergamaschi and Yunjian W. Jiang, "State-Based Power Analysis for Systems-on-Chip," in *Proceedings of the Design Automation Conference*, New York, NY, USA, 2003, pp. 638–641, ACM.

[18] Ankush Varma, Eric Debes, Igor Kozintse, Paul Klein, and Bruce Jacob, "Accurate and fast system-level power modeling: An XScale-based case study," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, 2008.

[19] Feng Liu, QingPing Tan, Xiaoyu Song, and Naeem Abbasi, "AOP-based High-level Power Estimation in SystemC," in *Proceedings of the 20th ACM Great Lakes Symposium on VLSI*. 2010, ACM.

[20] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, Luca Benini, and Matthias Gries, "A Virtual Platform Environment for Exploring Power, Thermal and Reliability Management Control Strategies in High-performance Multicores," in *Proceedings of the 20th symposium on Great lakes symposium on VLSI*. 2010, ACM.

[21] Roberta Piscitelli and Andy D. Pimentel, "A High-Level Power Model for MPSoC on FPGA," in *To appear in Proceedings of the 18th Reconfigurable Architectures Workshop*, Amsterdam, The Netherlands, May 2011, IEEE Computer Society.

[22] Torsten Kempf, Malte Dörper, Rainer Leupers, Gerd Ascheid, Heinrich Meyr, Tim Kogel, and Bart Vanthournout, "A Modular Simulation Framework for Spatial and Temporal Task Mapping onto Multi-Processor SoC Platforms," in *Design Automation & Test in Europe*, Munich, Germany, Mar. 2005, pp. 876–881.

[23] Jürgen Schnerr, Oliver Bringmann, Alexander Viehl, and Wolfgang Rosenstiel, "High-Performance Timing Simulation of Embedded Software," in *Proceedings of Design Automation Conference 2008*, 2008, pp. 290–295.

[24] Forte Design Systems, "http://www.forteds.com," .

[25] Edward A. Lee, Stephen Neuendorffer, and Michael J. Wirthlin, "Actor-Oriented Design of Embedded Hardware and Software Systems," *Journal of Circuits, Systems, and Computers*, vol. 12, no. 3, pp. 231–260, 2003.

[26] Edward A. Lee and Stephen Neuendorffer, "Actor-oriented Models for Codesign: Balancing Re-Use and Performance," in *Formal Methods and Models for System Design*, pp. 33–56. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[27] Hiren D. Patel, Sandeep K. Shukla, E. Mednick, and Rishiyur S. Nikhil, "A Rule-Based Model of Computation for SystemC: Integrating SystemC and Bluespec for Co-Design," in *Proceedings of International Conference on Formal Methods and Models for Co-Design*, 2006, pp. 39–48.

[28] Daniel L. Rosenband and Arvind, "Modular Scheduling of Guarded Atomic Actions," in *Proceedings of the 41st annual conference on Design automation*, 2004, pp. 55–60.