A HYBRID APPROACH FOR SYSTEM-LEVEL DESIGN EVALUATION*

Alexander Viehl[1], Markus Schwarz[1], Oliver Bringmann[1]
and Wolfgang Rosenstiel[1,2]

[1]*FZI Forschungszentrum Informatik*
Haid-und-Neu-Strasse 10-14
76131 Karlsruhe
[viehl,schwarz,bringman]@fzi.de

[2]*Universität Tübingen*
Sand 1
72076 Tübingen
rosenstiel@informatik.uni-tuebingen.de

Abstract: We present a novel approach for performance and power analysis of a system design. For that purpose, we combine coarse grain profiling and established analysis models in a hybrid approach for an early qualitative determination of system properties. Our approach considers the control-flow of communicating processes and the impact of blocking communication on the temporal behavior of the entire system. This can be used for the determination of the system performance and the consideration of the performance impact of dynamic power management. Based on this, the energy consumption of a system can be estimated. The application of our new methodology leads to an inclusion of probabilities concerning system properties and allows an early risk analysis of a design relating to predefined system requirements and constraints.

Keywords: Performance Analysis, Power Estimation, Risk Evaluation

1. INTRODUCTION

Early available design characteristics are essential for designers to validate a design with respect to requirements. Whereas different approaches exist to incorporate worst-case and best-case properties of the system, the resulting intervals (for e.g. performance [12, 5]) are often too pessimistic and distant from

*This work was partially supported by the BMBF project URANOS under grant 01M3075D and by the DFG project ”Communication Analysis for Network-on-Chip" under grant BR 2321/1-1

the real behavior of the system. Moreover, rare cases that appear only once like initial cache misses can expand the intervals of quantitative analysis too much. A resulting architecture of a system concerning these worst-case analysis is quite often overdimensioned. Although this might be acceptable for real-time critical systems, the designer does not know about the typical qualitative and quantitative behavior at an early design stage. Especially in the domains of multimedia and telecommunication as well as real-time embedded systems, there is a demand for early statements about the typical behavior of the system relating to requirements from the specification. In these areas, estimations are sufficient for an early evaluation of the behavior of a design consisting partitially of qualified IP, in which design parameters are configuration specific – for instance mapping, clock frequencies, instruction sets and caches. Besides performance characteristics, mobile applications demand early power estimations of an application platform processing a specific input stream. Without any qualitative characterization of the system behavior, the designer does not know quantitatively how many cases are critical and not met. We present a hybrid approach for qualitative system-level design evaluation in this article. Our novel methodology tries to combine the advantages of both worlds: the derivation of qualitative expressions from profiling and the analytic incorporation without simulating the entire system. In the case of a modification of the system towards exploration, our methodology requires only the determination of the run-times of modified parts.

On one hand, we are using coarse grain profiling for determining the runtime of communication-free parts of a system model. This architectural mapping process is briefly described in Section 3. The resulting quantities are incorporated in an analytic system model, namely the communication dependency graph (CDG) presented in Section 2. Using this graph, we are performing system-level analysis to determine qualitative and quantitative statements about the system behavior. Using the run-times in the activity model we calculate an idle time distribution, which describes how long the control-flow of the processes has to wait in blocking communication nodes. If the qualitative and quantitative behavior of the system relating execution and idle times is known, this information can be used for estimating the energy consumption. For that purpose, we use an established activity driven power model that is briefly presented in Section 2. Furthermore, the drawback of wake-up times from different power states can be incorporated in performance analysis. The derived qualitative information can be further used for an exploration of the underlying system architecture. Our evaluation methodology is presented in Section 4. We applied our methodology for qualitative design evaluation on a real design of a Viterbi decoder. The results are shown in Section 5.

2. RELATED WORK

The internal system behavior and probabilities concerning timing are considered by different analysis models (e.g. generalized stochastic Petri nets (GSPN) [9], stochastic automata networks (SAN) [8] and stochastic automata [4]). These models are used to explicitly model and analyze applications and systems including the internal behavior of a system. Timed Petri Nets [16] are able to represent the internal behavior of a system. Although there exist stochastic extensions by generalized stochastic Petri nets (GSPN) [9, 11] for performance and power estimation, the model does not represent execution times of real system components. Furthermore, synchronization by communication and the specification of communication types have to be modeled explicitly and can not be extracted from executable functional specifications like a SystemC model of a design. System-level performance and power analysis based on Stochastic Automata Networks (SAN) are introduced in [8]. The system including probabilities of execution times is modeled explicitly in SAN. The real execution behavior of the components related to timing and control-flow of a functional implementation is not considered. On the other hand, profiling is used for the determination of characteristics like execution time, resource utilization or memory requirements. This approach is very fine grain and slow. Transaction Level Modeling (TLM) [6] uses models with components on different levels of abstraction for speeding up simulation with a potential loss of accuracy. Profiling or simulating a complete system of parallel processes consumes a lot of computational resources. Moreover, synchronization overhead for coupling multiple simulators is an issue.

Activity Model

To represent the temporal behavior of a system, a model called communication dependency graph (CDG) is used. It was originally developed for formal communication analysis by determining communication instances that synchronize the control-flow of communication partners [13]. We are using this model for the system-level representation of a design. The model considers temporal best-case and worst-case properties of communication and computation. We are extending the model in Section 3 to express qualitative timing properties.

A *communication dependency graph* (CDG) denotes a consolidated representation of a system consisting of communicating processes. Only the temporal and causal behavior between communication endpoints is considered in each communication process. The control-flow is represented by edges connecting communication endpoints. An edge exists if a path in the control-flow graph connects the communication endpoints without passing any other communication endpoint. The communication endpoints are characterized relating

their synchronization behavior, and whether they represent sending or receiving events.

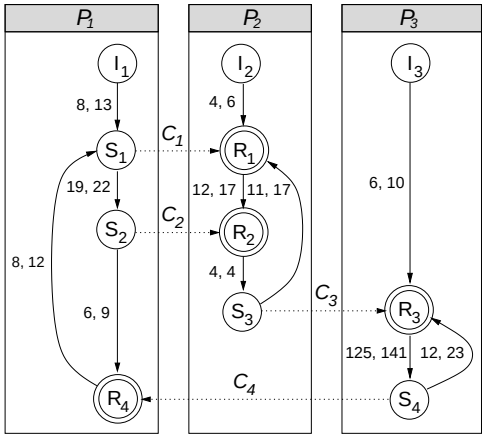


Fig. 1: Communication Dependency Graph

A CDG example consisting of three processes P_1, P_2, and P_3 is depicted in Figure 1. The processes communicate using asynchronous communication instances C_x with non-blocking sender and blocking receiver nodes.

Power Management Model

We use Power State Machines (PSM) [3, 2], to characterize the behavior of dynamic power management of the platform components. The model is flexible enough for describing arbitrary power management schemes. Nodes of

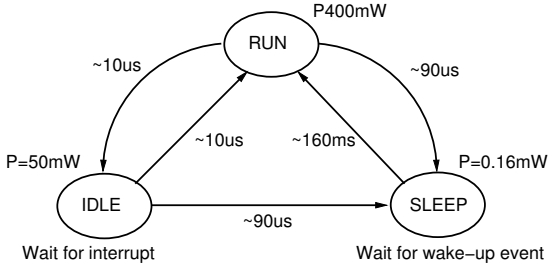


Fig. 2: PSM of SA1100

the PSM characterize different states like `IDLE` or `RUNNING`. Edges between nodes characterize state transitions relating to events. Both nodes and edges can be annotated with expressions of timing and power consumption. This

allows the expression of different properties of dynamic power management (e.g. wake-up penalties). The PSM of a StrongARM 1100 is depicted in Figure 2. It consists of three different power states. State transitions are triggered by inactivity or activation of resources due to external events. We can determine the energy consumption of platform components if a characterization of the temporal behavior of the related component is given.

3. ARCHITECTURAL MAPPING

The starting point of our mapping and exploration flow depicted in Figure 3 is a functional SystemC [10] model of a design, a platform description, mapping information, and the environment of the system. The SystemC model is

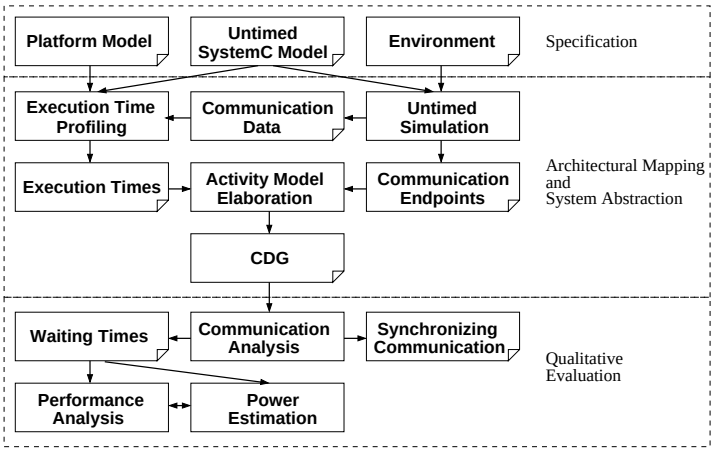


Fig. 3: Mapping and Analysis Flow

simulated together with a specified environment that defines the interaction of the world with the model (e.g. sensor packets for an automotive control system and the temporal interval between the packets). Introspection is applied for gaining access to the communication data sent between SystemC processes. The simulation determines the following two results: temporally ordered communication calls of each process and the data sent through each communication channel.

The ordered communication calls are used to elaborate a consolidated control-flow of each CDG process. The communication data are used as input for determining the run-time of code fragments on the target platform concerning the data computed by the design originating from the environment. For performing execution time profiling, each process from the original SystemC model has to be mapped to the target platform. Therefore, we are using the

tool PORTOS [7] to generate a set of C/C++ files, each including the functionality of one SystemC process. The access to SystemC ports is mapped to special `read(...)` and `write(...)` calls. We are instrumenting the code by linking these calls against communication stubs containing special system calls and assembler routines.

After compilation, each single process is executed on an Instruction Set Simulator (ISS). Therefore, the SimpleScalar [1] ISS was extended to handle the assembler routines that were added to the communication stubs. When a communication function is called, the special syscall is recognized by the extension to the ISS. In the case of write access, two timestamps are taken: one at the beginning of the routine and one at the end. Both timestamps encapsule the computation time of the communication stub. Using the first timestamp with the last timestamp of the previous communication access, the computation time between two subsequent communication of one process is calculated. It is not necessary to store the communication data because these were already collected during untimed SystemC simulation of the entire system. The stubs are implemented to have minimal impact on cache data. Hence, typical communication drivers have an impact on cache data and on execution time as well. These effects can be abstracted and included in the activity model by appending duration times to communication.

In the case of a read access, the syscall is additionally handled by inserting the data of the communication in the buffer address space of the ISS. These data were previously collected by running an untimed SystemC simulation. As a result, the same input data are used by the single processes running on the ISS in comparison to the parallel execution of the whole SystemC model.

In the next step, the collected execution times are combined with the elaborated control-flow to the activity model (i.e. CDG). The same methodology can be used to incorporate synthesized hardware blocks or IP by using HDL simulators instead of an ISS.

Based on the CDG, communication analysis is performed as described in Section 4 for determining the impact of blocking communication on the runtime of the entire system. Further results are the detection of non-synchronizing communication that indicate need for buffer insertion or structural bottlenecks in the system design.

In Section 4 we present the analytic incorporation of this information for qualitative system evaluation relating power and performance.

Execution Time Models

We use two different models of computation for qualitative design evaluation. For the calculation of the behavior concerning the real order of execution times, we extend it with an ordered vector of execution times. This means

that each particular execution time of each edge is stored. This enables analysis of the system performance without any need for determining execution times again. We assume a deterministic temporal behavior of the components relating to input data.

For our estimation approach, we assume that an edge can have n different execution times $lp_1, ..., lp_n$. These execution times appear with probabilities of $P(lp_1), ..., P(lp_n)$. The execution times and probabilities can be simply calculated from profiled execution times. Furthermore, the execution time distribution can be taken from a specification or assumed by the designer.

Activity Model Elaboration

The ordered vector of communication endpoints CE derived from untimed simulation is used as input for an elaboration algorithm. The $ce \in CE$ are characterized by unique IDs. The objective is the elaboration of a consolidated control-flow of each process. It assumes that a deterministic control-flow according to the order of communication endpoints exists. Nevertheless, the algorithm is able to incorporate branches that end at different communication endpoints. If the selection of branches are changing indeterministically, the control-flow is unrolled. If the selection of branches is deterministic, the algorithm detects a recurring state and inserts a backward edge in the elaborated process. Each communication endpoint is characterized by its blocking property, and whether it is $send$ or $receive$. It returns a set of nodes and a set of edges as a representation of one CDG process. At the beginning, new nodes are inserted as long as no other node with the same ID exists. When a node with an already existing ID is inserted, the assumption is that a backward edge to the existing node is inserted. The current state is stored. Then the next nodes from CE are compared with the existing successors of the already existing node in the CDG. If there is a deviation between the order of the nodes in CE and the already inserted nodes, the algorithm uses backtracking and restores the state that was saved before inserting the backward edge. After that, a new node and an edge connecting the node with its successor are inserted. Otherwise, if the same node from which the backward edge was created is reached, the insertion of a backward edge was correct and the state of the system after inserting the nodes from CE is stored. Later, execution times are appended to an edge of the elaborated graph using the ID of enclosing communication endpoints and the order of the communication.

4. QUALITATIVE EVALUATION

Communication Analysis

The goal of communication analysis is the determination of waiting times in blocking communication endnodes. The time the process has to wait in a blocking end node of communication instance C of the CDG is denoted by the slack variable $x(C)$. This time is calculated from the last communication instance C_{pre} that potentially synchronizes the two processes. This means that the control-flow has to wait, if it arrives at the blocking communication endpoint of a communication instance before the control-flow arrives at the non-blocking endpoint. With an iterative algorithm starting at the `init` nodes of each process, we are using already determined slack values $x(C_i)$ for the calculation of subsequent ones. The algorithm terminates, if the vector of system wide slack values recurs. The analysis has to be performed using both models of computation:

I. The determination of system characteristics by incorporating the ordered vector of particular execution times can be used for determining the same performance characteristics as determined by a simulation of the system of communicating processes. A modification of one component only demands for determining the execution time characteristics of this component if the functionality changes. In comparison to our estimation approach (II.), we need to keep the ordered vector of execution times $ls(e)$ instead of summing up the occurrence of equal execution times. Depending on the environment of each component, these data can grow up to some gigabytes for each configuration of a component. For the i-th usage of an edge in system analysis, the i-th value of the execution time vector is used.

II. The incorporation of different execution times and probabilities as distribution function for system property estimation assumes that each execution time of an edge can be followed by each execution time of a subsequent edge. Although the needed operations for communication analysis are based on basic probability theory, issues according the incorporation of waiting times and non-synchronizing communication have to be considered. Waiting times at communication endpoints can not be simply combined with each different execution time of the path leading to this node. When waiting times at blocking communication endpoints are incorporated in path calculation it has to be ensured that they are only combined with the execution times they were calculated from. Otherwise, the analysis itself would lead to a misprediction of the system behavior. Furthermore, if communication is not or not always synchro-

nizing the control-flow of the processes due to timing issues, this impact has to be incorporated as well.

Classification of Run-Times

For reducing computational effort of the estimation approach, different run-times can be classified in groups. Although this may lead to a loss of accuracy, it provides an efficient method for deriving an initial overview about the qualitative system behavior. Arbitrary classification algorithms can be used. In this article, we divide the run-time spectrum of an edge in equidistant intervals as example. This means that all execution times in each interval are combined to one run-time and weighted depending on their probabilities. The probabilities are added. If each interval has a size of w, the maximum number of operations m for e.g. combining the latencies of subsequent paths is m/w^2. Figure 4

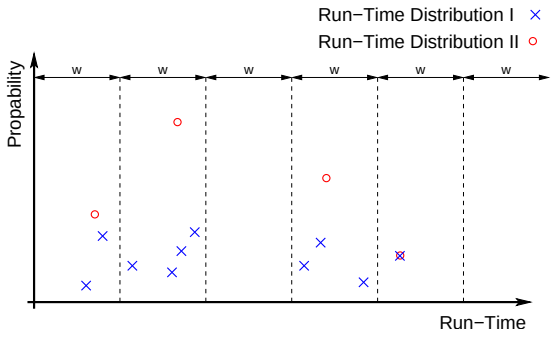


Fig. 4: Classification of Run-Time Distributions

shows a classification example. The run-time distribution II was derived from run-time distribution I using equidistant classification with intervals of size w.

Performance Analysis

Our novel approach for qualitative performance analysis is based on an approach for quantitative system analysis [13]. Performance analysis uses the information on waiting times in blocking communication endnodes that are determined by communication analysis. One direct result from communication analysis is the identification of permanently non-synchronizing communication. This leads to data loss due to high input rates. The probability of data loss can be determined using our methodology as well as the impact of buffer insertion. These information can be used for detection of bottlenecks and exploration of the underlying system architecture. We describe the qualitative evaluation of properties with regard to previously published analysis meth-

ods, like e.g. the determination of worst-case response time (WCRT), system latency, component utilization or access conflicts on shared communication resources [14].

Power Estimation

If a dynamic power management specification is given by a PSM, the impact of wakeup-times on the performance of a system can be determined. If the PSM is not triggered by any other event, power management is activated when the system is idle. We calculated idle phases of processes waiting on a communication C as $x(C)$ by communication analysis. The potential drawback of wake-up times on the activation of processes can be expressed by $\varphi_{PSM}(x(C))$. We have to include these additional waiting times in our method for calculating subsequent slack variables. According to Section 4,

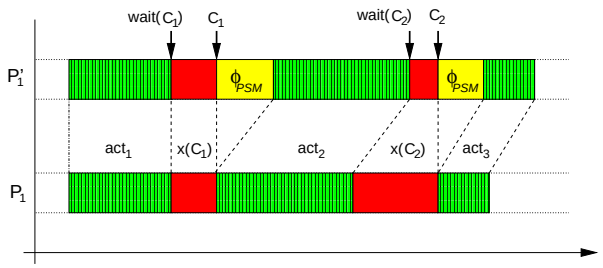


Fig. 5: Run-Time Impact of Dynamic Power Management

$|\varphi_{PSM}(x(C))|$ is added to each path that traverses a communication C or its blocking communication endpoint. For determining the energy consumption, we apply the PSM on the information about computation times stored in the CDG and on the information about $x(C)$ and $\varphi_{PSM}(x(C))$. Figure 5 depicts an example of a process $P1$ without and $P1'$ with dynamic power management. Although the activity times act_x in both examples are identical, the waiting times $x(C)$ and the impact of the PSM defined by $\varphi(x(C))$ are different. The reduced performance of this process may lead to a need for buffering, the violation of deadlines or data loss.

Architectural Exploration

Our analysis does not only allow to determine the characteristics of computation ressources. We can use the information about communication and the temporal relation between them to determine the impact of the communication infrastructure on system performance and power characteristics if typical parameters like latency, duration of communication, and a specification of

activity-based power management are given. Our approach allows an exploration towards the functionality, the environment, the mapping and the underlying platform. A modification of components or an exchange of complete subsystems can be incorporated without completely analyzing the system again. This leads to a structured reuse of qualified IP in platform based design.

5. EXPERIMENTAL RESULTS

Experimental results from the application of our methodology are presented in this section. All analysis steps were implemented and integrated into the SysXplorer [15] tool. We started with a SystemC model of a Viterbi decoder and mapped the two processes with the main functionality, `vit_fwd` and `vit_bwd` each to a PowerPC 604 with 100 MHz according to the flow presented in Section 3. The resulting elaborated CDG representing the communication structure of the communicating processes is depicted in Figure 6. The edge weights characterizing the latencies of the memories were annotated

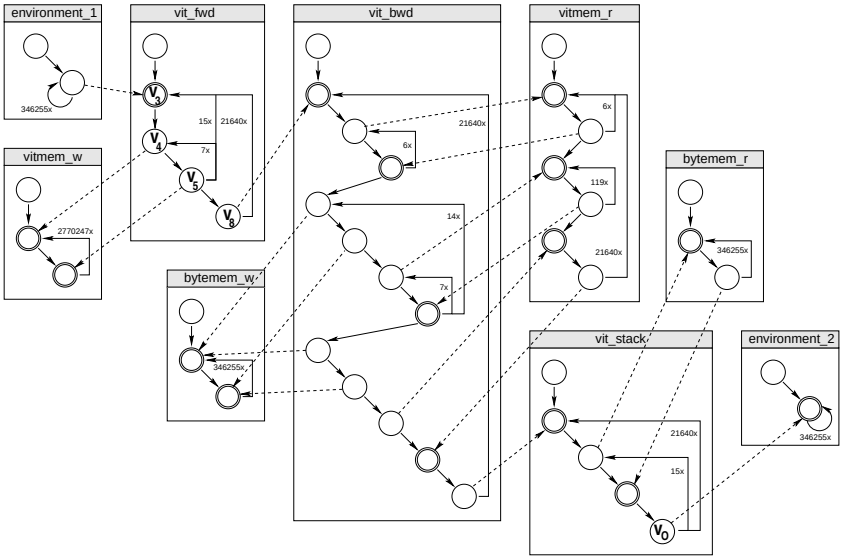
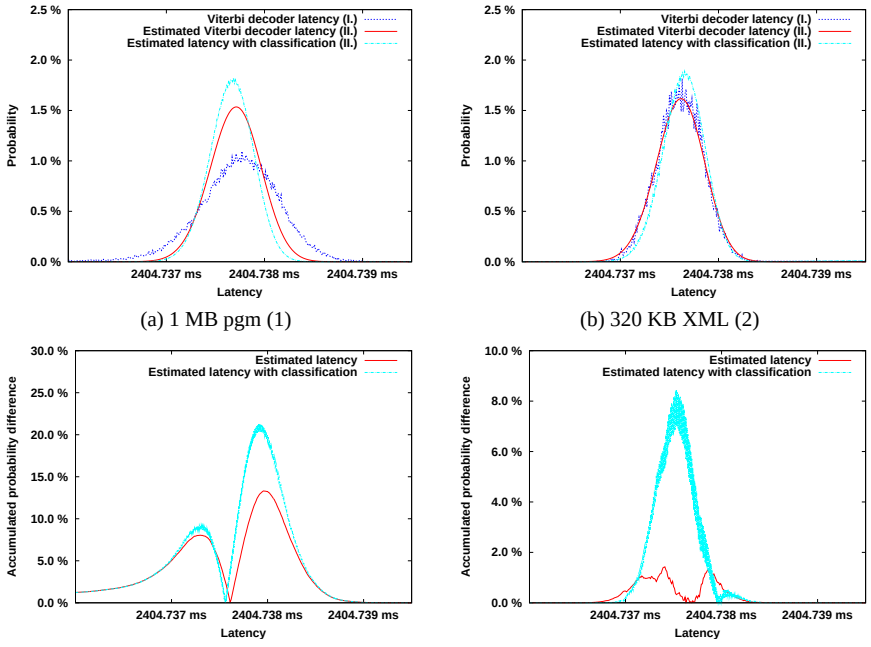


Fig. 6: Viterbi Decoder CDG

with 60 ns from the memory specification. The same latencies were used for the module `vit_stack` that has the same latency characteristics like memory. We profiled the run-times relating to two different environments the decoder processed. We used a 1 MB pgm image as one input and a 320 KB XML file as another. The data packages arrived with an inter arrival time of 160.3 ms. Of course, this arrival time model can be modified using probabilities and inter-

vals. Then we used the information derived from profiling and calculated the latency of the entire design between the nodes v_3 and v_O using both models of computation introduced in Section 3. A closer inspection on a comparison between particular packet latency calculation, latency estimation and latency estimation with weighted classification is depicted in Figure 7a and 7b. The classification uses equidistant intervals of 50 ns. For a comparison of the three results, we subtracted the accumulated values of both estimated curves from the particular packet curve. These results are depicted in Figure 7c and 7d. Although the estimated curve with classification does not exactly match the calculated curve using the particular execution times, it delivers a quite good approximation relating the small derivation in absolute numbers. We applied



(a) 1 MB pgm (1)

(b) 320 KB XML (2)

(c) Cumulated Difference (1)

(d) Cumulated Difference (2)

Fig. 7: Comparison of Accuracy

the PSM depicted in Figure 2 to both cores. The comparison of the design latencies with and without dynamic power management is depicted in Figure 8. We used the data annotated in the PSM to estimate the energy consumption of both processor cores for computing 16 packets. The transition costs and wake-up penalties defined by the PSM reduced the impact of variable run-times on different results in energy estimation because the activation request appeared while PSM transition to `IDLE` (`vit_fwd`) and to `SLEEP` (`vit_bwd`).

		Energy Consumption (mWs)		**Latency** (ms)
		`vit_fwd`	`vit_bwd`	($v_3 \rightsquigarrow v_o$)
w/o	DPM	1025.91	1025.92	2404.73
with	DPM	1025.31	65.43	2725.13

Fig. 8: Analysis Results

Furthermore, estimation results for energy consumption are given in Figure 8. Dynamic power management of the core executing `vit_fwd` only marginally improves the energy consumption but the system latency increases due to wake-up penalties. A closer inspection on the analysis results determined that the activating input events occur after the power state of the core started transition between `IDLE` and `SLEEP`. This means that the energy intensive wake-up procedure started just after beginning the transition. Due to this, the energy consumption is still at a high state and wake-up penalties reduce the system performance. The internal control-flow of `vit_fwd` sends one resulting output packet to `vit_bwd` from 16 received input packets.

Due to this, arrival-time intervals of the communication instance between `vit_fwd` and `vit_bwd` are much longer. As consequence, the PSM of `vit_bwd` stays longer in the energy efficient state `SLEEP`. The average energy consumption of the core executing `vit_bwd` is reduced by $93.6\ \%$ in comparison to the configuration without power management. Nevertheless, PSM wake-up penalties increase the latency of the component and its performance falls off. Finally, the latency of the whole decoder is $13.3\ \%$ higher than without dynamic power management.

6. CONCLUSION

The early availability of quantified qualitative design characteristics for comparing system requirements with the design properties is valuable for designers. We developed a hybrid approach for incorporating run-time profiling and system-level analysis techniques targeting this issue. Two different models of computation were presented: One for incorporating each particular execution time from profiling and one that combines all execution times to a distribution function. Our analysis approach is based on the determination of communication that synchronize the control-flow of communicating processes. Based on waiting times in blocking communication endnodes, we calculate qualitative performance and power properties of the system. We implemented our analysis methods and presented experimental results from a real-world design, a Viterbi decoder. The comparison of both approaches shows, the hybrid estimation approach allows quite accurate requirement evaluation at system-level. Next steps will be the analytic exploration of power management policies based on

the results of communication, performance and power analysis with regard to predefined constraints.

REFERENCES

[1] Todd Austin, Eric Larson, and Dan Ernst. Simplescalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[2] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. pages 231–248, 2002.

[3] Luca Benini, Robin Hodgson, and Polly Siegel. System-level power estimation and optimization. In *ISLPED '98: Proceedings of the 1998 international symposium on Low power electronics and design*, pages 173–178, New York, NY, USA, 1998. ACM Press.

[4] Jeremy Bryans, Howard Bowman, and John Derrick. Model checking stochastic automata. *ACM Trans. Comput. Logic*, 4(4):452–492, 2003.

[5] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-Based Embedded System Designs. In *Proceedings of DATE*, Munich, 2003.

[6] Adam Donlin. Transaction level modeling: flows and use models. In *CODES+ISSS '04*, New York, NY, USA, 2004. ACM Press.

[7] Matthias Krause, Oliver Bringmann, and Wolfgang Rosenstiel. Target Software Generation: An Approach for Automatic Mapping of SystemC Specifications onto Real-Time Operating Systems. *Springer: Design Automation for Embedded Systems*, 2007.

[8] R. Marculescu and A. Nandi. Probabilistic application modeling for system-level performance analysis. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 572–579, Piscataway, NJ, USA, 2001. IEEE Press.

[9] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, 1984.

[10] Wolfgang Müller, Wolfgang Rosenstiel, and Jürgen Ruf, editors. *SystemC: methodologies and applications*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[11] Qinru Qiu, Qing Wu, and Massoud Pedram. Dynamic power management of complex systems using generalized stochastic petri nets. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 352–356, New York, NY, USA, 2000. ACM Press.

[12] Simon Schliecker, Matthias Ivers, and Rolf Ernst. Integrated Analysis of Communicating Tasks in MPSoCs. In *CODES+ISSS '06*. ACM Press, 2006.

[13] Axel Siebenborn, Oliver Bringmann, and Wolfgang Rosenstiel. Worst-case performance analysis of parallel, communicating software processes. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, 2002.

[14] Axel Siebenborn, Alexander Viehl, Oliver Bringmann, and Wolfgang Rosenstiel. Control-Flow Aware Communication and Conflict Analysis of Parallel Processes. In *Proceedings of the 12th Asia and South Pacific Design Automation Conference ASP-DAC 2007, Yokohama, Japan*, 2007.

[15] SysXplorer Home. www.fzi.de/sim/sysxplorer.html.

[16] A. Yakovlev, L. Gomes, and L. Lavagno. *Hardware Design and Petri Nets*. Kluwer, 2000.