Dipl.-Ing. Christian Bachmann

Automated Power Emulation Methodology For Power-Aware Hardware/Software Codesign of SoCs

---

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr. techn.)

durchgeführt am Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Graz, im März 2011

Dedicated to my family:
Minu kallis Mari, my parents and my brother.
For your continuous support and understanding.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen / Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, the ....
(Signature)

Kurzfassung

Insbesondere im Hardware/Software Codesignprozess von Systems-on-Chip (SoC), die für den Einsatz in mobilen Geräten bestimmt sind, spielt die Berücksichtigung der Leistungsaufnahme eine immer wichtigere Rolle. Die Komplexität dieser Systeme nimmt durch die höher werdenden Performance-Anforderungen neuer Applikationen stetig zu. Die Fortschritte im Bezug auf die Energiespeicherung sowie das Nutzen der Umgebungsenergie ("Energy Harvesting") hinken dieser Entwicklung allerdings hinterher und verschärfen somit die Forderung nach SoC-Designs mit niedrigem Energiebedarf. Die steigende Komplexität wirkt sich auch erschwerend auf die Untersuchung der Leistungsaufnahme im Rahmen des Hardware/Software Codesigns aus, da die Simulation des gesamten Systems immer aufwändiger wird und erhebliche Simulationsdauern nach sich zieht. Der Hardwarebeschleunigte Ansatz zur Emulation der Leistungsaufnahme eines SoC-Designs, auch als "Power Emulation" bezeichnet, stellt eine vielversprechende Alternative zu Softwaresimulatoren dar. Die Abschätzung der Leistungsaufnahme kann parallel zur funktionalen Emulation mit hoher Geschwindigkeit durchgeführt werden. Allerdings operieren ursprüngliche Power-Emulations-Ansätze auf niedrigem Abstraktionsniveau, verursachen daher einen hohen zusätzlichen Hardwareaufwand und eignen sich nur beschränkt für die Emulation ganzer Systeme. Ansätze auf höherer Abstraktionsebene ("High-Level Power Emulation") verursachen nur geringen Zusatzaufwand und erlauben dadurch höhere Emulationsgeschwindigkeiten. Der Erstellung einer generischen Methodik, die die Verwendung des High-Level Power-Emulations-Ansatzes im Entwurfsprozess vereinfacht, wurde bis jetzt allerdings nur wenig Aufmerksamkeit geschenkt. Des Weiteren ist die Verwendung der Power Emulation, unter Berücksichtigung von Anforderungen an Leistungsaufnahme und Performance, im Hardware/Software Codesign bisher wenig untersucht.

Diese Arbeit stellt eine automatisierte Methode vor, um die High-Level Power Emulation von SoC-Designs zu ermöglichen. Zu diesem Zweck wird sowohl die automatisierte Erstellung von High-Level Modellen der Leistungsaufnahme als auch die automatisierte Hardware-Implementierung dieser Modelle behandelt. Die resultierende High-Level Power-Emulations-Plattform wird anschließend im Hardware/Software Entwurfsprozess verwendet. Durch die Integration der zur Laufzeit geschätzten Leistungsaufnahmeprofile in eine Software-Entwicklungsumgebung, wird die Software-Optimierung unter Berücksichtigung der Leistungsaufnahme stark vereinfacht. Des Weiteren können die Profile für die automatische Detektion und Optimierung von Spitzen in der Leistungsaufnahme, die die Stabilität des Systems gefährden, verwendet werden. Die zusätzliche Hardware-basierte Überwachung der Performance des Systems ermöglicht Hardware/Software-Optimierungen bezüglich der Leistungsaufnahme und des Energieverbrauches bei gleichzeitiger Berücksichtigung der Auswirkungen auf die Performance des Systems.

Anhand zweier, prototypischer SoC-Designs, deren jeweilige Anwendungsdomäne einen überaus energieeffizienten Entwurf erfordert, wird die vorgestellte Methode erfolgreich evaluiert. Durch die Erstellung einer High-Level Power-Emulations-Plattform für beide Systeme wird die Anwendbarkeit im Hardware/Software Codesignprozess dargestellt.

Abstract

Power-awareness has become increasingly significant in the hardware/software codesign process of system-on-chip (SoC) designs, particularly for mobile devices. With the increasing complexity of SoCs, fueled by new applications and higher performance requirements, and combined with the expected slow progress in the enhancement of energy storage devices as well as energy harvesters, the low-power design requirement for these systems will become even more important. The rising design complexity also affects the simulation-based power consumption profiling and analysis within the hardware/software codesign process, as the simulation of entire systems becomes increasingly difficult due to extensive simulation times. Hardware-accelerated power emulation techniques represent a promising alternative to software simulators, performing the power estimation process at high speeds alongside the standard functional emulation of the system-under-test. While low-level power emulation approaches suffer from large area overheads that complicate their use for the full-system emulation of large designs, high-level power emulation approaches entail only low overheads and allow for high emulation speeds. However, only little attention has been awarded so far to the problem of devising a generic methodology for automatically enabling the high-level power emulation of a given system-under-test. Furthermore, the use of the high-level power emulation technique in the codesign process for both power- and performance-aware hardware/software power optimization is little explored.

This work presents an automated power emulation methodology that aims at enabling the high-level power emulation of novel SoC designs. To this end, the methodology addresses both the automated high-level power model creation and the automated implementation of the derived power model in hardware. The resulting power emulation platform can then be used in the power-aware hardware/software codesign process. The integration of the run-time power estimates, generated by the platform, into a standard software development environment allows for truly power-aware software optimizations. Furthermore, the power profiles can be employed to automatically detect and reduce software-induced power consumption peaks that are threatening the system's stability. By additionally enabling the monitoring of hardware performance events, hardware and software power optimizations can be performed while considering their performance impact.

The automated power emulation methodology is successfully evaluated on two prototypical SoC designs that are operating in very power- and energy-constrained environments. By creating high-level power emulation platforms for both test systems, the applicability of the high-level power emulation technique in the hardware/software codesign process is illustrated.

Acknowledgements

This dissertation is part of the POWERHOUSE project that constituted a successful collaboration between the Institute for Technical Informatics at the Graz University of Technology, the Infineon Design Center Graz as well as the AustriaCard GmbH, Vienna. I would like to thank all the people from these three organizations who supported me and my work in the course of this dissertation. As it is not possible to mention everyone, I would like to name but a few.

I would like thank Prof. Reinhold Weiß of the Institute for Technical Informatics, for providing the research facilities as well as his guidance and helpful advice in the course of composing this dissertation. I am especially grateful to Christian Steger for his thoughtful supervision of the project, beneficial feedback and comprehensive advice. I would also like to thank all students contributing to this project: Daniel Reitz, Michael Schön, Daniel Wittibschlager, Michael Lackner, Stephan Gether and Michael Schaffernak. Special thanks also go to my PhD colleagues Armin Krieg and Johannes Grinschgl for providing beneficial feedback on this thesis.

Furthermore, I am particularly grateful to our project leader Josef Haid of Infineon, for the great amount of guidance, advice and inspiration he contributed to this dissertation. I would like to thank the entire team of the Infineon CC Concept Engineering group for their support and for welcoming me with open arms: Dietmar Scheiblhofer for providing a great working environment in his group and for all system-level support. Bernd Zimek, Thomas Leutgeb, Albert Missoni for sharing their deep knowledge regarding analog- and RF-related topics. Robert Hofer and Christian Goral for always being very helpful in investigating all aspects of firmware and software development. Furthermore, I would like to thank Holger Bock for his great effort in project funding and project management support. I would also like to thank Klaus Holler for always sharing his extensive EDA knowledge and for his helpful assistance in solving the numerous EDA-related issues.

I would like to thank Christiane Ulbricht of AustriaCard for providing beneficial insights into customer requirements for power-aware software development and helpful feedback. Furthermore, I would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology for providing us with funding for the POWERHOUSE project under the FIT-IT contract FFG 815193. Without this funding my dissertation would not have been possible.

Special thanks go to my colleague Andreas Genser for the perfect collaboration during the last three years. Our inspiring discussions based on his deep understanding of hardware/software codesign and his numerous well-founded comments constitute a considerable contribution to the success of this project.

Finally, I would like to express my gratitude to my family for their continuous support in the course of my studies and my dissertation. I owe my deepest gratitude to my dear Mari for her great support, her patient love and her understanding during this challenging period of both our lives.

Graz, March 2011 Christian Bachmann

Extended Abstract

Rapid advances in the integration level of semiconductor devices during the past four decades have enabled the realization of entire *Systems-on-Chip* (SoCs) that form the basis for a multitude of novel applications. The required functionality for these applications is in most cases achieved by the interplay of hardware and software components. These components are typically designed in a concurrent approach that is referred to as *hardware/software codesign*. Particularly in - but not limited to - the codesign process of mobile devices, *power-awareness* has become increasingly significant. In fact, the power consumption is considered as one of the major design constraints besides the speed and silicon area constraints.

While the complexity of future systems-on-chip is expected to increase further over the coming years, progress in extending the capacity of energy storage devices and in improving the efficiency of energy harvesters is forecast to be lagging behind. Therefore the low-power and low-energy design requirements will become even more stringent. For meeting these requirements, advanced power management techniques will have to be used, such as fine-grained clock- and power gating, dynamic voltage and frequency scaling (DVFS) as well as special low-power and hibernation states of system components. These power management features further increase the system's complexity and have to be considered by both hardware and software engineers in all stages of design and verification.

The rising design complexity poses a challenge to the purely simulation-based power consumption analysis and verification within the hardware/software codesign process. The full-system simulation of large designs is becoming more and more difficult due to extensive simulation times. By raising the level of abstraction employed in the simulation a higher simulation performance can be achieved but the simulation accuracy decreases. This increases the risk of concealing the low-level implications and side effects of design decisions and optimizations.

With the availability of *field programmable gate arrays* (FPGAs), able to contain large system designs, *emulation-based* power estimation approaches have become a promising alternative to software simulators. The *power emulation* technique, i.e., the hardware-accelerated power estimation process, allows for generating cycle-accurate power consumption estimates at high speeds. Initial low-level power emulation approaches, however, suffer from extensive overheads in terms of required FPGA resources, rendering the power emulation of large systems infeasible. Therefore, high-level power emulation approaches have been introduced that only entail small overheads and allow for high emulation speeds.

The POWERHOUSE project, in which this work is embedded, aims at harnessing the benefits of hardware emulators for accelerating the power-aware hardware/software codesign process. To this end, a high-level, run-time power emulation methodology is devised and integrated into the codesign process. The resulting power emulation platform

speeds-up the power-aware hardware/software codesign process and enables the rapid power management exploration for novel system designs.

In this work an automated power emulation methodology for use in a power-aware hardware/software codesign process of systems-on-chip is presented. This methodology and its main components are depicted in Figure 1.

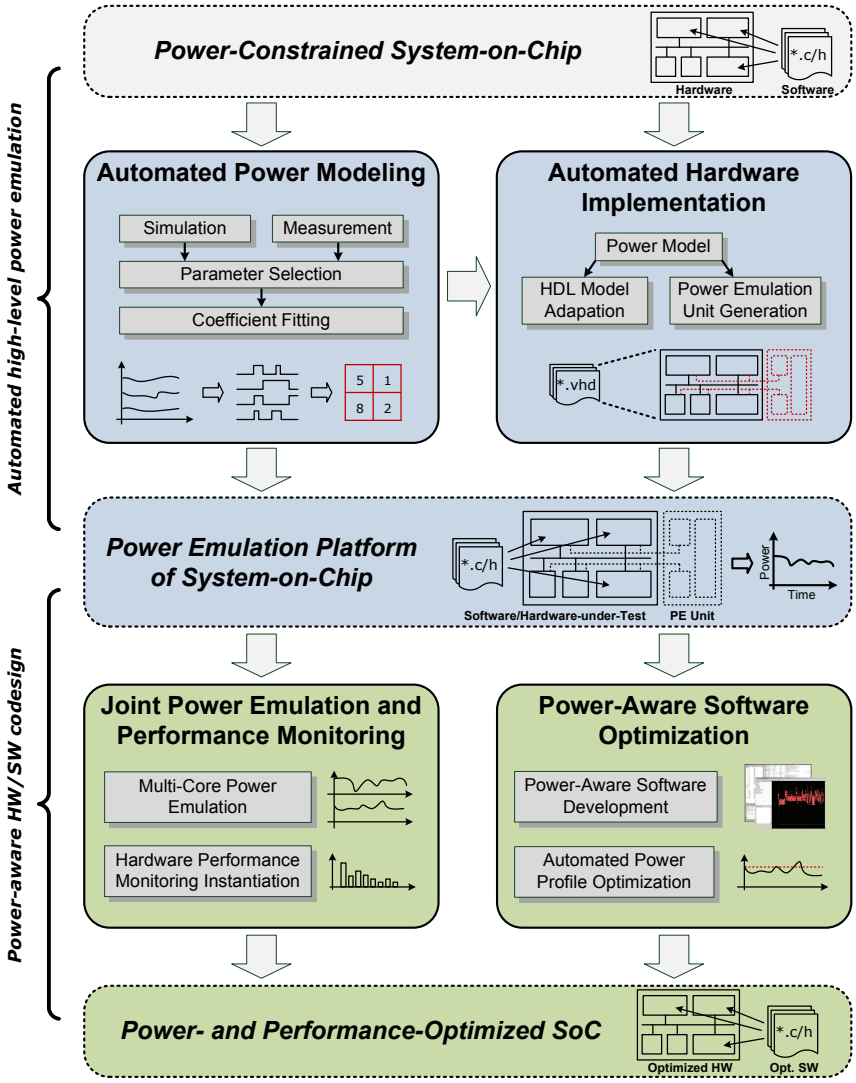

Figure 1: Automated power emulation methodology for power-aware hardware/software codesign

The proposed methodology aims at automatically enabling the power emulation of a given system-under-test. To this end, an *automated power modeling* step creates a high-level power macromodel that captures both dynamic and static power consumption of the given system[1]. Based on a set of microbenchmarking applications, a training set consisting of power consumption estimates and activity data is derived using state-of-the-art

[1] *Automated Power Characterization for Run-Time Power Emulation of SoC Designs*, 13th IEEE Euromicro Conference on Digital System Design: Architectures, Methods and Tools 2010 (DSD '10), Lille, France, 1-3 Sept. 2010.

simulation tools. At later design stages, i.e., when first tape-out silicon is available, power consumption measurement data can be integrated into the training set. The training set data are analyzed in a subsequent parameter selection stage that aims at identifying internal signals of which the activity data correlate well with the power consumption data. These signals are then used as power model parameters in the high-level power macromodel. Finally, a model coefficient fitting step employs a least squares fitting algorithm to derive coefficients for the chosen parameters.

For enabling the power emulation of the given system-under-test this high-level power model needs to be implemented in hardware. For this purpose an *automated hardware implementation* step has been devised that replaces the time-consuming and error-prone manual hardware implementation of the power emulation functionality[1]. First, the hardware description language (HDL) model of the system is analyzed and modified to allow for the monitoring of all internal signals that are employed by the high-level power macromodel. Second, the power emulation hardware, i.e., the *power emulation unit*, is generated from HDL templates and its internal structure is adapted to the used power model. This power emulation unit monitors all signals representing power model parameters and thereby generates cycle-accurate power estimates during the run-time of the system-under-test.

After performing the automated power modeling and hardware implementation, the resulting *power emulation platform* of the given system-on-chip design can be utilized in the power-aware hardware/software codesign process[2]. By additionally introducing a *joint power emulation and performance monitoring* approach, the run-time monitoring of hardware performance indicators allows designers to evaluate design trade-offs between the power/energy consumption and the performance of the system[3]. Furthermore, the automated power emulation methodology has been extended to enable the joint power emulation and performance monitoring in heterogeneous multi-core environments.

The application of the power emulation methodology for enabling *power-aware software optimization* represents a main goal of the POWERHOUSE project. To achieve this goal, the generated run-time power estimates are being integrated within a standard software development environment[4]. The analysis of the recorded power consumption traces provides valuable power feedback to software engineers and allows for refined power-aware software optimization.

The run-time power estimates can also be used for the automated *detection and reduction of power consumption peaks*. Especially in power constrained mobile systems these power peaks can lead to supply-voltage drops below critical limits and, hence, threaten the

[1] *Automated Power Characterization for Run-Time Power Emulation of SoC Designs*, 13th IEEE Euromicro Conference on Digital System Design: Architectures, Methods and Tools 2010 (DSD '10), Lille, France, 1–3 Sept. 2010.

[2] *An Emulation-Based Real-Time Power Profiling Unit for Embedded Software*, 9th IEEE International Symposium on Systems, Architectures, Modeling, and Simulation 2009 (SAMOS '09), Samos, Greece, 20–23 July 2009.

[3] *An Emulation-Based Platform for Power- and Performance-Aware HW/SW Development of Embedded Multi-Core Systems*, submitted for publication / under review, 2011.

[4] *Accelerating Embedded Software Power Profiling Using Run-Time Power Emulation*, 19th International Workshop on Power and Timing Modeling, Optimization and Simulation 2009, (PATMOS '09), Delft, The Netherlands, 9–11 Sept. 2009, published in Springer Lecture Notes in Computer Science, 2010, Volume 5953.

system's reliability. For analyzing and optimizing power consumption peaks during the design phase, an automated power peak reduction framework has therefore been introduced that automatically detects and reduces software-induced power consumption peaks[5]. Depending on the software-controllable power management features available on the given system an optimization strategy is chosen and applied to the software application's source code. Furthermore, a run-time power profile flattening approach based on the hardware-accelerated power estimation technique has been presented that eliminates the need for costly on-chip measurement hardware[6].

To summarize, this dissertation introduces an automated power emulation methodology that aims at automatically enabling the power emulation of novel system-on-chip designs. By covering both the high-level power macromodel creation and the subsequently required hardware implementation of the power model, the effort for enabling power emulation in a hardware/software codesign environment is vastly reduced. Furthermore, this work illustrates the benefits of using this high-level power emulation approach for power- as well as performance-aware hardware and software optimization.

[5] *An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software*, 20$^{\text{th}}$ International Workshop on Power and Timing Modeling, Optimization and Simulation 2010, (PATMOS '10), Grenoble, France, 8–10 Sept. 2010, published in Springer Lecture Notes in Computer Science, 2011, Volume 6448.

[6] *Estimation-Based Run-Time Power Profile Flattening for RF-Powered Smart-Card Systems*, 11$^{\text{th}}$ IEEE Asia Pacific Conference on Circuits and Systems, (APCCAS '10), Kuala Lumpur, Malaysia, 6–12 Dec. 2010.

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
ASIP	Application-Specific Instruction-Set Processor
CAN	Controller Area Network
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chip Multi-Processor
CS	Control Set
DES	Data Encryption Standard
DVFS	Dynamic Voltage and Frequency Scaling
EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HW	Hardware
IEEE	Institute of Electrical and Electronics Engineers
ITRS	International Technology Roadmap for Semiconductors
LUT	Look-Up Table
MAC	Multiply-Accumulate
NFI	Non-Functional Instruction
NNLS	Nonnegative Least Squares
NVM	Non-Volatile Memory
MPSoC	Multi-Processor System-on-Chip
NoC	Network-on-Chip
PE	Power Emulation
RFID	Radio Frequency Identification
RAM	Random-Access Memory
RMS	Root-Mean-Square
RNG	Random Number Generator
ROM	Read Only Memory
RSA	Rivest-Shamir-Adleman Public Key Cryptography
RTL	Register Transfer Level
SoC	System-on-Chip
SMP	Symmetric Multi-Processing
SPARC	Scalable Processor Architecture
SW	Software
TRNG	True Random Number Generator
TS	Training Set
UART	Universal Asynchronous Receiver / Transmitter
VGA	Video Graphics Array
VLIW	Very Long Instruction Word

Chapter 1

Introduction

1.1 Motivation

The rapid advances in the integration level of semiconductor devices during the past four decades, as predicted by Moore's law[1] [1], have drastically increased the performance and the functionality of integrated circuits (ICs). At the same time these advances have made it possible to decrease the chip area and, hence, to decrease the costs. As a consequence, the integration of entire systems on a single chip ("systems-on-chip") has become feasible.

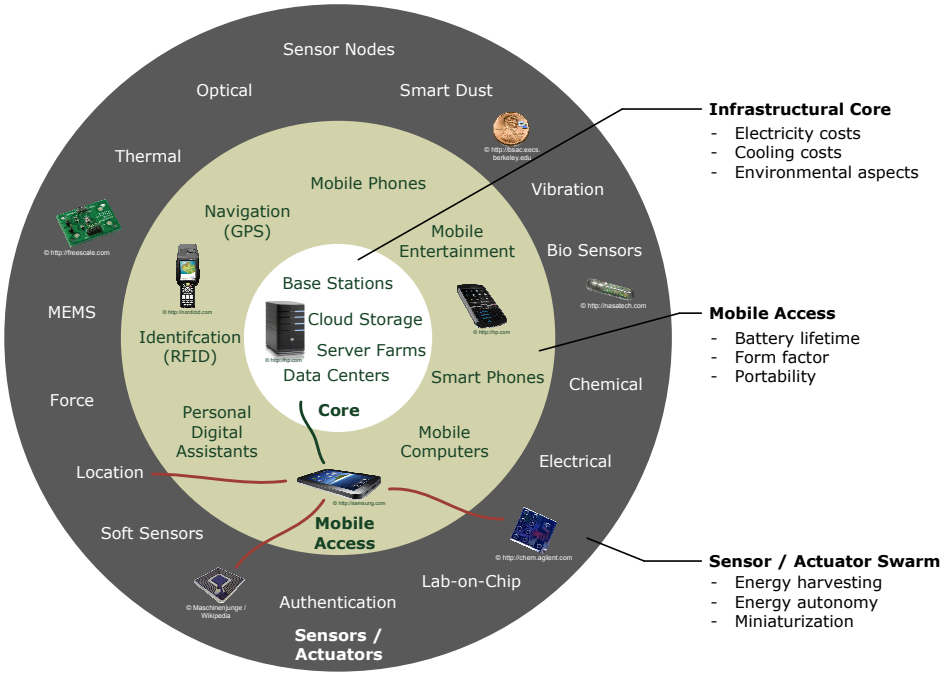

Figure 1.1: Power-awareness in the IT systems of the future ([2], with modifications)

The advent of these technologies paves the way for a multitude of new applications and may lead to a future IT platform as illustrated in Figure 1.1. The envisioned IT

[1]The doubling of components per chip roughly every 18 months.

platform [2], consists of three main layers: (1) an infrastructural core providing most of the computational resources as well as data storage capacities, (2) a mobile access layer enabling user interaction as well as serving as a gateway to the core and (3) a swarm of mostly minuscule sensor and actuator nodes.

For devices on all of these layers a minimization of power and energy consumption is of increased interest for several reasons [3]: In large-scale data and computing centers on the one hand a reduction of operational costs for electricity and cooling is the main motivator. Besides, the minimization is also consistent with the global awareness of environmental issues. For mobile devices and sensor/actuator nodes on the other hand, the stringent low-power and low-energy requirements are determined by the limitations of the used energy storage or energy harvesting devices. Furthermore, device form factors and the need for miniaturization play an important role.

1.1.1 The Need for Power-Awareness in Hardware/Software Codesign

A great majority of the systems envisioned in this future IT platform will possess the required functionality and abilities thanks to the interplay of hardware and software components. The design of these components is typically carried out in a concurrent matter that is referred to as *hardware/software codesign* [4]. In this codesign process the power consumption has become an increasingly stringent constraint.

Regarding hardware design, power-awareness has become a major issue in CMOS-based integrated circuit (IC) design since the early 1990s. At that time steep power dissipation increases, caused by exponentially rising microprocessor operating frequencies, sparked the interest in low-power design [5]. In the following years the power consumption constraint was added to the previously existing IC design constraints regarding speed and silicon area. For software design power-awareness similarly plays an important role, as software can significantly influence average, peak and instantaneous power of the hardware execution platform [6].

A number of power-aware optimizations, such as clock and power gating [7, 8], guarded evaluation [9] and dynamic voltage and frequency scaling (DVFS) [10], have been introduced, aiming at the reduction of the power consumption. The verification of power constraints and the evaluation of the effectiveness of power-aware optimizations represent significant parts of the hardware/software codesign flow as depicted in Figure 1.2.

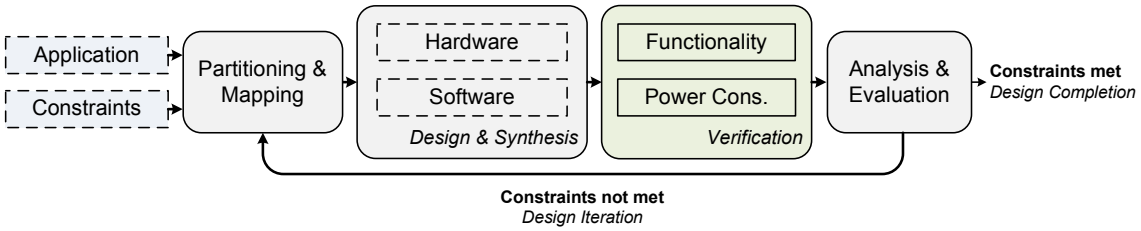

Figure 1.2: Generic hardware/software codesign flow with focus on power-awareness

1.1.2 Increasing SoC Design Complexity Challenge

Ever increasing demands for higher performance and new functionality, fueled by new applications, are the drivers for a continued rise in design complexity. This rising complexity poses a challenge to hardware and software development as well as to verification.

For a typical consumer portable system-on-chip (SoC) device, as it could be envisioned on the mobile access layer in the example introduced before, Figure 1.3 illustrates the *International Technology Roadmap for Semiconductors* (ITRS) [11] forecast regarding the increase in the number of processing engines. Each engine represents a specifically customized processor that is used alone or in conjunction with other engines to implement a given function [11].

The increase in design complexity is also reflected by an increase in forecast power consumption levels of these consumer portable SoCs as shown in Figure 1.4. This increase is affecting both the dynamic as well as the static power consumption of logic and memory components. While phases of total power reduction are also expected, due to the assumed introduction of new technologies and a decrease in supply voltage, the overall trend is clearly indicating an increasing power consumption. Note that the forecast power consumption requirement remains unaltered due to slow progress in extending the capacity of energy storage devices and in improving the efficiency of energy harvesters, further aggravating the need for innovative low power design and novel power management methods.

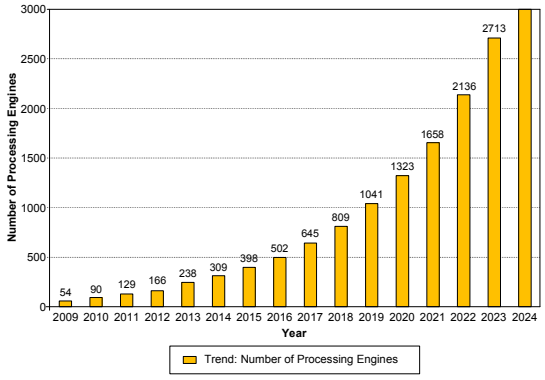

Figure 1.3: Expected consumer portable SoC design complexity trends (ITRS) [11]

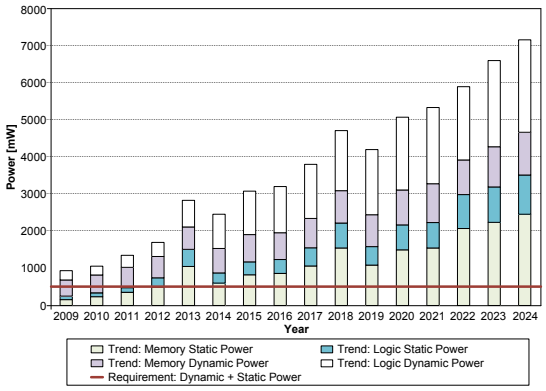

Figure 1.4: Expected consumer portable SoC power consumption trends (ITRS) [11]

For meeting these stringent low-power requirements advanced power management techniques will have to be deployed in SoC designs, such as such as fine-grained clock- and power gating, dynamic voltage and frequency scaling (DVFS) as well as special low-power and hibernation states of system components. Furthermore, leakage power management will be of increased interest with future nanoscale technology nodes [12]. The introduction of these advanced power management techniques will further increase the overall design complexity.

The increasing design complexity, however, renders simulation-based exploration and verification approaches more and more difficult. For an exemplary chip multi-processor (CMP) system consisting of 32 cores, executing a benchmarking application for one second, Figure 1.5 illustrates the simulation time effort [13]. Higher levels of abstraction, resulting

in less simulation accuracy, speed up the simulation but still fail to deliver results in reasonable time. Therefore, emulation-based approaches that allow for drastically reducing the timely effort represent a promising alternative to simulators.

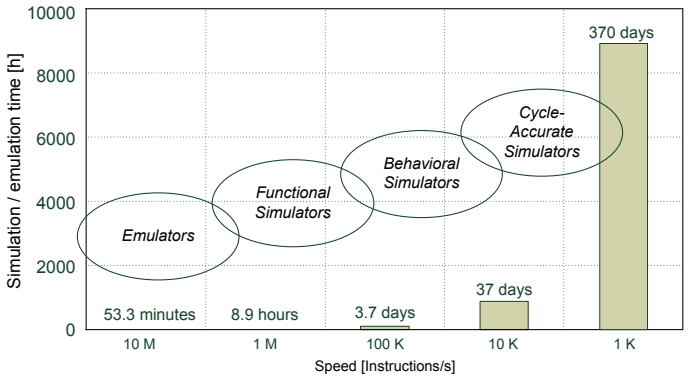

Figure 1.5: Time to simulate/emulate 1s of large-scale CMP workload execution ([13], with modifications)

1.1.3 Limitations of State-of-the-Art Power Emulation Methods

The advent of large but moderately priced field programmable gate arrays (FPGAs) has enabled the introduction of the *power emulation* technique, i.e., the emulation-based power estimation process. Initial low-level power emulation approaches at the register transfer level (RTL) [14, 15] achieve high estimation accuracies but suffer from extensive area overhead due to the additionally required power emulation hardware. The large overhead at this level of abstraction entails relatively low emulation speeds and a large set-up overhead for reducing the complexity of the used power models.

High-level power emulation approaches circumvent these large overheads by employing a power macromodeling approach. Implementations on the architectural level using hardware event counters have been shown [16]. However, these approaches require the polling and validation of event counter values through software and, hence, also impact the system-under-test. Furthermore, for novel designs a considerable manual power modeling and hardware implementation set-up overhead is required.

1.1.4 Missing Integration of Power Emulation in HW/SW Codesign

The power emulation approach allows for the rapid power profiling of a software application running on the system-under-test. While existing approaches focus on enabling the power emulation of a given system-under-test [14, 15, 16], the integration and utilization of the power emulation method in the hardware/software codesign process is little explored.

For enabling truly power-aware software development the integration of power emulation traces into an industry-grade software development environment, such as Keil μVision [17] or Eclipse [18], is required. By combining the high-level power emulation approach with a standard software development toolchain, the low-power design intent could also be conveyed to software engineers.

In a typical hardware/software codesign process designers are not only bound by power consumption but also by performance constraints. While power emulation approaches based on performance event counters exist [16], a joint power- and performance-emulation approach is yet missing. By recording both power consumption and performance event traces, hardware/software designers could explore power/performance trade-offs better.

1.1.5 Motivational Example: Low-Power ASIP HW/SW Codesign

An application-specific instruction-set processor (ASIP) for baseband processing according to the IEEE 802.15.4a ultra-wideband impulse radio standard amendment serves as a motivational example for the automated power emulation methodology. This baseband ASIP that is required for the purpose of synchronization and data reception between nodes, represents a critical part of future ultra-low-power wireless sensor nodes. Hence, an especially power-aware design process is required. The ASIP, as depicted in Figure 1.6, is implemented as a four issue slot VLIW[2] architecture that contains two scalar issue slots for executing scalar instructions and two vector issue slots capable of operating on 16 operands in parallel. The ASIP is presented in greater detail in the publication in Section 6.1.

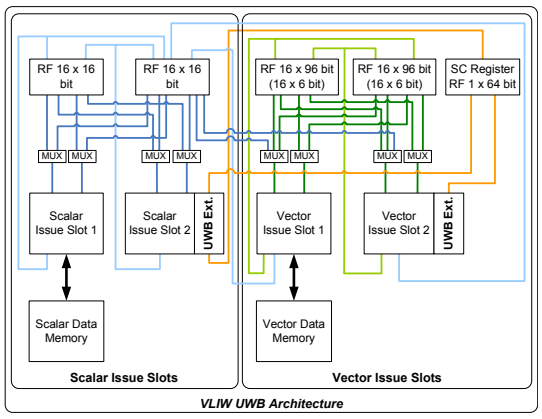

Figure 1.6: Architecture of the UWB ASIP [19]

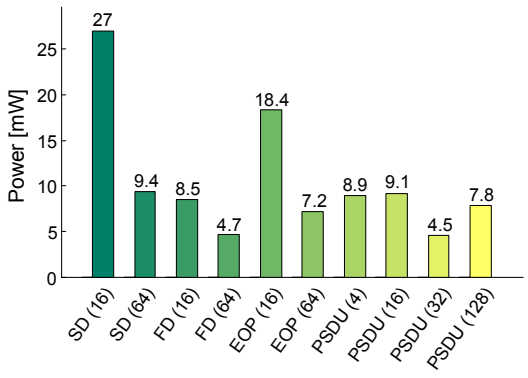

Figure 1.7: Different power states[3] of the UWB ASIP [19]

The execution of the baseband processing algorithms on this ASIP results in a number of different power states as depicted in Figure 1.7, according to the used mode of operation and the utilization of the individual issue slots. In a standard simulation-based HW/SW codesign approach the processor's power states as well as potential future power management features, required to meet certain constraints, are explored through time-intensive gate-level power simulations. The use of the power emulation technique would vastly decrease the timely effort and would allow for the execution of extensive benchmarks and real-world workloads, thus, enabling a more complete power analysis and verification.

[2] Very Long Instruction Word.

[3] Executed sub-algorithms: Signal detection (SD), fine acquisition (FD), end-of-preamble detection (EOP), payload decoding (PSDU).

1.2 Automated Power Emulation Methodology For Power-Aware HW/SW Codesign

1.2.1 The POWERHOUSE Project

The work at hand is part of the *"**Power**-Aware, **H**ardware-Supported **O**perating System and **U**biquitous Application **S**oftware Development **E**nvironment"* (POWERHOUSE) research project[4] that represents a collaboration between the Institute for Technical Informatics at the Graz University of Technology, Infineon Technologies Austria AG and AustriaCard GmbH. The POWERHOUSE methodology, as outlined in Figure 1.8, aims at employing the power emulation technique for increasing the power-awareness in the hardware/software codesign process.

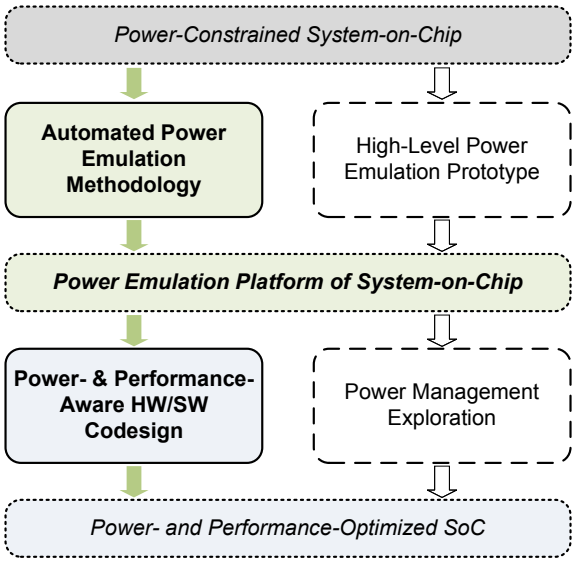

Figure 1.8: Overview of the POWERHOUSE methodology ([20], with modifications)

The main goals of the project are the creation of a power emulation platform for embedded systems. An initial high-level power emulation prototype serves as the proof-of-concept for the methodology. The automated power emulation methodology that derives a high-level power model and that performs the hardware implementation of these power models, facilitates the power emulation of novel system-on-chip designs. The run-time power consumption estimates, generated by the power emulation platform for the functionally emulated system, are used for increasing the power-awareness both in the hardware and in the software development process. By including hardware performance monitoring functionality, performance-awareness is added to the power-aware hardware/software-codesign process. Furthermore, the power emulation platform can be utilized in the rapid exploration process of novel power management techniques.

[4]The POWERHOUSE project was funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT contract FFG 815193.

1.2.2 Problem Statement

This thesis addresses the limitations and deficiencies in the power-aware hardware/software codesign process and previous power emulation approaches:

- Extensive timely effort for traditional simulation-based power estimation approaches
- Large hardware overhead for low-level power emulation approaches
- High power characterization and power modeling effort for novel designs
- Extensive hardware implementation effort for enabling power emulation
- Lack of hardware-accelerated power- and performance-aware HW/SW codesign approaches
- Lack of tight integration of power consumption feedback in software development and optimization environments

With respect to these limitations, this dissertation aims at introducing the power emulation method to the traditionally simulation-based hardware/software codesign process of systems-on-chip. By utilizing a high-level power macromodeling technique the extensive hardware overhead inherent to low-level power emulation is being vastly decreased. For enabling the use of the power emulation methodology in a productive hardware/software codesign environment, the effort for creating the power emulation platform of a novel system-under-test has to be considerably decreased. In this work, this reduction of effort is achieved by devising an automated power modeling and an automated power emulation hardware implementation method.

Furthermore, this dissertation addresses the lack of hardware-accelerated power- and performance-aware hardware/software codesign approaches. To this end, the utilization of this automatically generated high-level power emulation platform in the power-aware hardware/software codesign process is being explored. By integrating hardware performance monitoring functionality into the power emulation platform, not only power- but also performance-awareness can be considered in the HW/SW design optimization process. The missing integration of run-time power consumption feedback in software development is tackled by coupling a software IDE and an automated software optimization framework with the power emulation approach.

1.2.3 Contributions and Significance

The two main contributions, constituted by this dissertation, can be summarized as follows:

1. *Automated power emulation methodology:* The methodology addresses the automatic creation of a power emulation platform for use in a power-aware hardware/software codesign environment. An automated power modeling technique derives a high-level power model from training set data. To this end, a power model parameter selection process selects power-relevant system signals and derives fitting coefficients using linear regression. For the derived power model the power emulation unit is

automatically generated and its internal structure is adapted. The HDL model of the system-under-test is parsed, analyzed and modified in order to allow the monitoring of internal power-relevant signals that are used in the high-level power model that is implemented in the power emulation unit.

2. *Emulation-based power- and performance-aware HW/SW codesign:* The use of the power emulation methodology in the power-aware hardware/software codesign process is being explored. The combination of power emulation and hardware performance monitoring functionality permits joint power- and performance-aware system optimizations. By integrating the run-time power emulation traces into a standard software development environment, valuable power consumption feedback can be provided to software engineers and, hence, power-awareness can be increased. The run-time power traces can further be automatically analyzed to detect excessive power consumption peaks that threaten the stability of the system-under-test. An automated framework for use during the design phase is presented that utilizes run-time power emulation to detect software-induced power consumption peaks and adapts the software application to reduce these peaks. Furthermore, a run-time power profile flattening approach is outlined that uses the run-time power estimates to actuate dynamic frequency and voltage scaling power management functionality.

1.2.4 Structure of the Work

The remainder of this work is structured as follows. In Chapter 2 the state-of-the-art regarding measurement- and estimation-based power profiling methods is reviewed. In view of power estimation methods, different power simulators and previous hardware-accelerated power estimation approaches are outlined. Furthermore, the application of these power profiling methods in power-aware hardware/software codesign is reviewed. Chapter 3 outlines the automated power emulation methodology, consisting of the automated power modeling and the subsequent automated power emulation hardware implementation steps. In addition, the use of the resulting power emulation platform in the hardware/software codesign process is illustrated with regard to power- and performance-aware hardware as well as software optimizations. Evaluation results for the presented methodology are given in Chapter 4. These include case studies performed on two prototypical SoC designs, operating in power- and energy-constrained environments. Chapter 5 concludes the dissertation with a summary of findings and provides an outlook on future work. A number of selected publications that are further extending the presented methodology and the evaluation thereof are included in Chapter 6.

Chapter 2

Related Work

For enabling the *power-aware HW/SW codesign and optimization process* of novel system-on-chip designs the *power consumption profiling* of these systems is required. For this profiling process different methods are available, as illustrated in Figure 2.1. At late design stages, typically when first tape-out silicon is available, *power measurements* can be utilized while in early design phases *estimation-based* methods have to be employed. Power profiling methods that have been used in the early-stage power-aware hardware/software development process include *software simulators*, *hardware-accelerated approaches* utilizing either emulation platforms, dedicated hardware or already existing hardware implementations and *hybrid approaches*.

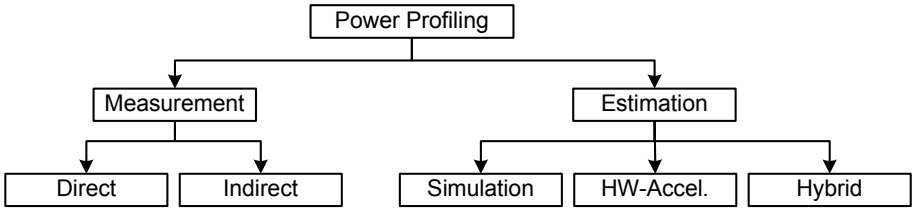

Figure 2.1: Overview of HW/SW power profiling methods

For the purpose of *power and performance profiling* in the HW/SW codesign process of complex systems, full-system simulation frameworks as well as emulation-based profiling approaches have been presented. Furthermore, the simulated power consumption profiles have been used for the detection and optimization of *power consumption peaks* caused by the execution of software applications.

2.1 Power Consumption Profiling Methods

2.1.1 Measurement-Based Power Profiling Methods

At more mature stages of the design process, when a physical implementation of the design-under-test, i.e., tape-out silicon, is available, power measurements represent a fast and accurate way to determine the system's power consumption.

Traditional *direct power measurement approaches* rely on ammeters, oscilloscopes and other direct-measurement data acquisition systems. In early works measurement-based

power profiling has been employed for investigating the power consumption caused by the execution of different software applications. Tiwari et al. have used a standard ammeter-based measurement setup to collect training set data for the characterization of instruction level power models [21, 22]. Flinn et al. have employed a similar setup for profiling the energy usage of software applications running on general purpose computer systems [23]. By combining the measured power consumption with system activity data collected by an operating system daemon process, their approach is able to map the system's energy consumption to program structure. Regarding commercial solutions, Texas Instruments has presented an oscilloscope-based approach measuring the currents drawn by a DSP system while executing given software applications [24]. The power measurement data are displayed within the provided software development environment. Hitex has provided another commercial measurement-based solution, allowing for the concurrent measurement of up to four loads [25]. By integrating the measurement data within the Keil software IDE [17], the correlation between power consumption and source code can be analyzed.

Indirect power measurement by thermal imaging has been presented as an interesting alternative to traditional power measurement approaches. Hamman et al. have presented an experimental technique that derives a chip's spatial power distribution from infrared (IR) thermal imaging [26]. They have illustrated the applicability of their technique for measuring the temperature and, hence, the power distribution as a function of the given workload. Mesa-Martinez et al. similarly have used an IR measurement setup for determining the thermal characteristics and the power consumption of a given chip [27]. By applying genetic algorithms they derive a detailed per-component power consumption breakdown that can subsequently be used to evaluate and refine existing power simulators.

While measurements provide a fast and accurate way of profiling a system's power consumption, they are only applicable late in the HW/SW codesign process. Furthermore, traditional power measurement approaches typically fail to deliver detailed per-component power consumption data due to chip integration and packaging that limit the measurement instrumentation. Indirect measurement approaches utilizing thermal imaging elude this limitation as no direct measurement probe contact is required. However, they entail large experimental setup overheads and require extensive preparation steps such as the removal of heatsinks, the thinning of the chip to improve the observability of the heating effects, thermal calibration, etc.

2.1.2 Estimation-Based Power Profiling Methods

At early design stages, when parts of the design are still incomplete, power estimation methods have to be utilized that build on assumptions regarding the detailed structure and layout of the later physical implementation of the system and its sub-components [28]. For accelerating the power estimation process, hardware-accelerated power estimation approaches have been derived in recent years. Figure 2.2 compares generic methods for the either simulation-based or hardware-accelerated power estimation process. While simulation-based approaches garner the activity data required for estimating the system's power consumption through the evaluation of models, hardware-accelerated approaches derive activity data by monitoring either the system's functional emulation or its physical implementation. Based on these activity data power consumption estimates are derived. Emulation-based approaches exist that further speed up the power estimation by imple-

menting both the activity data acquisition and the power estimation process in hardware.

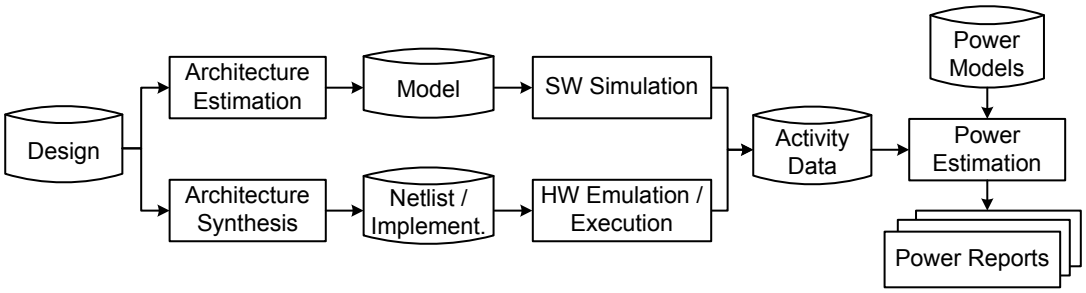

Figure 2.2: Generic simulation vs. generic hardware-accelerated power estimation methods

Simulation-Based Methods

A multitude of simulation-based methods, operating on different levels of abstraction, has been proposed. At low levels of abstraction such as the *gate level* and the *register transfer level* numerous state-of-the-art power estimation tools, e.g., Synopsys PrimeTime [29] and Magma Blastfusion [30], are available. For increasing the simulation speed, higher levels of abstraction have been researched.

Instruction level models, consisting of base costs per instruction as well as overhead costs for switching between different instructions, have been defined [21, 22]. These instruction-level models have been further refined by considering microarchitectural effects in, e.g., pipeline-aware models, resulting in improved estimation accuracies [31].

At the *system level* SimplePower, introduced by Ye et al. [32], and Wattch, presented by Brooks et al. [33], constitute processor power simulation tools based on the SimpleScalar tool set [34, 35]. SimplePower simulates the integer subset of the SimpleScalar instruction set and considers the processor's datapath, memory and on-chip buses but does not include the processor's control unit, as well as the clock generation and distribution network in the power simulation process [32]. Wattch simulates a processor containing a five-stage pipeline and performs the power estimation for numerous components such as caches, register files, functional units as well as the clocking network [33].

By raising the level of abstraction used in simulation-based approaches, higher simulation speeds can be achieved while reducing the simulation accuracy. For the simulation of large workloads on increasingly complex system-on-chip designs, however, simulation-based approaches still fail to deliver results in reasonable time [13]. Therefore, various hardware-accelerated methods have been researched and represent a promising alternative to simulation-based approaches.

Hardware-Accelerated Methods

With regard to *hardware-accelerated power estimation* methods, techniques leveraging existing *hardware event counters* [36, 37, 38], dedicated *estimation coprocessors* [39, 40, 41, 42] and *emulation-based approaches* [14, 15, 16, 43, 44] have been presented.

Initial works have been utilizing existing *hardware event counters* for the purpose of power estimation by correlating the occurrence of specific hardware events with the

system's power consumption. The feasibility of this approach has been shown for commercially available desktop processors by Bellosa et al. [36] and Joseph et al. [37]. The same approach has also been applied to embedded processors by Contreras et al. [38].

Especially for enabling dynamic power management in mobile embedded systems, *dedicated energy- and power-estimation coprocessors* have been introduced. Haid et al. have introduced the JouleDoc coprocessor that performs run-time energy accounting based on energy macromodels by counting the occurrence of energy-relevant system events [39, 40]. The CLIPPER methodology that has been presented by Peddersen et al. represents a similar approach which implements run-time power estimation hardware counters alongside the system-under-test, hence, resembling the power estimation coprocessor paradigm [41, 42].

With the advent of FPGAs that are large enough to hold entire system-on-chip designs, the *emulation-based power estimation* technique, referred to as *"power emulation"*, has become feasible. For enabling the power emulation of a given system-under-test, the system's HDL model is augmented with dedicated power estimation hardware that performs the power estimation process as a by-product of the functional emulation process. In recent work, power emulation approaches operating on different levels of abstraction have been presented. Initial power emulation approaches as presented by Coburn et al. employ RTL macromodels and achieve high estimation accuracies (mean error 3.4% as compared to gate level simulations) [14, 15]. By combining the power emulation with the cosimulation of non-synthesizable parts of the system, a hybrid simulation- and emulation-based power estimation technique has been introduced by Ghodrat et al. [44]. Low-level power emulation approaches suffer, however, from high area overhead (on average 3.1 times the area of the original design) [14]. This overhead decreases the maximum reachable operating frequency of the emulated design and entails a large set-up overhead for reducing the complexity of the used power models.

Therefore, a system level power emulation approach using event-based power accounting has been introduced by Bhattacharjee et al. [16]. Through the use of a high-level power model less than 3% of the available FPGA LUTs are required (the dual-core Gaisler LEON3 test system [45] is reported to require 60% in the used configuration) while the estimation error in comparison to gate-level simulations is reported to be below 10% [16]. However, the approach relies on manually inserted hardware performance event counters, entailing the required effort for the manual selection of power-relevant events, for the power modeling itself and for the hardware implementation of the used event counters. Furthermore, the evaluation of the event counter values, with regard to the used power model, is performed in software and slightly modifies the execution behavior of the system-under-test.

2.2 Power and Performance Profiling for HW/SW Codesign

In the power and energy optimization phase of the HW/SW codesign process, designers are typically also constrained by performance requirements for the given design. For the purpose of power- and performance-aware HW/SW design space exploration of complex systems, a number of full-system simulation- as well as emulation-based profiling approaches have been presented in literature.

Initial *full-system simulators* include the SimpleScalar [34, 35] simulator upon which the power simulators SimplePower [32] and Wattch [33] build. Magnusson et al. have presented the Simics full-system simulator at the instruction-set level that is offering simulation models for various microprocessor architectures [46]. The SystemC-based MPARM simulation platform by Benini et al. allows for the simulation of entire multi-processor systems-on-chip (MPSoC), including buses and memories [47]. Furthermore it contains a cycle-accurate ARM processor instruction set simulator (ISS) [47]. Cong et al. have introduced the MC-Sim simulation framework for heterogeneous multi-core systems that is consisting of a functional ISS for the processor cores, cycle-accurate structural models for the interconnect and behavioral models for coprocessors [48]. A transaction level simulation environment for MPSoCs with focus on software timing and performance estimation has been presented by Gerin et al. [49]. By annotating a given embedded software application the execution behavior of a specific target processor is simulated.

Simulation-based approaches offer the benefits of early design phase applicability and easy instrumentation of various system components for extracting power and performance statistics. However, with increasing hardware complexity and increasing number of collected statistics, the simulation speed drastically decreases. By introducing higher levels of abstraction this effect can be partially circumvented, nevertheless, the risk of concealing low-level effects and potential design errors increases. For overcoming the limitations of simulators, emulation-based power and performance profiling approaches have been introduced.

Due to the large area overheads of the initial register transfer level power emulation approach presented by Coburn et al. [14], its application in full-system profiling is limited. A full-system MPSoC emulation framework has been proposed by Del Valle et al. [50]. The framework has been extended by Atienza et al. with manually inserted sniffers for monitoring event statistics of different system components. The statistical data allow for the derivation of power and thermal estimates based on known power figures of already existing cores [43, 51]. Bhattacharjee et al. have introduced a similar high-level event-counter-based power emulation approach for a proposed chip multi-processor LEON3 design. By evaluating a set of manually inserted hardware performance event counters, a power model implemented in software (i.e., in the Linux timer kernel interrupt routine) derives power estimates every 10 ms (or multiples thereof) [16]. Due to the required power model evaluation in software that slightly modifies the system's execution behavior and consumes execution time, this approach cannot be regarded as truly non-invasive. Another emulation-based framework for the design space exploration of embedded multi-core systems has been presented by Meloni et al. [52]. The framework estimates technology-dependent power consumption and area requirements of proposed ASIC implementations based on the extraction of statistics collected by performance event counters [52]. In a case study different multi-processor NoC architectures are being explored using this approach, however, the accuracy of the power and performance estimates is not being quantified.

2.3 Power Peak Optimization of Embedded Software

The execution of software applications has an impact both on a system's average as well as its peak power consumption. Software-induced power consumption peaks can lead to

supply voltage drops that particularly threaten the reliable and stable operation of battery- and electromagnetic-field-powered devices [53]. Therefore, the detection and reduction of power consumption peaks is of increased interest. Furthermore, it has been shown that a system's power consumption leaks essential information about the execution flow of a given software application and the data processed within this application. This information can be used by attacks such as the differential power analysis [54] to extract secret cryptographic keys. Hence, the reduction of power profile variability is of increased interest for security applications as a countermeasure against these attacks.

For the purpose of reliability enhancements for smart card systems, a number of power peak reduction methods have been researched. Grumer et al. have presented a framework that profiles the power consumption of given software applications using an instruction set simulator coupled with a power model for a MIPS32 4KSc processor [55]. They employ instruction-level power models as initially introduced by Tiwari et al. [56] as well as additional power models considering the bus system, the memories and the peripherals. A subsequent power peak elimination stage aims at reordering operations at the instruction-level and at the intermediate-language-level by adapting the instruction schedule pass of the compiler [55]. By reordering the instructions, the algorithm tries to minimize the power consumption by reducing the switching activity due to circuit state changes. In [57] the same authors have presented a different approach for power peak optimization. By utilizing an iterative compilation process that is using different combinations of compiler optimization passes on the intermediate-language-level of the compiler the authors try to reduce the existing peaks. Using the same power profiling framework Wendt et al. have explored the insertion of non-functional instructions (NFI) that do not modify the functionality but exhibit a lower power consumption (due to less switching activity during their execution) as a means for reducing short-term average power consumption and, hence, for reducing power peaks [58].

For software applications in the security domain, a number of power profile flattening techniques aiming at the reduction of the profile's variability, have been investigated in order to avoid information leakage. Muresan et al. have presented a source code transformation technique and a hardware architecture that are both employing NFI insertion for reducing power profile variability [59]. Power profile flattening by means of an integrated real-time current-injection module has been proposed by Li et al. [60]. For reducing the energy consumption overhead inherent to this flattening approach, they have proposed to only enable the current-injection module during security-critical execution periods. This approach has been extended by Vahedi et al. to include dynamic voltage scaling capabilities [61]. The combined current-injection and dynamic voltage scaling approach, allows for improved flattening performance while limiting the energy consumption overhead.

While simulation-based approaches allow for the power peak detection and reduction at early design phases, the risk of omitting power peaks caused by sources not considered in the simulation model exist. Furthermore, the profiling of entire real-world applications scenarios requires extensive simulation times. Hardware-based approaches are able to reduce the power profile's variability at a much finer level of granularity, allowing for the effective flattening. The additionally required in-system measurement hardware as well as the presented flattening approach utilizing current-injection, however, lead to a higher energy consumption.

2.4 Summary

Power consumption profiling methods represent a necessity in the power-aware hardware/software codesign process and have been extensively researched in related work. Especially at early design phases, when measurement-based approaches are not applicable, estimation-based power profiling approaches have to be employed. Numerous power simulators, operating on different levels of abstraction, have been presented in literature. By raising the level of abstraction, higher simulation speeds are achievable at the cost of decreased estimation accuracy. However, the simulation of long software test cases on increasingly complex hardware designs still results in extensive simulation times.

Hardware-accelerated power estimation approaches speed-up the profiling process by deriving activity data from the system's hardware implementation. For deriving power estimates these activity data are then evaluated by a power model implemented either in hardware or in software. More recent emulation-based approaches have focused on using the hardware-accelerated power estimation approach during the system's design phase. While initial low-level power emulation methods suffer from large hardware overheads that inhibit their use for the power profiling of complex system designs, high-level power emulation approaches entail only small overheads and permit high emulation speeds. Existing approaches, however, have not addressed the issue of automatically enabling the high-level power emulation of a given system-under-test, including the required power modeling and hardware implementation steps.

Regarding the power-aware development process that is also constrained by performance requirements, the utilization of the high-level power emulation technique for HW/SW analysis and optimization is little explored. Full-system simulation frameworks, operating at higher levels of abstraction to accelerate the simulation, bare the risk of concealing low-level effects and design errors. Existing emulation-based frameworks entail large setup overheads and cannot be regarded as truly non-invasive due to the required high-level power model evaluation in software. In terms of power-aware software development, the use of the power emulation technique is little explored. Existing simulation-based approaches for detecting and optimizing software-induced power consumption peaks, exhibit slow simulation speeds and contain the risk of omitting peaks caused by sources not considered in the simulation model.

With regard to related work, the objectives of this dissertation can be summarized:

- Accelerating the required power profiling in the HW/SW codesign and optimization process by utilizing an emulation-based approach
- Decreasing the power emulation hardware overhead of initial low-level power emulation approaches by using a high-level power macromodel
- Enabling the automated power modeling of novel systems-under-test to decrease the power emulation setup overhead
- Automated power emulation hardware implementation to allow for the fast deployment of the system-under-test's power emulation platform in the design process
- Exploration of a joint power emulation and performance monitoring approach for use in the HW/SW codesign and optimization process
- Exploration of manual as well as automated power-aware software development and software-induced power peak optimization utilizing emulation-based power profiling

Chapter 3

Novel Automated Power Emulation Methodology For Power-Aware HW/SW Codesign

3.1 Overview

The high-level power emulation technique represents a promising alternative to simulation-based power profiling approaches. The proposed automated power emulation methodology aims at overcoming the limitations of previous power emulation approaches to facilitate its use in the hardware/software codesign process for novel system-on-chip designs. Figure 3.1 provides an overview of the individual contributions that represent the automated power emulation methodology and illustrate its use in the power- and performance-aware codesign process. These contributions are presented in more detail in the publications included in Chapter 6.

The low-power SoC design introduced in Section 6.1 serves as a motivational example and illustrates the need for an accelerated power estimation approach for use in the power-aware hardware/software codesign process. The publications in Sections 6.2 and 6.3 introduce the *high-level power modeling* approach for enabling the power emulation of a given SoC design. The initial prototype of the platform that has served as a proof-of-concept for the high-level power emulation approach is presented. Furthermore, the novel methodology, consisting of *automated power modeling* and the *automated power emulation hardware implementation*, is outlined in Section 6.3. The application of the resulting high-level power emulation platform in the power- and performance-aware codesign process is illustrated in the publications in Sections 6.4-6.8. By augmenting the power emulation approach with hardware performance monitoring hardware, power/energy optimizations can be performed while observing their performance impact (Section 6.4). The integration of the run-time power traces into a standard software development environment, enables truly power-aware software optimizations (Sections 6.5, 6.6). Furthermore, the run-time power estimates can be used for the detection and reduction of power peaks that threaten the system's stability. A software-based power peak reduction framework is presented in the publication in Section 6.7, whereas a hardware-based approach is introduced in Section 6.8.

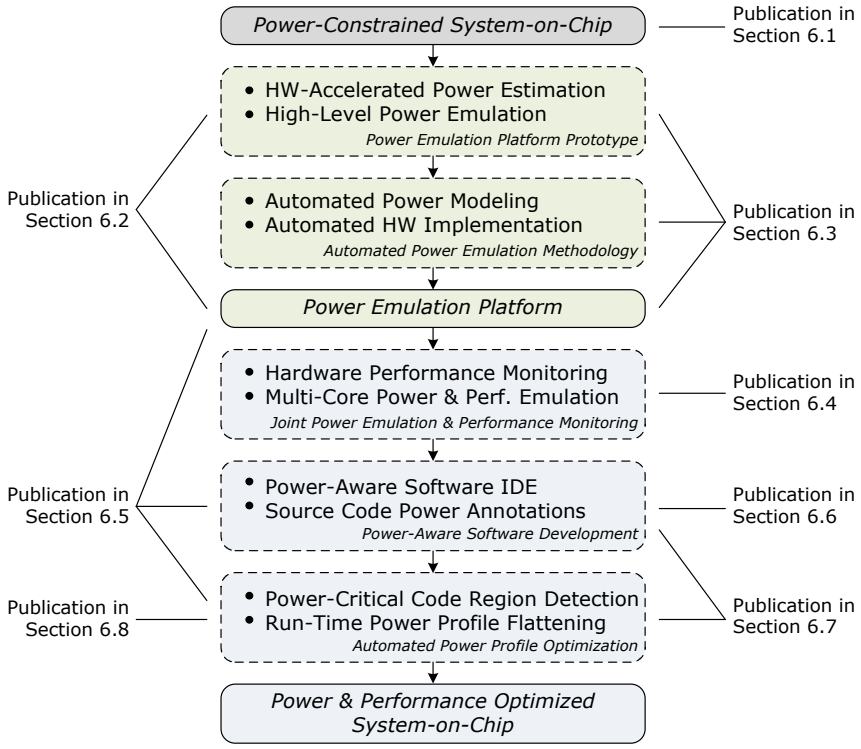

Figure 3.1: Overview of automated power emulation methodology for power-aware hardware/software codesign

3.2 Automated Power Emulation Methodology

3.2.1 High-Level Power Emulation Technique

The high-level power emulation method represents a key component of the POWER-HOUSE project. In contrast to initial RTL power emulation approaches as outlined in Section 2.1.2, the high-level approach offers the benefit of decreased hardware overhead and decreased emulation impact, resulting in higher emulation speeds and reduced emulation platform costs.

For a CMOS-based system the power consumption can be given as $P = P_{dyn} + P_{sta}$, consisting of dynamic and static power consumption. The dynamic power consumption $P_{dyn} = \alpha(fCV^2)$ is dependent on the component activity factor α, the clock frequency f, the capacitance C being switched and the supply voltage V. P_{sta} represents sources of static power consumption including leakage power. In the high-level power emulation method presented in this thesis the total power consumption P of a given system is approximated using a power macromodel that derives a cycle-accurate power consumption estimate $\hat{P}[t]$ by observing a number N of power-relevant signals $x_i[t]$ of the system. The model is implemented as an additive linear equation of model parameters $x_i[t]$ and according coefficients c_i as expressed in Equation 3.1. In this macromodel, the coefficient c_0 accounts for sources of static power consumption whereas the coefficients c_i $(i = 1 \ldots N)$

represent the dynamic power consumption.

$$\hat{P}[t] = c_0 + \sum_{i=1}^{N} c_i x_i[t] = c_0 + c_1 x_1[t] + \ldots + c_N x_N[t] \tag{3.1}$$

The power emulation unit, as depicted in Figure 3.2, represents the hardware implementation of the power macromodel. It monitors the power-relevant signals $x_i[t]$ and derives the cycle-accurate power estimate $\hat{P}[t]$ for the system and its sub-components. Furthermore, post-processing and host data transfer functionality is provided by the power emulation unit.

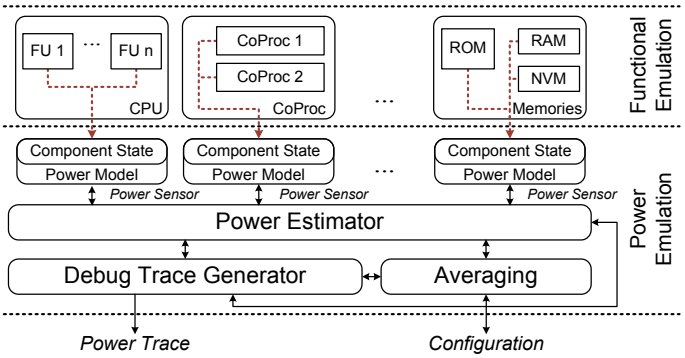

Figure 3.2: Power emulation unit [62]

The high-level power emulation methodology and its hardware implementation are described in more detail in *"An Emulation-Based Real-Time Power Profiling Unit for Embedded Software"* (Section 6.2) and *"Automated Power Characterization for Run-Time Power Emulation of SoC Designs"* (Section 6.3).

3.2.2 Automated Power Modeling for Power Emulation

For enabling the power emulation of a novel system-under-test a power model suitable for the implementation in the power emulation unit has to be created first. To this end an automated power modeling methodology as depicted in Figure 3.3 is employed that derives a power model from activity and power consumption data.

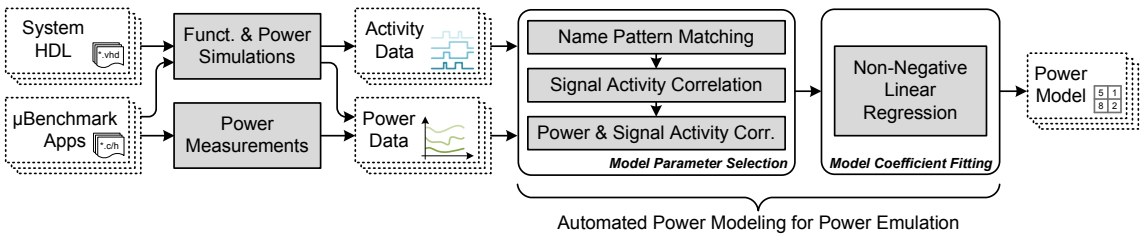

Figure 3.3: Simulation- and measurement-based automated power modeling methodology for power emulation

These data are typically derived from functional and power simulations using state-of-the-art RTL or gate-level power simulation tools such as Synopsys PrimeTime [29] and

Magma Blastfusion [30]. At later design stages, i.e., after first tape-out silicon is available, power measurements can be conducted and the resulting measurement data can also be used in the power modeling process. The automated power modeling methodology consists of two main stages: 1) The model parameter selection stage consists of the signal name pattern matching, the intra-signal correlation and the power - signal activity correlation filtering steps. It aims at identifying a number of N model parameters so that $N \ll N_{all}$, i.e., the number of selected parameters is smaller than the total number of available parameters. 2) The model coefficient fitting process employs a least squares regression technique with non-negativity constraint to derive coefficients for the parameters selected before.

The automated power modeling method for power emulation is described in greater detail in *"Automated Power Characterization for Run-Time Power Emulation of SoC Designs"* (Section 6.3).

3.2.3 Automated Power Emulation Hardware Implementation

After deriving a power model another obstacle for enabling the power emulation of a novel system-under-test is constituted by the time-consuming and tedious task of implementing the automatically devised model in hardware and integrating it into the existing HDL model of the system-under-test. This obstacle is addressed by introducing an automated power emulation hardware implementation method as depicted in Figure 3.4.

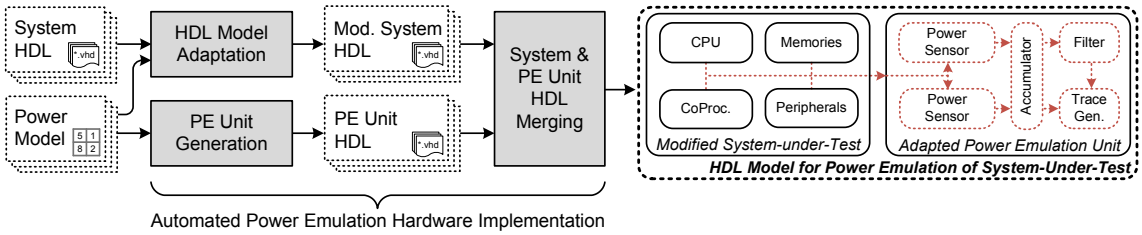

Figure 3.4: Automated power emulation hardware implementation

The approach consists of two main stages: 1) The automated HDL adaptation of the given system-under-test to allow for the monitoring of internal signals that are used within the system's previously derived power model. This stage comprises the parsing of the system's HDL model and its modification by routing power-relevant signals originating in different design entities at different levels of hierarchy to the instantiation of the power emulation unit. 2) The power emulation unit HDL generation for implementing the devised power model. Based on HDL template files of the power emulation unit an adaptation algorithm modifies the number and the structure of used power sensors as well as the structure of the power value accumulator and the filtering unit. Afterwards the power emulation platform for the given system-under-test can be created from the adapted HDL model using a state-of-the-art FPGA hardware implementation tool flow that performs the typical hardware synthesis, mapping, place and route, as well as the bitstream generation.

The publication *"Automated Power Characterization for Run-Time Power Emulation of SoC Designs"* (Section 6.3) further details the automated power emulation hardware implementation approach.

3.3 Power-Aware HW/SW Codesign Based on Power Emulation

3.3.1 Joint Power Emulation and Performance Monitoring for Multi-Core Systems

While minimizing the power and energy consumption of a system in the hardware/software codesign process, designers are typically also bound by performance constraints that are dictated by the given application. In an emulation-based hardware/software codesign process both the profiling of the power consumption as well as the analysis of performance indicators of various system components is therefore of increased interest.

For this purpose a joint power emulation and performance monitoring approach has been introduced by additionally enabling the monitoring of performance indicators of the given system-under-test. This performance monitoring and logging allows designers to develop a deeper understanding of trade-offs between the power/energy consumption and the performance of the system. Furthermore, this approach allows for identifying both the source as well as the causing mechanisms of excessively high power consumption or poor execution performance.

In addition, the previously introduced high-level power emulation methodology has been extended to enable the joint power and performance emulation approach in multi-core environments. This allows for individually estimating the power consumption and monitoring the performance of heterogeneous system components.

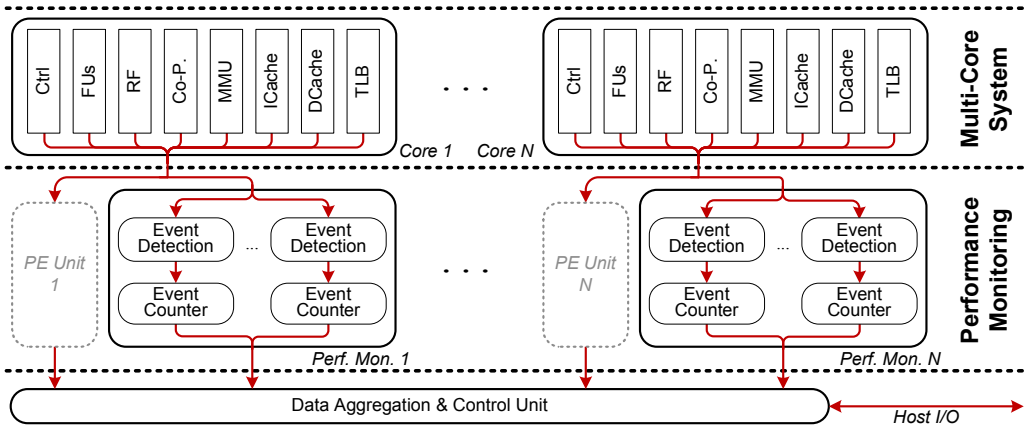

Figure 3.5: Joint power emulation and performance monitoring method

In *"An Emulation-Based Platform for Power- and Performance-Aware HW/SW Development of Embedded Multi-Core Systems"* (Section 6.4) an extended discussion of the joint power emulation and performance monitoring approach for multi-core systems is provided. Furthermore, evaluation results for an embedded multi-core test system are presented.

3.3.2 Power-Aware Software Development Using Power Emulation

A main goal of the POWERHOUSE project is the refinement of power-aware software development trough the use of the power emulation methodology. By providing the run-time power-feedback derived from the power emulation methodology, the low-power design intent can be realized not only by hardware but also by software engineers.

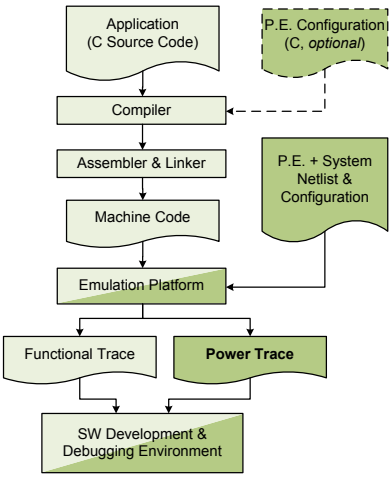

Figure 3.6: Use of power emulation platform in a standard software development tool flow [63]

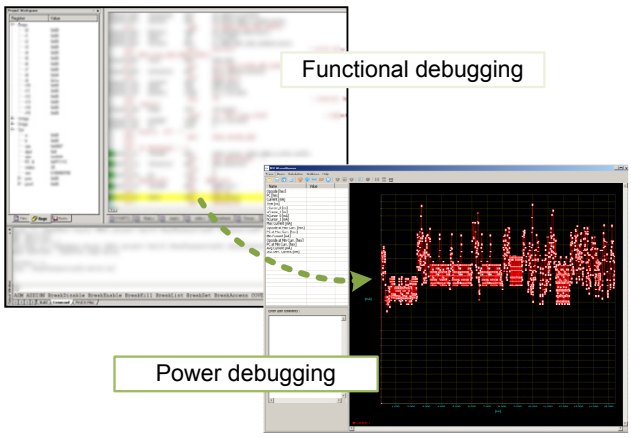

Figure 3.7: Integration of power emulation data into a standard software development environment

For the purpose of enabling power-aware software development, a standard software development tool flow has been extended with the power emulation technique as depicted in Figure 3.6. A standard toolchain consisting of compiler, assembler and linker translates the application source code. The compiled application is then loaded onto and executed on the power emulation platform in the same way as performed for purely functional emulation. During the execution of the application on the power emulation platform the power trace is recorded alongside the functional execution trace. These data are evaluated within an integrated software development and debugging environment (IDE) that is used for visualizing and analyzing the recorded traces as illustrated in Figure 3.7. The IDE performs the execution trace to source code correlation, allowing for the assignment of power consumption estimates to individual source code lines.

The use of the power emulation methodology for enabling power-aware software development is covered by *"Accelerating Embedded Software Power Profiling Using Run-Time Power Emulation"* (Section 6.6).

3.3.3 Emulation-Based Power Peak Optimization of Embedded Software

Especially in power-constrained mobile systems, power consumption peaks can lead to supply voltage drops below a critical limit and can, therefore, threaten a system's stability and reliability. The detection and reduction of power consumption peaks, caused, e.g., by the concurrent activation of power-intensive peripherals during the execution of a given software application, is of increased interest in power-constrained systems development.

Design Phase Power Peak Optimization

For analyzing and optimizing power consumption peaks caused by software applications during the design phase, a complete framework consisting of multiple stages as depicted in Figure 3.8 has been introduced. First, the embedded software application is processed using a standard software development tool chain (A). The resulting machine code is then automatically profiled on the power emulation platform (B). The resulting power profiles are then analyzed and evaluated in terms of their impact on the supply voltage circuitry of the given system-under-test (C). Based on these data, power-critical software source code regions that cause power peaks, leading to critical supply voltage drops, are detected. Depending on the power management features available on the given system-under-test, an optimization algorithm chooses a power management approach that is afterwards automatically applied to the source code (D).

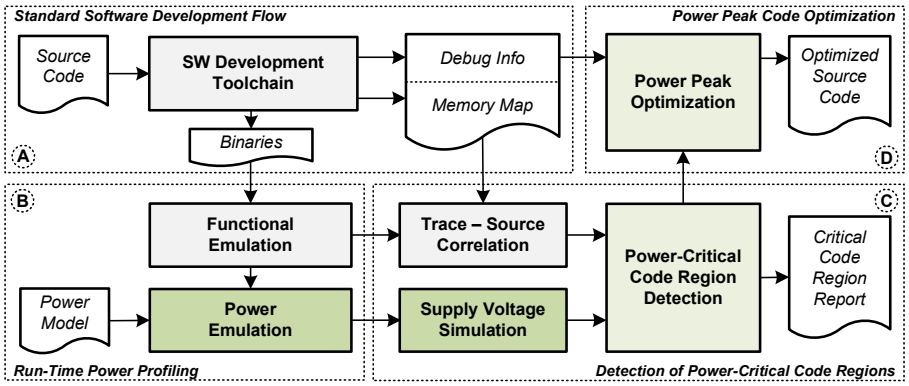

Figure 3.8: Framework for emulation-based power peak optimization of embedded software [64]

Run-Time Power Profile Flattening

Furthermore, a run-time power profile flattening approach has been presented that is based on the hardware power estimation architecture used in the high-level power emulation approach. The run-time estimates are used to actuate system-level dynamic frequency and voltage scaling (DVFS). The system's power consumption is, hence, controlled in a purely digital manner, rendering costly analog on-chip-power measurement hardware unnecessary.

The emulation-based peak-power optimization method for embedded software is described in more detail in *"An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software"* (Section 6.7), whereas the run-time flattening approach is outlined in *"Estimation-Based Run-Time Power Profile Flattening for RF-Powered Smart Card Systems"* (Section 6.8).

Chapter 4

Evaluation of Methodology and Case Studies

4.1 Overview

For evaluating the automated power emulation methodology and its use in the power-aware HW/SW codesign process, high-level power emulation platforms for two prototypical SoC designs have been created. This chapter presents the evaluation results for these test systems. By using the automated power modeling and hardware implementation methods the high-level power emulation platforms have been devised. For these platforms the power estimation accuracy, the additionally required hardware overhead and the power emulation speed-up in comparison to software simulators are being quantified in this chapter. The applicability of the joint power emulation and performance monitoring to multi-core architectures is being illustrated. Furthermore, the use of the high-level power emulation in power-aware software development and the automated reduction of stability-critical power consumption peaks is being presented.

4.2 Power Emulation Test Systems

For the evaluation of the novel automated power emulation methodology a 16-bit Infineon SLE78 smart card microcontroller system-on-chip [65] and a 32-bit Gaisler LEON3 multi-processor system-on-chip (MPSoC) [45] have been chosen. These two test systems represent SoC designs that are operating in very power- and energy-constrained operating environments that make a power-aware HW/SW codesign process particularly necessary. The architectural parameters of the two test systems are given in Table 4.1 and Table 4.2 respectively.

Test System 1: Heterogeneous Smart Card System-on-Chip
A *16-bit smart card microcontroller SoC* [65], supplied by an industrial partner, represents the first test system. It contains volatile and non-volatile memories, cryptographic coprocessors as well as other peripherals such as random number generators and timers. The system is also intended to operate in contactless smart cards that are harvesting energy from the electromagnetic field generated by a reader device. For this reason the system is

Architecture	16-bit Infineon SLE 78
Number of Cores	2[a]
Clock Rate	1-33 MHz
Technology	0.13 μm[b]
CoProc./HW Acc.	AES/DES, RSA Crypto
Interfaces	UART, RF
Peripherals	Timers, (T)RNG
Memories	ROM, RAM, NVM

Table 4.1: Architectural parameters for the smart card microcontroller test system [65]

[a]Redundant cores for increased security.
[b]Technology node for power modeling.

Architecture	32-bit SPARC V8
Number of Cores	1-16
Clock Rate	35 MHz[a]
Technology	90 nm[b]
CoProc./HW Acc.	HW MAC/MUL/DIV
Interfaces	CAN, Ethernet, UART
Peripherals	PS/2, Timers, VGA
Memories	ROM, RAM

Table 4.2: Architectural parameters for the multi-processor test system [45]

[a]Depending on FPGA implementation.
[b]Technology node for power modeling.

based on an especially power-optimized design, containing dedicated power-aware system states (e.g., low-power halt and sleep modes of certain components). Due to these low-power states and the especially power-aware design of the system, it represents an ideal test system for the evaluation of the automated power emulation methodology.

Test System 2: Multi-Processor System-on-Chip
An Aeroflex Gaisler LEON3 *32-bit SPARC V8-compliant multi-processor SoC* [45] constitutes the second test system. The LEON3 processor contains a SPARC V8 integer unit with a 7-stage pipeline, incorporates hardware multiply, divide as well as MAC units and a number of other peripherals. A LEON3-based multi-processor SoC can be implemented with up to 16 cores, enabling the execution of symmetric multi-processing (SMP)-enabled operating systems (e.g., Snapgear Linux [66]). Furthermore, the system is also optimized for low power consumption, allowing for its use in wireless and mobile applications. Due to the higher hardware complexity and the multi-processor support the LEON3 constitutes an ideal second test system for the automated power emulation methodology.

4.3 Automated Power Emulation Methodology

4.3.1 Automated Power Modeling

High-level power macromodels, as introduced in Section 3.2.1, of both test systems have been created by utilizing the automated power modeling approach (see Section 3.2.2). For evaluating the effectiveness of the automated power modeling approach, different power modeling techniques have been benchmarked for the smart card microcontroller test system: (1) *Manually* selected power model parameters. Power-relevant signals of the system-under-test have been selected in a manual way to allow for the first implementation of the power emulation platform. These signals have been used as candidate power model parameters. (2) A *brute force approach* by applying the linear regression algorithm to the large set of potential power model parameters chosen only by their corresponding signal names. All parameters assigned significant coefficient values have been retained and used as power model parameters. (3) Power model parameters retained after *intra-signal activity correlation filtering*. Signals of which the activity data correlate with other signals for the used set of benchmarks have been discarded. The remaining signals have been used as

power model parameters. (4) Power model parameters retained after *intra-signal activity correlation and power correlation filtering*, which represents the finally used parameter selection technique in this work. Only signals of which the activity data correlate with the training set power consumption data have been used as power model parameters.

Table 4.3 compares the timely effort for these different power modeling approaches. The automated power modeling approach clearly outperforms the manual power modeling as well as the naive brute force approach. In terms of power modeling accuracy, Figure 4.1 summarizes the estimation errors for the different power modeling techniques. To this end, a number of benchmarking applications have been divided into a set of characterization microbenchmarks, i.e., the training set (TS), and a set of evaluation benchmarks, i.e., the control set (CS). The automated power modeling approach outperforms the manual modeling and the brute force approach both in terms of average estimation error and root-mean-square estimation (RMS) error that represents the cycle-by-cycle estimation accuracy. Figure 4.1 illustrates the reduction of the average error from 11.78% to 4.71% and for the RMS error from 42.24% to 24.04% for the automatically generated power model in comparison to the manually created one. The publication in Section 6.3 further elaborates the evaluation of the automated power emulation technique.

Parameter Selection Method	Effort / Execution Time
Manual selection	several days
Brute force[a]	$\sim$ 12 days
Signal corr. filter[a]	8.2 min
Sig. and pow. corr. f.[a]	8.3 min

Table 4.3: Power model parameter selection effort [62]

[a]Executed on a 3 GHz AMD Opteron system.

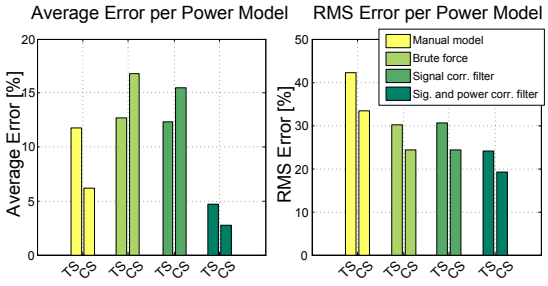

Figure 4.1: Average and RMS estimation error for different power models [62]

4.3.2 Power Model Evaluation

The high-level power models required for enabling the power emulation of the given test systems have been created using the automated power modeling approach. Table 4.4 and Table 4.5 summarize the power modeling parameters for both test systems.

Models	Single model for SoC
Characterization Method	Simulation-/Measurement-Based NNLS (entire SoC)
# Model Parameters	54 (entire SoC)

Table 4.4: Power model parameters for smart card microcontroller test system

Models	Core (IU, FUs, MMU) I/D Caches, Register File
Characterization Methods	NNLS (Core) eCACTI (Caches, RF)
# Model Parameters	53 (Core) 6 (Caches), 2 (RF)

Table 4.5: Power model parameters for multi-processor test system

A single model is covering the entire smart card microcontroller test system. In the characterization process training set data derived both from simulations and from measure-

ments have been used (see Table 4.4). For the multi-processor SoC (MPSoC) test system the core's power model has been created from simulation-based training set data using the automated power modeling approach. For both test systems power model parameters have been automatically selected and coefficients have been derived using a nonnegative least squares (NNLS) coefficient fitting algorithm. Due to the lack of a physical implementation of the cache memories and the register files, these components have been modeled using characterization data obtained from the eCacti tool [67] (see Table 4.5).

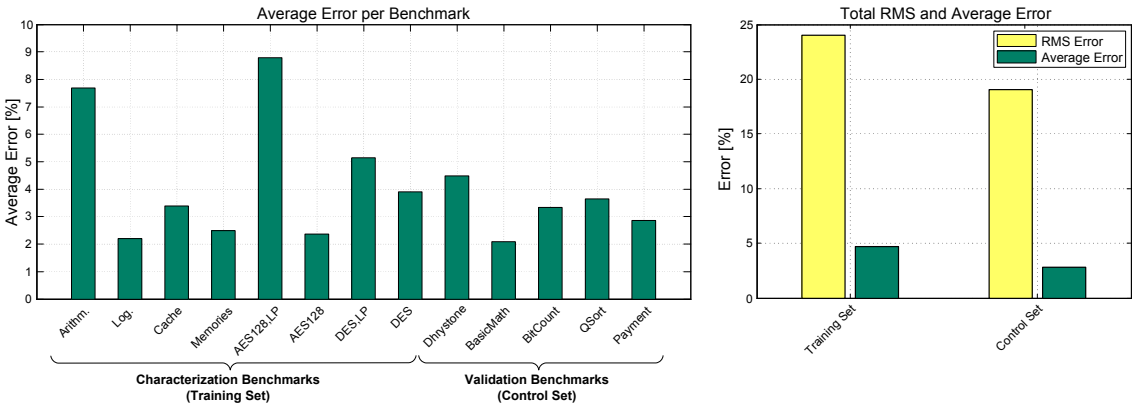

Figure 4.2: Per-benchmark average error (left) and total average and RMS error (right) for the smart card microcontroller power model

Figure 4.2 and 4.3 illustrate the average error for training set and control set benchmarks for both test systems in comparison to detailed gate-level power simulations. For the smart card microcontroller test system the average error per benchmark remains below 9% for the training set (TS) and below 5% for the control set (CS). Summarized, for the smart card test system the power model achieves an average error across all benchmarks of 4.71% (TS) and 2.78% (CS). For the root-mean-square (RMS) error these numbers are 24.04% (TS) and 19.07% (CS).

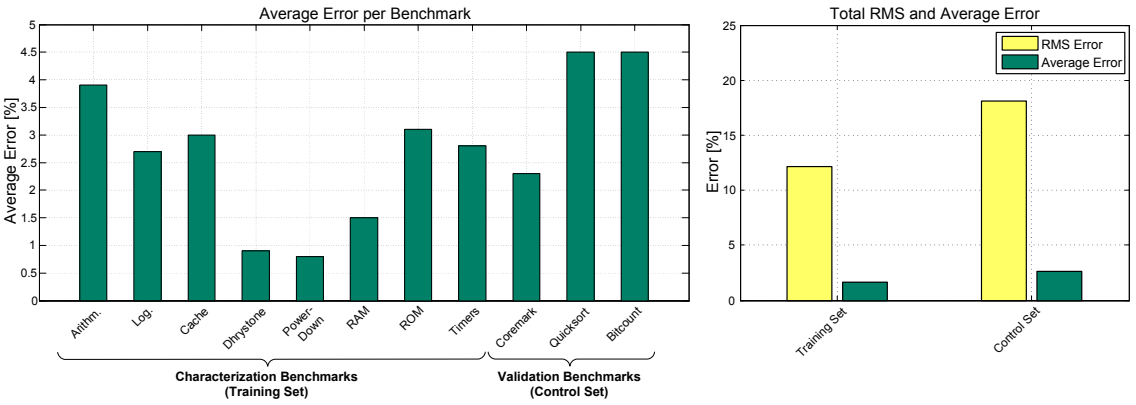

Figure 4.3: Per-benchmark average error (left) and total average and RMS error (right) for the multi-processor core power model

For the MPSoC core power model the average error per benchmark remains below 4%

(TS) and 5% (CS) respectively. Summarized, for the MPSoC core power model the average error is 1.66% (TS) and 2.60% (CS). The RMS error amounts to 12.14% (TS) and 18.12% (CS). Considering the high-level nature of the used power macromodeling technique this power model accuracy can be considered sufficiently high for use in the power emulation methodology. The publications in Sections 6.3 and 6.4 discuss the power modeling of the two test systems in greater detail.

4.3.3 Power Emulation Hardware Generation and Implementation

After performing the automated power modeling, the obtained high-level power macromodels have been implemented in hardware to enable the power emulation of the given test systems. To this end, both test systems have been augmented with a power emulation unit as illustrated in Figure 4.4 and Figure 4.5. The concept for the automated power emulation hardware generation and implementation, as outlined in Section 3.2.3, has been used to implement the automated HDL model adaptation[1,2] as well as the generation of the power emulation unit itself[3].

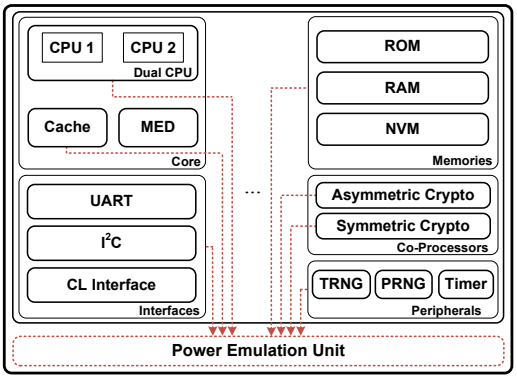

Figure 4.4: Integration of smart card test system and power emulation unit

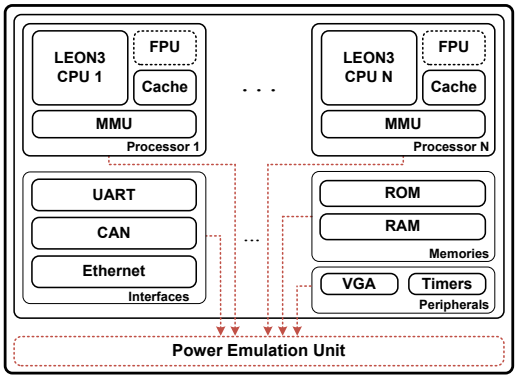

Figure 4.5: Integration of multi-processor test system and power emulation unit

Table 4.6 provides an overview of the execution time required for the automated power emulation hardware implementation of the smart card microcontroller test system, performed on a 3.2 GHz Intel Xeon server system. The HDL model of this test system comprises 400 HDL files in total, approximately 10% thereof have been automatically adapted to enable the monitoring of internal signals that represent power model parameters. Furthermore, the timely effort for executing the template-based HDL model generation of the power emulation unit itself is listed. While the manual effort varies largely with the hardware designer's expertise and knowledge regarding the test system's architecture, in any case the automated approach largely reduces the timely effort.

In terms of hardware overhead Table 4.7 and Table 4.8 provide FPGA resource utilization figures for the two test systems. For the smart card system the power emulation

[1] D. Reitz, "Automated test system adaptation for power emulation," *Bachelor's Thesis, Graz University of Technology*, 2009.

[2] M. Schön, "Automated test system adaptation for power emulation: Extended features," *Technical Report, Infineon Technologies Austria AG*, 2010.

[3] D. Wittibschlager, "Modelling of a real-time power emulation architecture," *Bachelor's Thesis, Graz University of Technology*, 2009.

HDL Implementation Method	Effort / Execution Time
HDL analysis	28.9 s
Dependency extraction	4 s
Signal routing & Write-back	24.3 s
PE unit HDL generation	0.2 s
Automatic implementation total	57.4 s
Manual implementation	several hours

Table 4.6: Comparison of HDL implementation effort for power emulation on smart card microcontroller test system [62]

Component	Utilization[a] [%]
Test System	66.0
Power Emulation	1.6
Total (Test System + PE)	67.6

Table 4.7: FPGA utilization for power emulation of smart card test system

[a]Percentage of total available LUTs on Altera Stratix II platform.

Component	Utilization[a] [%]
Test System	80.8
Power Emulation	7.8
Total (Test System + PE)	88.6

Table 4.8: FPGA utilization for power emulation of quad-core multi-processor test system

[a]Percentage of total available LUTs on Xilinx Virtex 5 platform.

accounts to an additional 1.6% of look-up table (LUT) utilization on top of the 66.0% used for purely functional emulation. For the MPSoC test system, used in a quad-core configuration, the additional utilization accounts to 7.8% (∼1.95% per core) as compared to 88.6% for the functional emulation. Hence, the impact of the power emulation approach onto the emulation platform can be considered minor and still leaves a margin for implementing additional functionality. The power emulation hardware generation and implementation for both test systems is discussed in greater detail in the publications in Section 6.3 and Section 6.4.

4.3.4 Power Emulation Performance

The high emulation speed represents one of the main advantages of the high-level power emulation methodology as compared to low-level approaches or software simulators. Table 4.9 illustrates the speed-up achieved for the MPSoC test system for some benchmarking applications. The power emulation performance of both test systems is discussed in greater detail in the publications in Section 6.2 and Section 6.4.

Benchmarks	Bitcount	Coremark	Quicksort	OS Booting
RTL Simulation Time	26.46 s	144.56 s	30.71 s	7.39 d[b]
IS Simulation[a]	2.45 ms	11.81 ms	2.51 ms	54.23 s[b]
Emulation Time	0.50 ms	2.39 ms	0.50 ms	11 s
Speedup vs RTL Sim.	52744	60507	60890	58047[c]
Speedup vs IS Sim.[a]	4.9	4.9	5.0	4.9[c]

Table 4.9: Simulation vs emulation time comparison for multi-processor test system

[a]Purely functional simulation, no power estimation process.
[b]Value is extrapolated from average speedup of emulation vs RTL/IS simulation.
[c]Average value.

4.4 Emulation-Based Power-Aware HW/SW Codesign

4.4.1 Joint Power Emulation and Performance Monitoring

By additionally enabling the monitoring of performance indicators of the given system-under-test, trade-offs between the power/energy consumption and the performance of the system can be more easily explored. To this end, performance monitoring units, as introduced in Section 3.3.1, have been added alongside the power emulation hardware. Figure 4.6 depicts the power emulation and performance monitoring architecture implemented for the LEON3 multi-processor test system[4]. A number of performance-relevant events, as listed in Table 4.10, are covered by monitoring internal signals of each core's pipeline, register file, and cache memories.

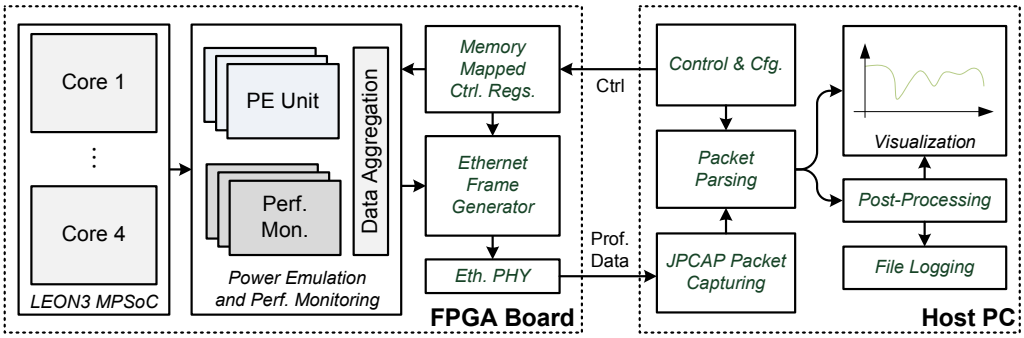

Figure 4.6: Integration of multi-processor test system and joint power emulation and performance monitoring functionality

General	Execution Stage	Register File	Instruction Cache	Data Cache
Clock Cycles	Add, Logic, Shift	S/D Read	Read Hit	Read/Write Hit
Stall Cycles	Mul, Div Op.	Write	Read Miss	Read/Write Miss

Table 4.10: Monitored performance events

The benefits of the joint power emulation and performance monitoring approach in the HW/SW codesign process are illustrated by profiling prototypical applications. In the first example, as depicted in Figure 4.7, the profiling of different compiler optimization levels applied to the Coremark [68] benchmark is shown. While the unoptimized version of the benchmark (compiler level 0) exhibits a number of data cache stall cycles, the number of stalls is largely reduced in the optimized version (compiler level 3), hence, largely reducing the execution time. Note that these optimizations are purely compiler-based, i.e., no manual code modifications have been performed.

The second example, depicted in Figure 4.8, illustrates the usefulness of the joint power emulation and performance monitoring platform during manual software optimization. The unoptimized version subsequently modifies a large data array in two execution steps. However, due to the large size of the array it cannot be entirely held in the cache and has to be loaded again for the second execution step, causing a number of data cache stalls.

[4]M. Lackner, "Design and implementation of a multi-core power and performance emulation platform," *Master's Thesis, Graz University of Technology*, 2010.

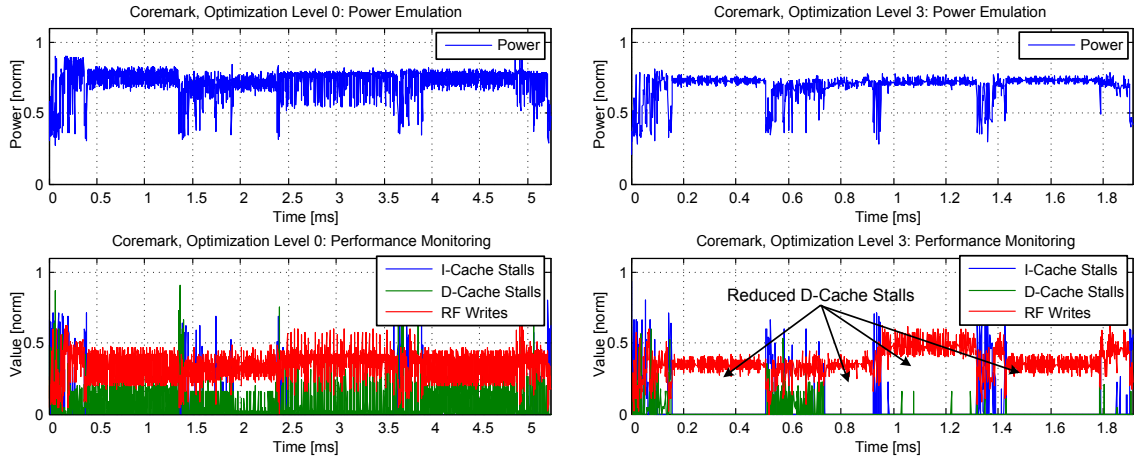

Figure 4.7: Emulation-based power and performance profiling illustrating the impact of compiler optimizations[5]

The optimized version the two execution steps are performed only on smaller parts of the array that are entirely fitting into the data cache. Therefore, the number of data cache stall cycles is being reduced, resulting in a shorter execution time.

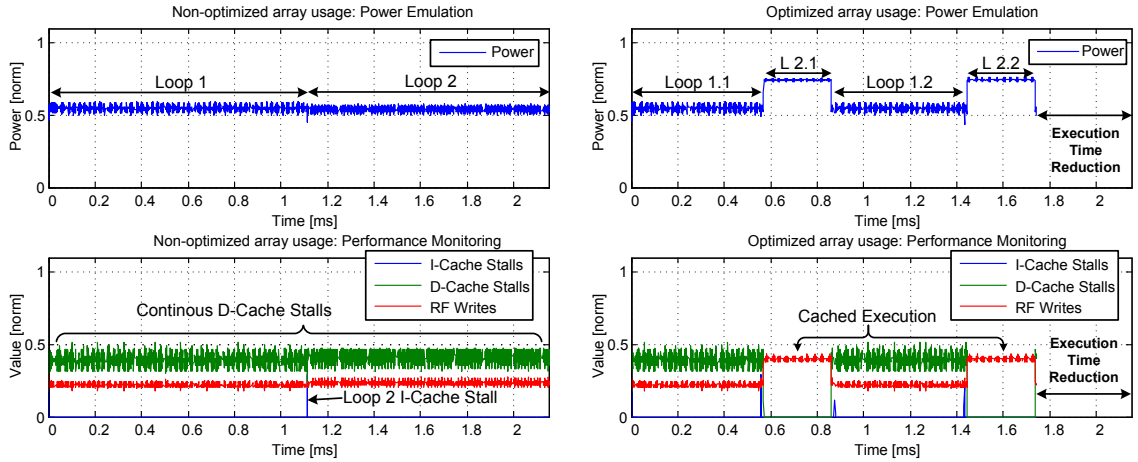

Figure 4.8: Emulation-based power and performance profiling illustrating the impact of manual array access optimizations[5]

The high emulation speed that represents one of the main advantages of the power emulation approach as compared to software simulators allows the profiling of applications exhibiting long execution times. In Figure 4.9 the profiling result of a task migration procedure on an SMP-enabled SnapGear Linux 2.6 [66] is shown. The migration process is clearly visible both in the cores' power profiles as well as in the performance profiles monitoring the cache activity. The detailed evaluation of the joint power emulation and performance monitoring approach is presented in the publication in Section 6.4.

[5]Data normalized due to existing confidentiality agreement.

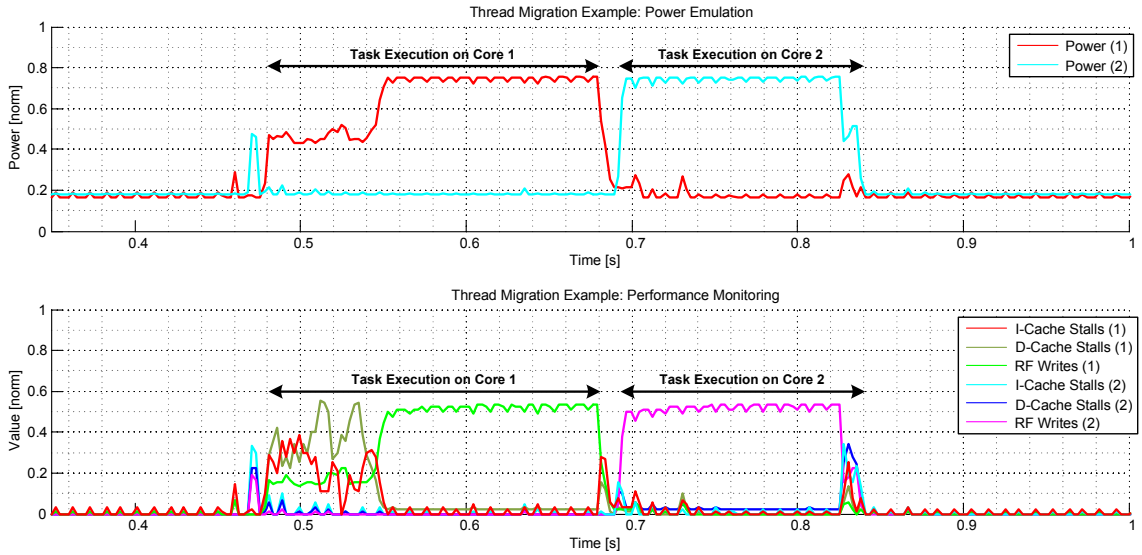

Figure 4.9: Emulation-based power and performance profiling of a Linux task migration between two processor cores[6]

4.4.2 Power-Aware Software Development Using Power Emulation

The integration of the power emulation methodology into the software development process in order to increase the power-awareness represents a major goal of the POWERHOUSE project. To this end, the high-level power emulation technique has been incorporated in the software development tool chains and development environments for both test systems using the concept outlined in Section 3.3.2.

For the smart card test system, script-based power emulation support has been integrated for Keil μVision [17] software projects. Furthermore the WaveViewer [20] Java application has been devised to analyze and visualize power emulation traces that have been recorded using the Hitex HiTop [69] IDE/debugger that is controlling the emulation platform. Furthermore, a Java-based power emulation GUI[7] has been implemented that performs the execution trace to source code correlation for both test systems, allowing for the quick assignment of power consumption estimates to individual source code lines (see Figure 4.10). The publication in Section 6.6 provides more details on the power-aware software development process based on the power emulation technique.

4.4.3 Emulation-Based Power Peak Optimization of Embedded Software

The run-time power profiles generated using the power emulation technique have been employed in the software-induced power peak optimization framework as introduced in Section 3.3.3. This framework has been evaluated for the smart card test system, assuming an RF-powered smart card device.

Figure 4.11 depicts the profiling result of an unoptimized benchmarking application

[6]Data normalized due to existing confidentiality agreement.

[7]S. Gether, "Design und Implementierung einer grafischen Benutzeroberfläche für Power Emulation," *Bachelor's Thesis, Graz University of Technology*, 2010.

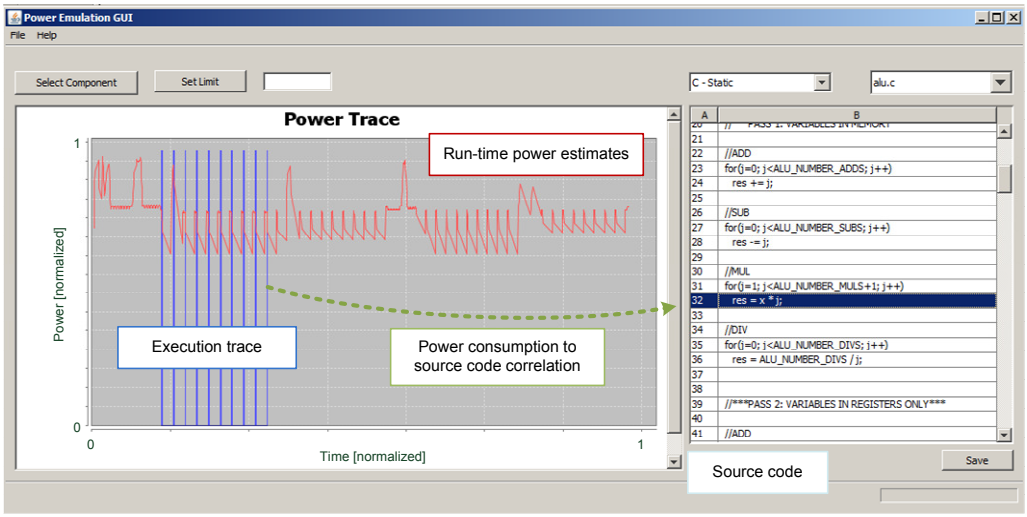

Figure 4.10: Power-aware software development GUI based on power emulation[8]

that exhibits a series of power consumption peaks leading to supply voltage drops below a critical limit. By modifying the benchmarking application by means of the automated power peak optimization framework, the critical power peaks can be diminished, reducing the severity of the supply voltage drops as illustrated in Figure 4.12. This is achieved by applying frequency scaling and non-functional instruction (NFI) insertion to the source code regions causing these peaks, hence decreasing their impact on the power consumption and the supply voltage.

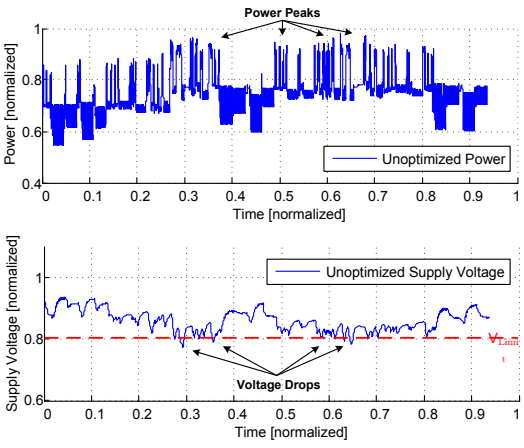

Figure 4.11: Emulated power consumption and resulting supply voltage profiles of non-optimized software application[8] [64]

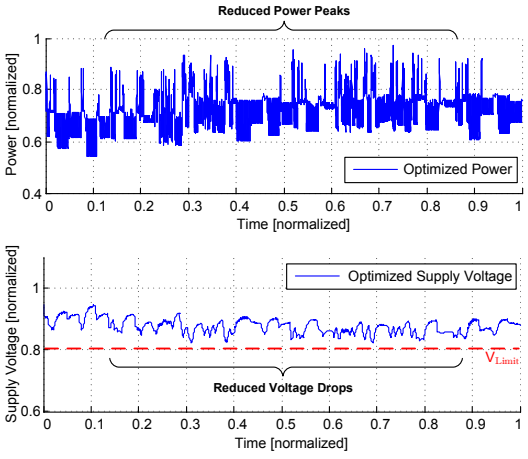

Figure 4.12: Emulated power consumption and resulting supply voltage profiles of optimized software application[8] [64]

While the insertion of power management control instructions improves the system's stability and reliability, it also inherently impacts the code size and the execution time. Table 4.11 and Table 4.12 compare these overheads for different benchmarking applications.

[8]Data normalized due to existing confidentiality agreement.

Regarding execution time the automatic approach (1.2% (Crypto) up to 6.8% (Authentication)) outperforms the manual optimization (∼10%) due to the finer granularity of code modifications. In terms of code size the increase for the manual approach is almost negligible (smaller than ∼1%) while for the automatic approach it is higher (0.2% (Crypto) up to 3.2% (Dhrystone)) as more frequency scaling control instructions and NFIs are inserted.

Benchmark	Execution Time[9] [%]		
	Original	Manual	Autom.
Authentication	100.0	110.8	106.8
Coremark	100.0	110.4	105.2
Crypto	100.0	110.5	101.2
Dhrystone	100.0	111.1	104.1

Table 4.11: Power peak optimization impact on the execution time

Benchmark	Code Size[9] [%]		
	Original	Manual	Autom.
Authentication	100.0	100.2	101.7
Coremark	100.0	100.1	102.7
Crypto	100.0	100.2	100.2
Dhrystone	100.0	101.1	103.2

Table 4.12: Power peak optimization impact on the code size

Hence, in terms of overhead the automatic approach slightly increases the code size but this increase is compensated by gains in execution time due to this higher selectivity of modifications as compared to manual optimizations. The automatic approach assists software engineers and decreases the required development as well as optimization effort. The emulation-based power peak optimization of embedded software is discussed in greater detail in the publication in Section 6.7.

[9]Data normalized due to existing confidentiality agreement.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

With the rising number of mobile devices, power-aware hardware/software codesign has become increasingly important. The rising design complexity of novel system-on-chip designs renders the simulation-based power consumption profiling increasingly difficult due to extensive simulation times. Therefore, hardware-accelerated power estimation techniques based on the emulation principle have been devised.

In this work a novel methodology has been introduced that allows for automatically creating a high-level power emulation platform for use in the hardware/software codesign process of system-on-chip designs. The methodoloy is based on a high-level power macromodeling approach that minimizes the required hardware overhead. By observing a number of power-relevant signals in the design-under-test, the power model derives cycle-accurate power consumption estimates for the system and its sub-components.

For the purpose of creating the power macromodel of a given design-under-test an automated power modeling method is employed. The method derives the high-level power macromodel from system activity and power consumption training set data generated by simulations or measurements. In a parameter selection stage internal system signals are selected as potential power model parameters if their activity data correlate well with the power consumption data. Afterwards a model coefficient fitting stage, utilizing a least squares fitting algorithm, computes coefficients for the selected parameters.

The previously created power model needs to be implemented in hardware for implementation on the emulation platform. To this end, an automated power emulation hardware implementation method, consisting of two stages, is utilized. First, the HDL model of the design-under-test needs to be modified to allow for the monitoring of internal signals that are used as power model parameters. An adaptation algorithm analyzes all HDL files associated with the design and performs the "routing", i.e., the creation of the required internal connections, of signals originating in various sub-components at various levels of hierarchy. Second, the power emulation unit itself is created from HDL templates and adapted to the used power model. The resulting adapted HDL model can afterwards be processed by a standard FPGA implementation tool flow that generates the netlist of the power emulation platform.

Furthermore, the utilization of the resulting power emulation platform in the power-

aware hardware/software codesign process has been illustrated. The integration of the power emulation platform within a standard software development flow allows for providing power consumption feedback to software engineers, enabling truly power-aware software development and optimization. Additionally, the profiles can be used to detect software-induced power consumption peaks that threaten the reliable operation of the system. A framework utilizing the run-time power emulation traces for detecting power consumption peaks, assessing the supply-voltage impact and reducing their impact by means of software-controllable power management has been presented. By extending the power emulation platform with the ability of monitoring hardware performance indicators, both power- and performance-awareness in the codesign process can be improved. The joint power emulation and performance monitoring approach allows for performing hardware and software power optimizations while considering their performance impact. Hence, it enables the quick evaluation of trade-offs between a system's power/energy consumption and its performance.

For the purpose of illustrating the effectiveness of the automated power emulation methodology, an evaluation on two test systems has been performed. The first test system is represented by a heterogeneous smart card microcontroller SoC whereas the second test system is constituted by a multi-processor SoC (MPSoC). The intended low-power operating environment and the considerable amount of hardware complexity, including power management features, make these two test systems particularly interesting for the evaluation of the automated power emulation methodology.

For creating the power emulation platforms of both test systems, high-level power macromodels have been created using the automated power modeling approach. For the smart card test system the automated power models have been additionally benchmarked against different modeling techniques, illustrating a reduction of the average error from 11.78% to 4.71% and for the RMS error from 42.24% to 24.04% as compared to gate-level simulations. For the MPSoC test system the derived per-core power model achieves an average estimation error of 2.60% and an RMS of 18.12%. Considering the high-level nature of the used power macromodels these numbers illustrate the applicability of the automated power modeling approach in the power emulation methodology.

By using the automated power emulation hardware implementation approach, the power emulation units for the power models of both test systems have been generated and the original HDL models have been adapted to allow for the monitoring of model-relevant signals by the power emulation units. In terms of hardware overhead, for the smart card system the power emulation accounts to an additional 1.6% of FPGA utilization as compared to 66% already required for the functional emulation. For the quad-core MPSoC system the power emulation units account to 7.8% ($\sim$1.95% per core) as compared to 80.8% used by the functional emulation. Hence, the impact of the high-level power emulation approach in terms of hardware overhead is relatively small, enabling the full-system power emulation of complex SoCs using moderately priced FPGA platforms.

Furthermore, the use of the resulting power emulation platforms in the HW/SW codesign process has been evaluated. By adding performance monitoring units to the MPSoC test system, a total of 18 performance events per core can be detected and logged. The benefits of the joint power emulation and performance monitoring approach has been illustrated by comparing the impact of manual as well as compiler optimizations on bench-

marking applications. For the Keil μVision and Hitex HiTop software development tool chains power emulation support has been integrated, providing graphical user interface applications for the analysis and the source-code-correlation of the run-time power consumption traces. Additionally, the applicability of the emulation-based power peak reduction framework has been illustrated for the smart card test system, assuming an RF-powered implementation. In comparison to manual optimizations, the automated insertion of software power management instructions into the application's source code reduces execution time overheads.

5.2 Directions for Future Work

5.2.1 Hybrid Power and Fault Attack Emulation for Trusted SoC Design

In recent years the use of mobile system-on-chip devices in security applications has strongly increased. A growing number of higher-order fault attacks have been devised with the aim of extracting sensitive information from these devices. For detecting and mitigating the effects of these fault attacks more complex fault detection and recovery mechanisms will have to be integrated into future trusted SoCs. Due to the power-limited operating environment inherent to mobile applications, the power- and energy-impact of these mechanisms is of increased interest. Therefore, both power- and fault-attack-awareness will have to be considered in the design process of mobile security-related SoCs. Furthermore, the effectiveness of these mechanisms will have to be evaluated for a large set of different fault attacks in realistic application scenarios that entail extensive execution times. The simulation-based analysis and evaluation of the power-consumption-impact of both fault attack detection as well as mitigation mechanisms will therefore be difficult due to extensive simulation times.

The follow-up project *"**Pow**er **E**mulato**r** and **Mo**del Based **De**pendability and **S**ecurity Evaluation Platform"* (POWERMODES) [70] will address the limitations of state-of-the-art simulation-based approaches by investigating the feasibility of a hybrid power and fault attack emulation platform. By combining a fault attack emulation approach with the automated high-level power emulation methodology presented in this thesis, the platform could be used to enable rapid power-effectiveness evaluations of novel fault attack detection and recovery mechanisms.

5.2.2 Run-time Thermal Estimation Based on Power Emulation

The combination of ever increasing performance requirements with higher levels of integration enabled by CMOS technology scaling, has led to higher system-on-chip operating temperatures. These higher temperatures negatively influence both the performance as well as the reliability of SoCs by, e.g., exponentially increasing leakage currents, increasing interconnect resistivity, degrading device lifetime, etc. Therefore, the operating temperature has to be already considered during the design process for appropriately selecting chip packages and cooling solutions. In many cases, however, the worst-case design in terms of cooling solution is prohibitively expensive, therefore, requiring system designers to utilize run-time power and temperature management techniques for keeping the system's temperature below a critical limit. For analyzing the run-time temperature behavior of SoC

designs and evaluating the effectiveness of thermal management strategies, execution time intervals of several seconds have to be considered in order to reach steady state temperature distributions accounting for the thermal capacities of the design. Both the thermal as well as the power simulation of these temporal intervals using thermal finite element simulators and power simulators respectively, represent computationally expensive tasks resulting in prohibitively long simulation times.

Accelerated architectural level thermal simulators, such as the Hotspot compact thermal model proposed by Skadron et al. [71], address the slow simulation speed of finite-element-based approaches by representing a given SoC design as a thermal resistor-capacitor (RC) circuit. The coupling of this thermal RC circuit simulation with the high-level power emulation method derived in the POWERHOUSE project represents an interesting approach for enabling long-time thermal evaluations and has already been investigated within the project[1]. By integrating high-level power models alongside the power emulation hardware within the emulation platform the thermal estimation process could be further accelerated, allowing for novel approaches such as, e.g., the consideration of run-time temperature-feedback in the power emulation process. Therefore, the emulation-based power and thermal estimation approach promises to be an interesting topic for future work.

[1] M. Schaffernak, "Thermal estimation for SoC design based on power emulation," *IT Project Report, Graz University of Technology*, 2010.

Chapter 6

Publications

This chapter comprises the publications covering the previously presented methodology (Chapter 3) and the evaluation results thereof (Chapter 4), as well as the motivational example (Chapter 1) in greater detail.

Publication 1: *A Low-Power ASIP for IEEE 802.15.4a Ultra-Wideband Impulse Radio Baseband Processing*, 12$^{\text{th}}$ IEEE Design, Automation & Test in Europe Conference & Exhibition 2009 (DATE '09), Nice, France, 20–24 April 2009.

Publication 2: *An Emulation-Based Real-Time Power Profiling Unit for Embedded Software*, 9$^{\text{th}}$ IEEE International Symposium on Systems, Architectures, Modeling, and Simulation 2009 (SAMOS '09), Samos, Greece, 20–23 July 2009.

Publication 3: *Automated Power Characterization for Run-Time Power Emulation of SoC Designs*, 13$^{\text{th}}$ IEEE Euromicro Conference on Digital System Design: Architectures, Methods and Tools 2010 (DSD '10), Lille, France, 1–3 Sept. 2010.

Publication 4: *An Emulation-Based Platform for Power- and Performance-Aware HW/SW Development of Embedded Multi-Core Systems*, submitted for publication / under review, 2011.

Publication 5: *Power emulation: Methodology and applications for HW/SW power optimization*, 8$^{\text{th}}$ IEEE/ACM International Conference on Formal Methods and Models for Codesign 2010 (MEMOCODE '10), Grenoble, France, 26–28 July 2010.

Publication 6: *Accelerating Embedded Software Power Profiling Using Run-Time Power Emulation*, 19$^{\text{th}}$ International Workshop on Power and Timing Modeling, Optimization and Simulation 2009, (PATMOS '09), Delft, The Netherlands, 9–11 Sept. 2009, published in Springer Lecture Notes in Computer Science, 2010, Volume 5953.

Publication 7: *An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software*, 20$^{\text{th}}$ International Workshop on Power and Timing Modeling, Optimization and Simulation 2010, (PATMOS '10), Grenoble, France, 8–10 Sept. 2010, published in Springer Lecture Notes in Computer Science, 2011, Volume 6448.

Publication 8: *Estimation-Based Run-Time Power Profile Flattening for RF-Powered Smart-Card Systems*, 11$^{\text{th}}$ IEEE Asia Pacific Conference on Circuits and Systems, (APCCAS '10), Kuala Lumpur, Malaysia, 6–12 Dec. 2010.

The publications in this chapter discuss various aspects of the automated power emulation methodology for power-aware HW/SW codesign (Chapter 3) as well as evaluation results and case studies (Chapter 4), as depicted in Figure 6.1.

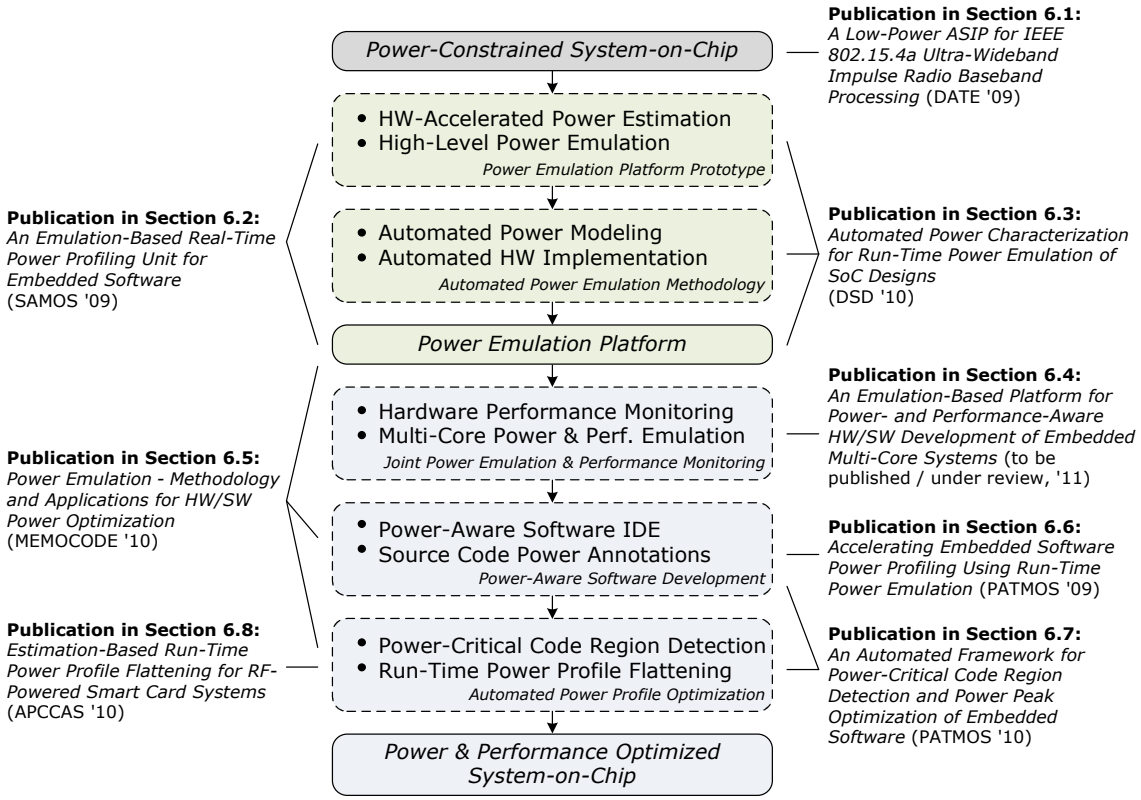

Figure 6.1: Overview of the publications covering the automated power emulation methodology for power-aware HW/SW codesign

The publication in Section 6.1 serves as a motivational example by illustrating the need for fast power consumption profiling methods to support the power-aware hardware/software codesign process. The high-level power emulation technique that allows for the rapid power profiling of a given system-under-test is introduced in the publications in the Sections 6.2 and 6.3. The novel automated power emulation methodology, consisting of automated power modeling and power emulation hardware implementation, is introduced in Section 6.3.

The resulting high-level power emulation platform is then utilized in the power-aware HW/SW codesign process as illustrated in the publications in the Sections 6.4-6.8. By extending the power emulation method for use in heterogeneous multi-core environments and by additionally enabling the monitoring of hardware performance events, a truly power- and performance-aware development and optimization process can be realized (Section 6.4). Furthermore, the integration of the run-time power profiling results, obtained from the power emulation platform, into the standard development process is illustrated in the Sections 6.5 and 6.6. Finally, the use of the run-time power estimates for the purpose of detecting and reducing power consumption peaks by means of a software-based framework (Section 6.7) and a hardware-based approach (Section 6.8) is shown.

A Low-Power ASIP for IEEE 802.15.4a Ultra-Wideband Impulse Radio Baseband Processing

Christian Bachmann*, Andreas Genser*, Jos Hulzink†, Mladen Berekovic‡ and Christian Steger*

*Institute for Technical Informatics, Graz University of Technology
Inffeldgasse 16, 8010 Graz, Austria

†IMEC Netherlands, Holst Centre
High Tech Campus 31, 5656 AE Eindhoven, The Netherlands

‡IDA, TU Braunschweig
Hans-Sommer-Str. 66, 38106 Braunschweig, Germany

***Abstract*—The IEEE 802.15.4a amendment has introduced ultra-wideband impulse radio (UWB IR) as a promising physical layer for energy-efficient, low data rate communications. A critical part of the UWB IR receiver design is the low-power implementation of the digital baseband processing required for synchronization and data decoding. In this paper we present the development of an application-specific instruction-set processor (ASIP) that is tailored to the requirements defined by the baseband algorithms. We report a number of optimizations applied to the algorithms as well as to the hardware architecture. This enables performance increases up to a factor of 122x and energy consumption decreases up to 90x as compared to a 16-bit baseline architecture. Furthermore, this ASIP offers greater flexibility due to programmability as compared to an ASIC implementation.**

I. Introduction

Within IMEC's Human++ research program [1], ultra-low power radio transceivers are identified as the key components in future wireless sensor networks for mobile healthcare applications. IEEE 802.15.4a ultra-wideband impulse radio (UWB IR) represents a potential communication scheme for these low-power radios.

UWB IR transceivers typically make use of an analog/digital partitioning: An analog radio front-end is used to transmit and receive modulated RF signals. The baseband signals are encoded and decoded by a digital baseband processor as depicted in Fig. 1.

In first generation systems an ASIC implementation has been employed for digital baseband processing [2]. This implementation has been extended to an application-specific instruction-set processor (ASIP), offering greater flexibility as well as better energy-efficiency [3]. Both implementations are, however, operating on a predecessor modulation scheme and do not support the latest revision of the standard. This paper presents the design and implementation of a new baseband ASIP supporting the low-power and low-rate modes of the novel IEEE 802.15.4a standard amendment [4]. In addition to the previous ASIP, basic channel estimation and a rake receiver structure are being considered.

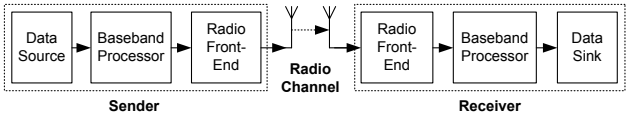

Fig. 1. Baseband processing principle

Section II gives a brief introduction to IEEE 802.15.4a ultra-wideband impulse radio and baseband algorithms. The design process of the baseband ASIP is described in Section III. Furthermore, the design methodology is outlined. In Section IV various optimizations are presented. Finally, Section V illustrates implementation results and in Section VI conclusions are drawn.

II. IEEE 802.15.4a Ultra-Wideband Impulse Radio

Ultra-wideband impulse radio communication is based on the emission of very short pulse waveforms, exhibiting wideband spectral characteristics in the frequency domain. Due to the stringent UWB emission limits (e.g., [5]), the energy contained in a single pulse is very low. In order to be still able to extract information out of UWB transmitted data and to fight noise and interference, a technique called *spreading* is used. A symbol to be transmitted is represented by a number of consecutive pulses instead of a single pulse. The pseudo-random bit sequence determining the polarity of these pulses is referred to as *spreading code*.

In order to restore the original bit, i.e. to *despread* the sequence of pulses, a cross-correlation operation

$$y = \sum_{i=1}^{N} (x[i] \cdot C[i]) \tag{1}$$

of N pulses x with the spreading code C, also of length N, is calculated. The value of the decision variable y determines the value of the original bit.

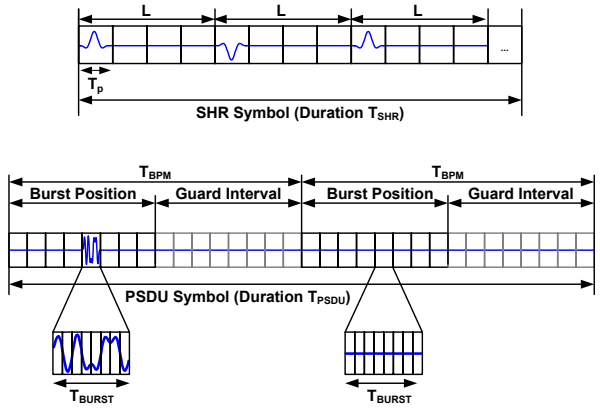

Fig. 2. SHR (top) and PSDU (bottom) symbols

A. IEEE 802.15.4a Frame Structure

In IEEE 802.15.4a UWB IR systems data is being transmitted in frames consisting of three major sub-parts:

- Synchronization header (SHR) / Preamble
- Data header (PHR)
- Data unit, i.e. the actual payload data (PSDU)

The synchronization header, also referred to as *preamble*, is being transmitted in order to aid receiver algorithms in timing acquisition and frame synchronization. It also serves the purpose of channel estimation and gain control setting optimization. This preamble is composed of *SHR symbols* that contain a number of isolated pulses as depicted in Fig. 2.

The data unit and its header are constructed out of *PSDU symbols*. Each symbol is able to carry two bits of information: One bit is coded in the symbol half in which a burst, i.e. a concatenation of pulses, occurs (burst position modulation, BPM). Another bit is used to determine the polarity (phase) of the burst itself (binary phase shift keying, BPSK). The state of a linear feedback shift register (LFSR) varies the spreading code for each transmitted PSDU symbol. This spreading code determines the burst sequence as well as the exact burst position within one of the symbol halves (”hopping code”). A PSDU symbol is depicted in Fig. 2.

B. Baseband Algorithms

When designing the UWB IR baseband algorithms, the frame structure as pointed out in Section II-A has to be taken into account. The different sub-parts of a frame are processed by different sub-algorithms.

1) Synchronization / Timing Acquisition: The synchronization phase is commonly composed of noise estimation, signal detection (coarse acquisition), fine acquisition, channel estimation and end-of-preamble search sub-states [7]:

- Initialization & Noise Estimation
  All necessary data structures are initialized and a threshold value corresponding to current noise levels is computed.
- Signal Detection (SD)
  The purpose of the signal detection state is to detect the presence of an ultra-wideband impulse radio signal within input data that is corrupted by noise.
- Fine Acquisition (FA) & Channel Estimation
  The fine acquisition algorithm is used to optimize synchronization and to detect false positives during signal detection. Furthermore, channel estimation is integrated into this algorithm.
- End-Of-Preamble (EOP) Search
  The EOP search algorithm detects the end of the preamble (synchronization header) and the start of the actual payload data within the UWB IR frame.

2) Payload Decoding (PD) / PSDU Data Despreading: Once all synchronization algorithms have been successfully passed, the decoding of the actual payload data can be carried out. A selective-rake receiver structure is employed in order to achieve higher signal-to-noise ratios. The channel coefficients previously derived by the channel estimation algorithm are used as weights within the rake receiver structure.

III. Design of a Baseband ASIP

The methodology used for designing this ASIP employs a gradual exploration process covering multiple processor architectures. The use of the following architectures for baseband processing is investigated:

- Scalar RISC
- Vector RISC
- Vector Very Long Instruction Word (VLIW)

Each of these architectures is adapted to optimize the execution of the baseband algorithms.

A. ASIP Design Flow

In the design process the IP Designer [6] tool suite by Target Compiler Technologies as depicted in Fig. 3 was employed.

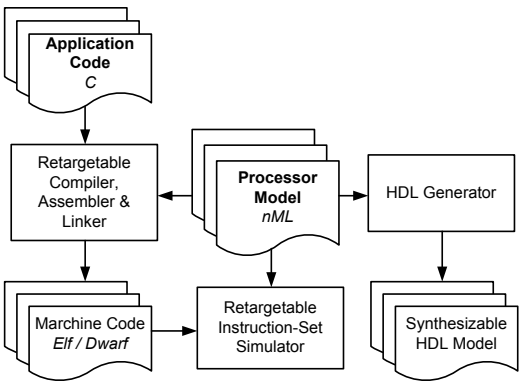

Fig. 3. Target tools environment [6] used for baseband ASIP design

It allows for fast architectural exploration due to *retargetability*. The high-level description of the processor facilitates rapid architectural changes.

The highly optimizing C-compiler, the (dis)assembler and the linker, are used to generate machine code for a given processor architecture. This code is then executed and benchmarked on the cycle-accurate instruction-set simulator. If performance requirements are met, a HDL model can be generated out of the same high-level description.

B. Architectural Exploration

We use a scalar 16-bit Harvard RISC processor as depicted in Fig. 4 as the starting point in the architectural design process. It comprises a 16-bit arithmetic logical unit (ALU), a 16 $\times$ 16-bit register file (RF), load-store unit, branch unit and instruction decoder. Furthermore, a custom extension unit is used to speed up scalar UWB operations.

Due to the fact that the cross-correlation, which is heavily employed by large parts of the baseband algorithms, is an ideal candidate for parallelization, the move to a vector processor seems obvious. A single instruction multiple data (SIMD) architecture is hence used to investigate the impact of data parallelism on execution performance. The vector architecture as depicted in Fig. 4 is derived from the scalar architecture but additionally includes a vector ALU, a vector register file and vector data memory. In addition to standard vector operations, dedicated UWB application-specific instructions are enabled through the use of a custom extension unit and a spreading code register.

In order to support real-time processing of IEEE 802.15.4a UWB IR baseband data, the algorithmic functionality specified above has to be executed on the baseband architecture obeying several timing limitations. The minimum time spans of SHR and PSDU symbols as well as the amount of data sampled per symbol have to be taken into account. Maximum input data rates of 1250.0 Mbit/s for SHR symbols[1] and 624.4 Mbit/s for PSDU symbols[1] have to be sustained by the architecture. Both, scalar and vector architecture, fail to meet these data rates for moderate clocking frequencies (<250 MHz).

[1]These numbers are derived from timing requirements set forth in the standard [4], an assumed ADC resolution of 5 bit as well as algorithm-dependent settings.

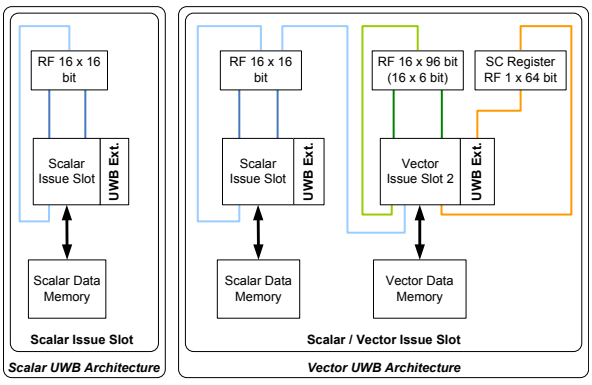

Fig. 4. Architectural overview of scalar (left) and vector baseband processors (right)

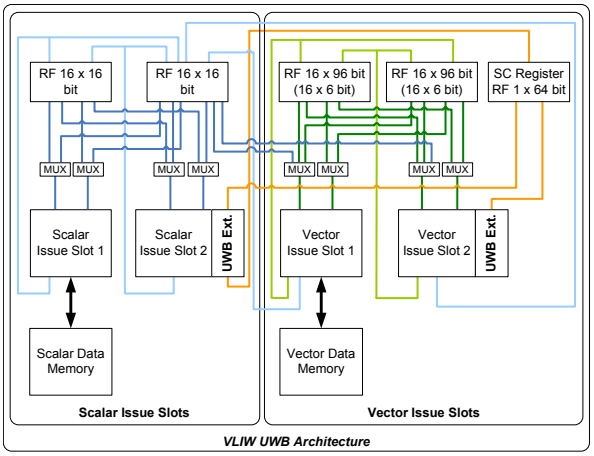

Fig. 5. Architectural overview of final VLIW baseband processor

To achieve sufficient throughput, our proposed UWB baseband architecture resembles a four issue-slot VLIW architecture, consisting of two slots containing scalar and control functional units and two slots containing vector units. Functional units specially tailored to the requirements of baseband processing are present in the design and can be accessed through custom instructions. Register-file sizes are adapted to the resolution of the ADCs used in the radio front-end as well as to algorithm requirements. Similar to the vector processor described above, a dedicated spreading code register is present. Fig. 5 presents an architectural overview of the final baseband processor.

IV. ASIP Optimizations

A. Vector Correlation

A vital part of UWB baseband processing is the cross-correlation of input data and spreading code as expressed in Equation 1. The multiplications inside the sum can be calculated independently from each other, hence enabling their parallel computation. Due to the binary or ternary structure of the spreading code the multiplication itself can be implemented as invert/replace operation.

Furthermore, the different sub-sums can partly be calculated in parallel, resulting in an adder-tree of depth $log2(n)$ where n is the number of vector elements.

B. Custom Instructions

The instruction-set of the VLIW processor is extended by a number of custom instructions as listed in Table I. Scalar operations are used to load, read out and modify the spreading code register. Furthermore, special arithmetic operations are present in the form of the addition of absolute values as well as the emulation of a linear feedback shift register (LFSR).

Custom vector instructions are used to implement the parallelized computation of the cross-correlation as pointed out

TABLE I
LIST OF IMPLEMENTED CUSTOM INSTRUCTIONS

Instruction	Input par.	Outp. par.
Addition of absolute values	(scl,scl)	(scl)
Spreading code load[a]	(scl)	(-)
Spreading code store[b]	(-)	(scl)
Spreading code rotate[a]	(-)	(-)
Spreading code shift-load[a]	(scl)	(-)
LFSR clocking	(scl)	(scl)
Vector SC correlation (PSDU)	(vec)	(scl)
Vector SC correlation (SHR)	(vec)	(scl)
Vector shift	(vec)	(vec)

[a]Internal spreading code register is modified.
[b]Data is read from internal spreading code register.

in Section IV-A. Different variants for header as well as payload decoding, operating on different representations of the spreading code, are available. An additional vector logic operation is present in the form of a vector shift instruction.

C. Algorithmic Optimizations

The baseband algorithms are adapted to the architectural extensions implemented in the ASIP. The synchronization as well as the payload decoding algorithms harness the cross-correlation custom instructions as well as vectorization. Expensive mathematical operators such as divisions, squares and roots can be replaced by suitable shift and absolute value add instructions. Furthermore, by exploiting the size of the register files, a multi-buffer synchronization approach can be implemented. This speeds-up the signal detection state by correlating more than one input data vector during each synchronization phase.

D. Low-Power Techniques

Techniques for low-power logic design employed within the baseband processor are *clock gating* and *operand isolation*. By applying the clock gating technique [8] the distribution of the clock signal to inactive modules of the processor is avoided. The clock signal is then turned off, reducing the power dissipation due to switching activity.

Operand isolation [9], often also referred to as *guarded evaluation*, is used to avoid the propagation of switching activity to inactive parts of the architecture's datapath. Thus, power dissipation due to unnecessary activity is eliminated in unused modules. Both types of logic optimization are performed automatically by the Target HDL generator [10].

V. EXPERIMENTAL RESULTS

Experimental results for evaluating the impact of architectural optimizations are structured into *performance* and *power consumption metrics*. Performance metrics for different processor architectures have been determined by means of cycle count measurements on the instruction-set simulator provided by the Target tool suite. For that purpose, various versions of the baseband algorithms have been executed on the different baseband processor architectures as introduced in Section III-B.

In order to obtain power consumption characteristics of the final UWB baseband processor, the VHDL model generated by the ASIP tool flow, extended with the user primitives for the custom functional units, is synthesized as well as placed and routed for a TSMC 90nm process including memories. The baseband algorithms are then simulated on the back-annotated netlist of the placed and routed processor including parasitics using Cadence NCSim 5.7. The value change data (VCD) information obtained during these simulations is used to obtain power consumption values with Synopsys PrimeTime PX Z2007.

A. Performance Results

For evaluating the overall performance of an algorithm-architecture combination, the different sub-algorithms have been profiled while processing an entire UWB IR frame. Fig. 6 and Fig. 7 depict the worst-case amount of cycles consumed by the different sub-parts of the baseband algorithm. In Fig. 6 the cycle counts for the execution on the basic scalar architecture are shown. Fig. 7 illustrates the cycle counts for processing the same data executing the optimized algorithms on the optimized vector VLIW baseband processor. It can be seen that the cycle count numbers have been drastically reduced due to these optimizations.

Fig. 8 and Fig. 9 illustrate the impact of different optimizations on performance. For this, the baseband algorithms were profiled on the vector VLIW architecture while gradually enabling optimizations in the algorithms. The improvements

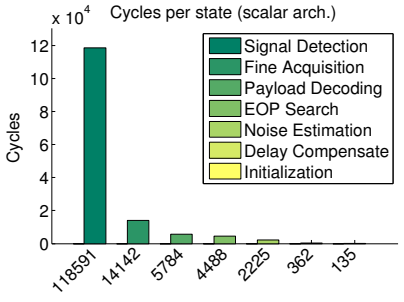

Fig. 6. Cycles per algorithm on scalar architecture

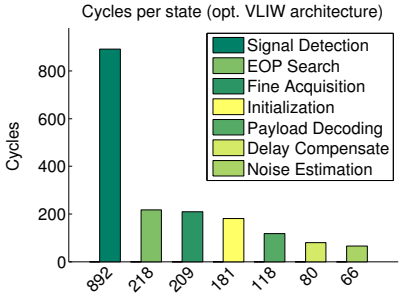

Fig. 7. Cycles per algorithm on vector VLIW architecture

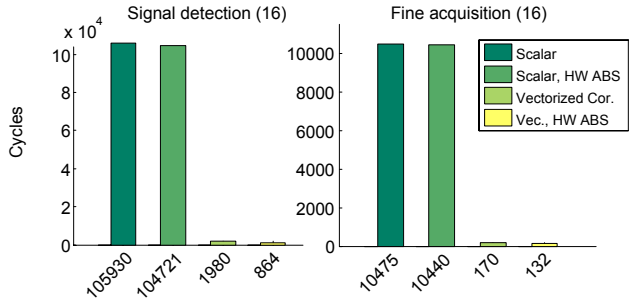

Fig. 8. Degrees of performance optimization for synchronization (vector VLIW architecture)

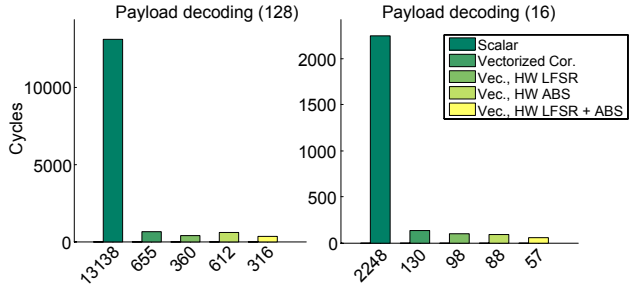

Fig. 9. Degrees of performance optimization for payload decoding (vector VLIW architecture)

due to vectorization and enabling different custom instructions, i.e. vector correlation, addition of absolute values and LFSR emulation (see Section IV-B), are shown as compared to a scalar implementation. For the signal detection and fine acquisition algorithms cycle count reductions by factors of 122x and 79x can be achieved. For the payload decoding algorithm the combined use of all architectural improvements enables a reduction by factors of 39x (spreading code length 16) and 41x (SC length 128).

It is observed that the signal detection during synchronization is the most cycle-intensive state, hence determining the required clock frequency. Only due to all architectural optimizations, real-time processing of baseband data at a relatively moderate clocking frequency of 225 MHz is made feasible. At this frequency, a computational performance of 7.65 GOPS is achieved by the VLIW architecture as compared to 0.23 GOPS and 3.6 GOPS for scalar and vector architecture respectively.

B. Power and Energy Results

Out of active and idle power consumption as well as the execution time profiling information, energy values and average power consumption values per algorithm state are calculated.

Fig. 10 summarizes the average power values obtained for different baseband algorithms running on the final UWB baseband processor. It can be seen that during the execution of the signal detection (SD) and the EOP search algorithms for a

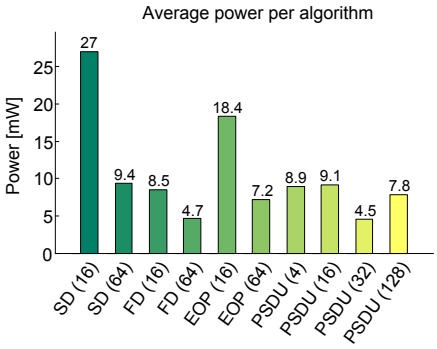

Fig. 10. Average power consumption of different baseband sub-algorithms (vector VLIW architecture)

spreading code length of 16, the average power consumption is quite high. This is due to the fact that in these phases the computational effort is high while the time span available for processing is very limited. For the fine acquisition (FA) and the decoding of the actual payload data (PD) on the other hand, the duty cycle is considerably smaller, resulting in a lower average power consumption.

Fig. 11 and Fig. 12 depict the decrease of energy consumption for different versions of the synchronization and the payload decoding algorithms. All algorithms are being profiled on the final vector VLIW baseband architecture but make use of different processor optimizations. The impact of vectorization as well as the benefit of enabling different custom

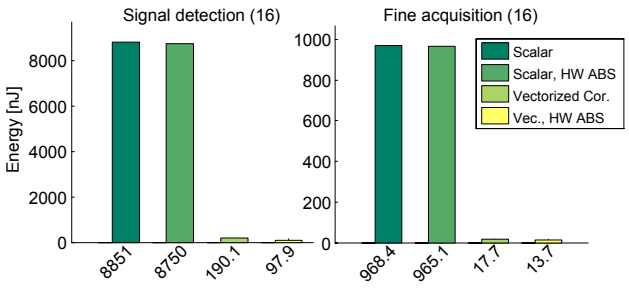

Fig. 11. Energy optimization results for synchronization (vector VLIW architecture)

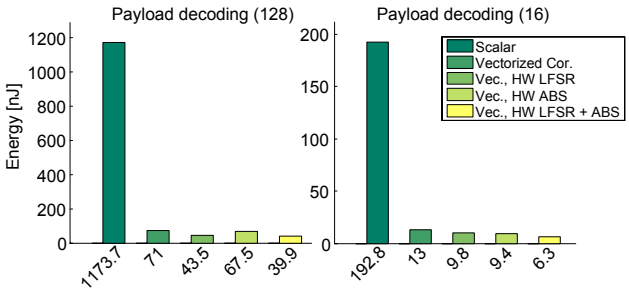

Fig. 12. Energy optimization results for payload decoding (vector VLIW architecture)

instructions, as introduced in Section IV-B, is shown. Due to the shorter run-times, energy consumption is reduced by a factor of 90x for signal detection algorithm and 70x for the fine delay search state. For the payload decoding, improvements by a factor of 30x for a spreading code length of 16 and a factor of 29x for a SC length of 128 are achieved as compared to the unoptimized version. These improvements enable energy-efficient baseband processing.

C. Area Results

The overall area of the final baseband processor (vector VLIW) after synthesis, place and route, is 0.77 mm^2. Program, scalar data and vector data memories contribute to almost 60% of this area. Large data memories are required to buffer incoming data of the radio front-end. A large part of the processor core itself is used by the vector register file (58%) that is especially needed for synchronization. The UWB IR functional units implementing the custom instructions as mentioned in Section IV-B only contribute to a small part (≈3%) of the core area.

VI. Conclusions

Novel wireless sensor networks require ultra-low power radios for data communications. The IEEE 802.15.4a ultra-wideband impulse radio amendment represents a promising physical layer for energy-efficient, low data rate communications. In this paper we present an application-specific processor architecture tailored to the needs of digital baseband processing.

Within the design process, various optimizations on different levels of abstraction are introduced. By exploiting these optimizations and migrating from a scalar to a vector and finally to a vector VLIW architecture, performance speed-ups in the range of 39x to 122x for payload decoding and signal detection algorithms are presented. Furthermore, energy consumption can be decreased by a maximum of 30x for payload decoding and 90x for signal detection as compared to the baseline implementation.

We conclude that an energy-efficient, low-power baseband ASIP implementation for IEEE 802.15.4a is feasible while still offering a greater amount of flexibility due to programmability than an ASIC.

Acknowledgement

We would like to thank Target Compiler Technologies and IMEC for their support in the course of this project.

References

[1] B. Gyselinckx, R. Vullers, C. Hoof, J. Ryckaert, R. Yazicioglu, P. Fiorini, and V. Leonov. Human++: Emerging Technology for Body Area Networks. *VLSI-SoC*, 2006.

[2] M. Badaroglu, C. Desset, J. Ryckaert, V. D. Heyn, G. V. der Plas, P. Wambacq, and B. V. Poucke. Analog-digital partitioning for low-power UWB impulse radios under CMOS scaling. *EURASIP*, 2006.

[3] J. Govers, J. Huisken, M. Berekovic, O. Rousseaux, F. Bouwens, M. D. Nil, and J. L. van Meerbergen. Implementation of an UWB Impulse-Radio Acquisition and Despreading Algorithm on a Low Power ASIP. *HiPEAC*, 2008.

[4] IEEE. *P802.15.4a Draft Amendment to IEEE Standard for Information technology - Telecommunications and information exchange between systems*, 2006.

[5] FCC. *Revision of Part 15 of the Commission's Rules Regarding Ultra-Wideband Transmission Systems*, 2002.

[6] G. Goossens, D. Lanneer, W. Geurts, and J. Van Praet. Design of ASIPs in multi-processor SoCs using the Chess/Checkers retargetable tool suite. *Intl. Symp. on System-on-Chip*, 2006.

[7] C. Desset, M. Badaroglu, J. Ryckaert, and B. van Poucke. UWB Search Strategies for Minimal-Length Preamble and a Low-Complexity Analog Receiver. *SPAWC*, 2006.

[8] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and S. Kaijian. *Low Power Methodology Manual*. Springer, 2007.

[9] M. Munch, B. Wurth, R. Mehra, J. Sproch, and N. Wehn. Automating RT-level operand isolation to minimize power consumption in datapaths. *DATE*, 2000.

[10] G. Goossens, J. Van Praet, D. Lanneer, and W. Geurts. Ultra-Low Power? Think Multi-ASIP SoC!. *IP-07*, 2007.

An Emulation-Based Real-Time Power Profiling Unit for Embedded Software

Andreas Genser[1], Christian Bachmann[1], Josef Haid[2], Christian Steger[1] and Reinhold Weiss[1]
[1]Institute for Technical Informatics, Graz University of Technology, Austria
[2]Infineon Technologies Austria AG, Design Center Graz, Austria
{andreas.genser, christian.bachmann, steger, rweiss}@tugraz.at
josef.haid@infineon.com

***Abstract*—The power consumption of battery-powered and energy-scavenging devices has become a major design metric for embedded systems. Increasingly complex software applications as well as rising demands in operating times while having restricted power budgets make power-aware system design indispensable. In this paper we present an emulation-based power profiling approach allowing for real-time power analysis of embedded systems. Power saving potential as well as power-critical events can be identified in much less time compared to power simulations. Hence, the designer can take countermeasures already in early design stages, which enhances development efficiency and decreases time-to-market. Accuracies achieved for a deep sub-micron smart-card controller are greater than 90% compared to gate-level simulations.**

I. Introduction

Rising complexity of embedded software applications and the advance in processing power available in embedded systems require power analysis techniques to identify power saving potential. Furthermore, the detection of power-critical events, such as power peaks, which can affect system stability of energy-scavenging devices (e.g. contact-less smart-cards) is of great importance.

Among all abstraction layers the greatest power reduction potential can be identified on the application layer [1]. To enable the design of power-efficient software applications, power consumption feedback to the software designer should be available already at early design stages. However, commercially available power estimation and analysis tools are often operating on low abstraction layers, which are usually not available to the software designer. Moreover, low-level power simulations lead to extensive run-times. This makes power simulations for complex designs unfeasible.

Functional hardware emulation by means of prototyping platforms, such as FPGA-boards, has become a widespread technique for functional verification. Power information, however, is still in many cases gathered by power simulators. In this work, which is part of the PowerHouse[3] project, we propose an emulation-based real-time power profiling approach to circumvent this limitation. A given design augmented with power estimation hardware allows for obtaining power alongside functional characteristics in real-time. Power saving potential or power peaks can hence be detected earlier in the design cycle, which normally is not feasible before the design is available in silicon and actual physical measurements can be carried out.

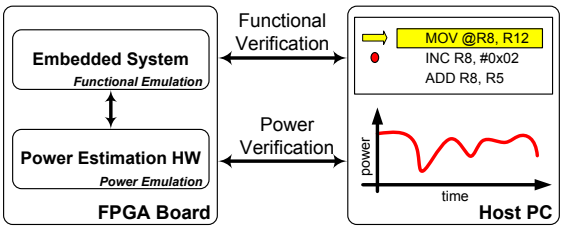

Fig. 1. Overview of an emulation-based power profiling approach for embedded systems comprising host computer interaction and visualization

By coupling this approach with a software development environment, valuable power information can be transfered to the software designer. This concept is depicted in Fig. 1. The FPGA-platform collects functional verification and power characteristics information, which is transmitted to a host computer. These information can be evaluated and visualized in a software development environment.

This paper is structured as follows. Section II provides information on previous work on power profiling. Section III briefly shows our research contributions. In Section IV the design of the real-time power profiling unit is discussed. Section V outlines a case-study applying the concepts developed in this work to a contact-less deep sub-micron smart-card controller and finally, conclusions drawn from the current work are summarized in Section VI.

II. Related Work on Power Profiling

Embedded software application power profiling can be categorized in (i) *measurement-based* and (ii) *estimation-based* methods.

Measurement-based methods are performed by taking actual physical measurements. This yields high accuracy compared to other approaches but requires additional measurement-equipment.

In contrast, power profiling by means of estimation methods is often based on power modelling. These techniques are usually less accurate but provide greater flexibility, since

[3]Project partners are Infineon Technologies Austria AG, Austria Card GmbH and TU Graz. The project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT contract FFG 815193.

also power consumption for sub-modules of the system can be derived. In the following we compare ongoing research activities in the field of power profiling.

A. Measurement-Based Methods

In [2], PowerScope an energy profiling tool for mobile applications is introduced. The system's current consumption is automatically measured during run-time by a digital multimeter. Measurement data are collected for later analysis on a host computer.

An oscilloscope measurement-based profiling technique is proposed by Texas Instruments in [3]. The current drawn by a DSP system is profiled and results are visualized on a host computer in TI's software development environment.

B. Estimation-Based Methods

Power profiling by means of estimation techniques can be subdivided into (i) *simulation-based* and (ii) *hardware-accelerated* approaches.

Simulation-based power estimation executes programs on simulators to obtain circuit activity information. Power values are acquired using these information. In hardware-accelerated power estimation approaches, power information is derived from power models, which are implemented in hardware.

Estimation techniques can be employed on various levels of abstraction resulting in different estimation accuracies. Moreover, the degree of abstraction influences simulation times for simulation-based approaches and hardware-effort for hardware-accelerated methods. Real-time power estimation, however, is limited to hardware-accelerated estimation techniques.

Commercially available power estimation tools (e.g. [4]) operate on low abstraction levels, such as gate- or register-transfer level (RTL). Achievable estimation accuracies are high, while extensive simulation times render power estimation of elaborate applications unfeasible. On top of this, low-level simulators are often not available to software designers. Therefore, attempts to estimate the system's power consumption on a higher level of abstraction are carried out.

A *simulation-based* approach employing power models for instruction-level power estimations is proposed by Tiwari et al. in [5]. It allows for power and energy consumption estimation for given applications. The underlying power model considers the power consumption during instruction execution (i.e. base costs) and power consumption during the transition between instructions (i.e. circuit state overhead costs). In [6], Sami et al. consider additional microarchitectural effects to enhance the accuracy of instruction-level power estimation based on a pipeline-aware power model for Very-Long-Instruction-Word (VLIW) architectures. A co-simulation based power estimation technique is introduced by Lajolo et al. in [7]. This approach for System-On-Chips (SoCs) works on multiple abstraction levels. In principle, power estimation is performed on system level, while for refinement purposes and accuracy enhancements various components are co-simulated on lower levels of abstraction. Countermeasures against high simulation times are caching, statistical sampling and macro-modelling. A simulation framework for system-level SoC power estimation is introduced by Lee in [8]. This approach is based on power models developed for the processor, memories and custom IP blocks. Power values derived are provided cycle-accurately to the designer in a dedicated profile-viewer.

Hardware-accelerated power estimation techniques are performed by augmenting the given system with dedicated hardware blocks. A power characterization process performed beforehand determines power values, which are mapped towards corresponding power states. For example, hardware events (e.g. CPU idle/run states, memory read/write states, etc.) are representatives of such power states. For energy accounting, existing power estimation hardware can be extended to power state counters. In [9], Bellosa gathers information by means of hardware event counters to derive thread-specific energy information for operating systems. Joseph et al. obtain the power consumption of a system by exploiting existing hardware performance counters of a microcontroller [10]. A power macro-model based coprocessor approach for energy accounting is proposed in [11]. Energy events identified by energy sensors are tracked by a central controller. The additional power estimation hardware requires extra chip area but yields also a speed-up compared to simulation-based approaches.

Power emulation represents a special case of hardware-accelerated power estimation. FPGA-boards can be used as a typical prototyping platform to emulate not only the functional system behavior but also its power consumption. A given system comprising power estimation hardware is mapped onto an FPGA-platform. Functional verification and power estimation can be performed in real-time even before the silicon implementation of the system is available.

An overview of the power emulation principle is presented in [12]. Run-time improvements by power estimation hardware-acceleration of about 10x to 500x compared to commercial power estimation tools are achieved. Strategies to minimize the hardware overhead introduced by power estimation are proposed. In [13], this approach is extended to a hybrid power estimation methodology for complex SoCs. This framework combines simulation and emulation techniques, which significantly reduce power analysis times. In [14], the power consumption of processor cores is estimated employing power emulation to guide process migration between cores.

III. Contributions

Power profiling by means of physical measurements is typically very coarse-grained and limited to the entire chip due to chip integration and packaging. Moreover, the final chip is not available at early design stages.

Simulation-based power profiling techniques can be employed at the beginning of the design cycle. However, they are rendered unfeasible for complex applications due to extensive simulation times. To encourage the software designer to consider power aspects at early design stages, we provide a real-time emulation-based power profiling approach. Power

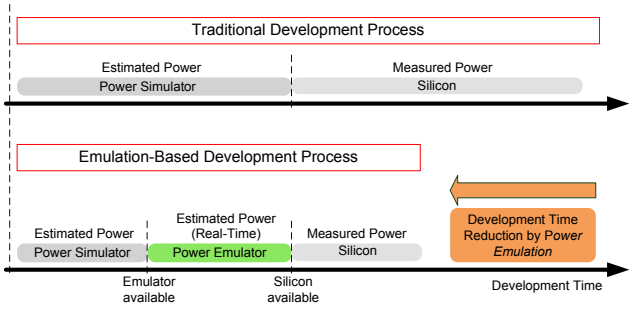

Fig. 2. Emulation-based vs. traditional power profiling approach

information is delivered to the software designer before silicon is available by utilizing an FPGA prototyping platform comprising power estimation hardware. Expensive redesigns caused by 'power bugs' can be avoided, which helps to decrease time-to-market (see Fig. 2).

The main goals of this work can be defined as follows:

- Deliver power information to the designer at early design stages to allow for:
  - Power-efficient software application design
  - Power-critical event detection
- Reduce development times

IV. Design of a Real-Time Power Profiling Unit

Estimation-based power profiling methods derive power information by exploiting power models. The abstraction layer on which these models are set up determines complexity and accuracy. Low-level models established on transistor- or gate-level are complex and therefore not suitable for emulation-based power profiling. In contrast, on a higher abstraction layer only main system components (e.g. CPU, memory, coprocessor, etc.) are taken into account. This leads to more compact models, hence real-time power profiling with moderate area increases is only feasible following this approach.

A. Power Model

Power models on a high level of abstraction are often based on linear regression methods. Details can be obtained in [15] and implementations are discussed in [5], [16]. A linear regression model can be given as

$$y = \sum_{i=0}^{n-1} c_i x_i + \epsilon. \tag{1}$$

$\mathbf{x} = [x_1, x_2, ...x_{n-1}]$ gives the vector of model parameters. x_i represent system states, such as CPU modes (e.g. idle, run) or memory accesses (e.g. read, write) etc. The vector of model coefficients is given as $\mathbf{c} = [c_1, c_2, ...c_{n-1}]^T$. Each model coefficient c_i contains power information and has to be determined during a preliminary power model characterization process. The linear combination of model parameters $\mathbf{x}$ and model coefficients $\mathbf{c}$ form the power estimate y. The deviation between the real power value and its estimate y is stated by ϵ (i.e. the estimation error).

B. Power Characterization Process

Typically a linear regression model can be designed in three major steps.

(i) Selection of model parameters. The choice of model parameters directly influences the model's accuracy and is therefore of great importance. In addition, the cross-correlation between model parameters reflects the amount of redundancy in the model. This metric helps to keep a model as small as possible and thus efficient.

(ii) Selection of the training-set. The training-set is based on m power measurements for a number of m vectors $\mathbf{x^i}$, each of which containing n model parameters

$$\mathbf{x^i} = [x_0^i, x_1^i, ...x_{n-1}^i] \text{ for } 0 \leq i \leq m-1. \tag{2}$$

Vectors $\mathbf{x^i}$ can be combined to the matrix

$$\mathbf{X} = [\mathbf{x^0}, \mathbf{x^1}, ...\mathbf{x^{m-1}}]^\mathbf{T}. \tag{3}$$

Power values y acquired for corresponding vectors $\mathbf{x^i}$ can be expressed as

$$\mathbf{y} = [y_0, y_1, ...y_{m-1}]^T. \tag{4}$$

Finally, $\mathbf{X}$ and $\mathbf{y}$ define the training-set and can be given as the tuple $\mathbf{T}$ in (5).

$$\mathbf{T} = (\mathbf{y}, \mathbf{X}) \tag{5}$$

The linear regression model given in (1) can also be written in matrix-form. This is depicted in (6).

$$\mathbf{y} = \mathbf{Xc} \tag{6}$$

Vectors $\mathbf{x^i}$ in $\mathbf{X}$ are derived from test applications (benchmarks) on a given embedded system and corresponding power values $\mathbf{y}$ are determined by physical power measurements or gate-level simulations.

(iii) Least squares fit method. The number of elements in the training-set $\mathbf{T}$ is usually much higher than the number of model parameters $\mathbf{c}$. This implies that the number of rows in $\mathbf{y}$ and the number of columns in $\mathbf{X}$ are higher than actually required to solve the linear system of equations in (6). Hence, the system is overdetermined and no exact solution exists. To overcome this issue model parameters are determined while minimizing the square error by using the least squares fit method.

The above steps can be carried out iteratively. If the model's accuracy does not meet the requirements, a model refinement by accounting more low-level information can be applied.

C. Power Emulation Architecture

The power emulation (PE) architecture that integrates the power model in hardware is illustrated in Fig. 3.

Power sensors are employed to track state information of system modules. For accuracy purposes also lower abstraction layer information can be considered (e.g. state information of functional units of the CPU). State vector and power

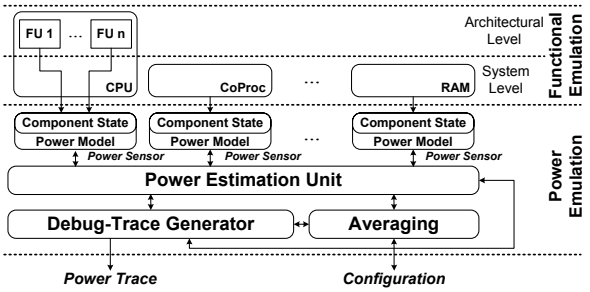

Fig. 3. Power emulation architecture

information are stored in a software-configurable table. These state information is mapped towards power values using a table-lookup approach. Fig. 4 depicts the principle structure of a power sensor module.

Each of a number of k power sensors covers l system states and contributes to the entire power model as expressed in (7). The PE-architecture delivers power information each cycle, hence time-dependency t is introduced in the following equations to account for power values estimated at different points in time.

$$y_{j,i}(t) = c_i \; x_i(t) \text{ for } 0 \leq i \leq l-1 \;\wedge\; 0 \leq j \leq k-1 \quad (7)$$

16-bit registers are provided to configure the power sensors with the power coefficients information obtained from $\mathbf{c}$. It is worth noting that this power table can also be reconfigured during program run-time. This enables the masking of system modules, allowing the tracking of the power consumption of single sub-modules.

The power estimation unit accumulates 16-bit power sensor outputs according to (8). This constitutes an instantaneous, cycle-accurate up to 32-bit wide power estimate $y(t)$ for the overall system. The entirety of power sensors comprising the power estimation unit represent the power model established in hardware (see equality in (8)).

$$y(t) = \sum_{j=0}^{k-1} \sum_{i=0}^{l-1} y_{j,i}(t) = \sum_{i=0}^{n-1} c_i \; x_i(t) \quad (8)$$

Further post-processing is applied by the averaging module, which allows for smoothing and de-noising of a sequence of power values. This is enabled by a configurable moving average filter as shown in (9). Filtering properties can be changed by adjusting N.

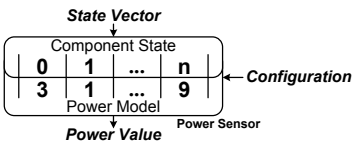

Fig. 4. Power sensor

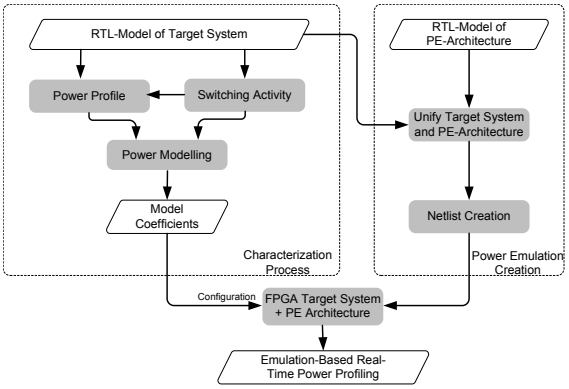

Fig. 5. Design flow for emulation-based real-time power profiling

$$y_{avg} = \frac{1}{N} \sum_{j=0}^{N-1} y(t-j) \quad (9)$$

The debug-trace generator unit captures power information of the power estimation unit and the averaging module. A debug-trace message is composed out of these data and is delivered to the host computer for evaluation and further processing.

D. Design Flow

Fig. 5 outlines the design flow of the emulation-based real-time power profiling approach. A synthesizeable RTL-model of the target system is provided to perform the characterization process. After synthesis gate-level simulations based on benchmarking applications are performed and activity information as well as power profiles are acquired from value change dump (VCD) files. These information is fed to a power modelling process, deriving power model coefficients.

The target system RTL-models and the PE-architecture are merged to allow the generation of a single netlist. After downloading the netlist onto the FPGA-platform, power model coefficients determined beforehand are used to configure the power sensors for tailoring the PE-architecture to the given target system. Now applications of interest can be executed and real-time power profiles can be obtained.

E. System Set-Up

Power model coefficients obtained during the characterization process deliver configuration data for the power sensors. Listing 1 illustrates how to configure power sensors to tailor them to the power consumption of system modules. 16-bit registers are provided for this purpose.

```
// start of program

//configuration of power sensor1
PWRSEN0_STATE0 = 0x005A;  //CPU run mode
PWRSEN0_STATE1 = 0x0011;  //CPU halt mode
PWRSEN0_STATE2 = 0x0013;  //CPU sleep mode

//configuration of power sensor2
PWRSEN1_STATE0 = 0x0013;  //memory read
```

```
PWRSEN1_STATE1 = 0x001A;   //memory write
...
activate_power_emulation();
start_main_program();
```

Listing 1. Power emulation set-up, power sensor configuration

A variable number of system states for a variable number of system modules can be configured with power coefficients. Finally, power profiling is activated before normal application execution starts. The run-time overhead introduced due to power sensor configuration is negligible compared to the number of cycles executed by a typical application. A debug-trace containing power information is automatically generated and transfered to the host computer for power information evaluation and profile visualization without the interaction of the software designer.

If power analysis of sub-modules of the system is desired, the power sensor attached to the module of interest is configured as usual. The configuration of the remaining modules is skipped, so they do not contribute to the overall power consumption.

V. Case Study: Profiling of Power-Critical Smart-Card Applications

Smart-card applications have been penetrating manifold market segments in the last years. Access control, electronic passport or payment are only a few out of many existing applications.

Smart-cards in general can be categorized in (i) *contact-based* and (ii) *contact-less* derivatives. Contact-based smart-cards are powered if inserted into a reader device, while contact-less systems consume power via an RF-field generated by the reader. Therefore, contact-less devices are subjected to stringent power limitations.

Fig. 6 illustrates the coupling of the reader device with the contact-less smart-card by a magnetic field H. A certain amount of power is transfered from the reader device to the smart-card at a time. The available power is limited, hence exceeding a maximum power limit due to power-peaks affects system stability and can cause malfunctions. This case-study demonstrates the capability of our emulation-based power profiling approach to support the software designer early in the development process to avoid such worst-case scenarios.

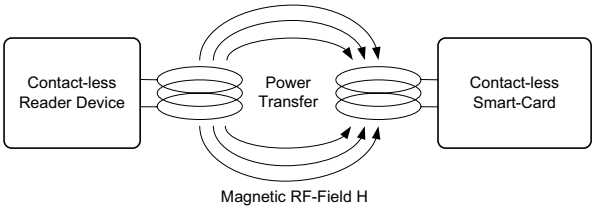

Fig. 6. Power supply of a contact-less smart card by a magnetic field generated by a reader device

A. Smart-Card Architecture Overview

Fig. 7 depicts a typical contact-less smart-card system. It is based on a 16-bit pipelined cache architecture comprising volatile and non-volatile memories. A symmetric coprocessor (SCP) is included for Advanced Encryption Standard (AES) and Data Encryption Standard (DES) algorithm acceleration. Moreover peripherals, such as a UART for communication purposes, timers or a random number generator (RNG) are provided. System modules are powered by an externally generated RF-field. Energy is collected by an antenna system and power supply conditioning and stabilization by means of an analog front-end are carried out.

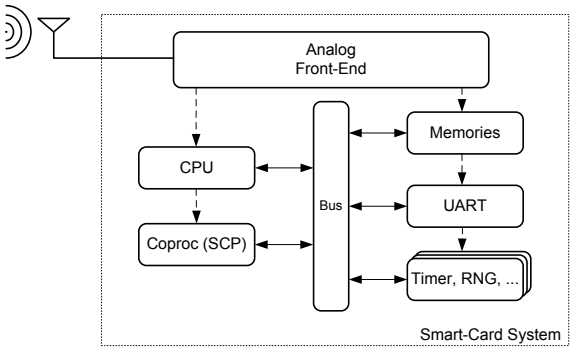

Fig. 7. Block diagram of a typical contact-less smart-card system

System modules that are major contributors to the overall power consumption are identified during a power characterization process. Moreover, available operating modes of each system module influencing the amount of power consumed are considered. A number of benchmarking applications to test many of these operating modes were applied. Based on the result of the characterization process, power model coefficients were obtained to configure the power sensors as shown in Listing 1. Table I summarizes system modules and corresponding operating modes relevant for the power model.

TABLE I
Operating modes of typical smart-card components considered in the power model

Unit	Mode(s)
CPU	run, halt, sleep
Cache	read, write (hit, miss)
Memories	read, write
UART	read, write
Peripherals	on, off
SCP	Encryption: AES128/192/256, (Single-, Double-, Triple-) DES
SCP	Decryption: AES128/192/256 (Single-, Double-, Triple-) DES

B. Payment Application Profile Analysis

A typical application for smart-cards incorporating a symmetric cryptographic coprocessor is payment. Payment applications usually contain authentication procedures requiring

cryptographic operations. Fig. 8 illustrates a typical power profile of a future payment application obtained with the emulation-based power profiling approach. The first power peak marks the power consumption of an AES computation, while the remaining slightly smaller and shorter power peaks result from DES computations.

We assume that the maximum available power provided by the RF-field is 0.9 as shown in Fig. 8 (Note that power values are normalized). Hence, the payment authentication process would fail due to power peaks caused by the SCP.

If the source of these power peaks is not obvious to the designer already at this step, the power profile can be decomposed into sub-modules by reconfiguring power sensors. Fig. 9 depicts power profile results for the CPU, memories, SCP and UART decomposed for sub-module power profile analysis. As a reference also the cumulated power profile is shown.

The major contributor to the power consumption in this example is the CPU with more than 60%. Memories (including cache) and the SCP account for the remaining power consumption. The UART is inactive in this application and therefore consumes no power. It can clearly be seen that the SCP's power consumption causes the overall power profile to exceed the absolute maximum power of 0.9.

Various countermeasures could be taken to circumvent this issue. The AES algorithm could be implemented in software to avoid using the SCP. Another alternative is reducing the system clock frequency. Both solutions reduce the payment application's speed, but ensure reliable operation. Fig. 10 shows the power profile when scaling down the system frequency from 33 MHz to 28 MHz. Power peaks are below 0.9, hence system operation is stable.

C. Accuracy of Emulation-based Power Profiling

Fig. 11 shows power profiles of the payment application. A comparison between the emulation-based power profiling result and gate-level power simulation profiles obtained with *Magma Blastfusion 5.2.2* [17] are given. The relative error on average and the variance are 8.4% and 9.4%, respectively.

Accuracy considerations for other executed applications are summarized in Table II. The relative power error on average and the corresponding variance are given. Moreover, relative energy error values are depicted. For all tested applications relative power errors are less than 10% on average. Energy accounting reaches accuracies of greater than 93%.

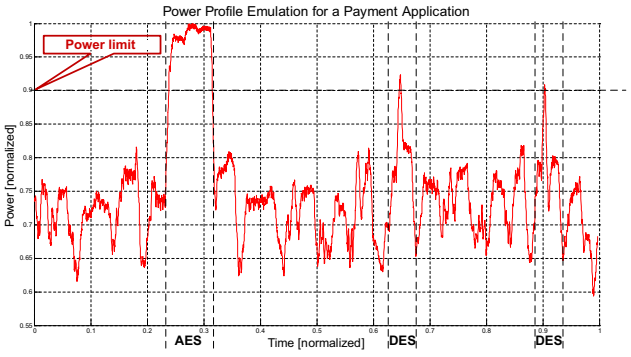

Fig. 8. Payment authentication application power profile obtained by emulation-based power profiling (power exceeds affordable level)

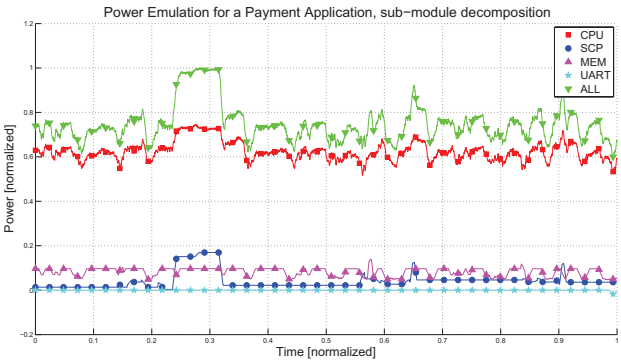

Fig. 9. Payment authentication application power profile decomposed for sub-module power analysis

TABLE II
POWER EMULATION ACCURACY COMPARISON FOR VARIOUS APPLICATIONS

Algorithm	Duration (μs)	Performance (Cycles)	Error (%) Power avg.	Power σ^2	Energy avg.
ALU1	70.8	2336	7.3	0.5	6.4
ALU2	44.2	1458	4.0	0.2	3.9
CPU	31.3	1032	-2.1	0.9	-3.2
Cache	12.4	4092	-1.5	0.9	-2.6
RAM	56	1848	-4.9	0.3	-5.5
SCP-AES128	13.5	4455	1.2	1.8	-0.2
SCP-AES256	15.7	5181	1.1	1.6	-0.2
SCP-DES	82.2	2712	1.1	0.8	0.3
SCP-DDES	76.9	2537	0.5	0.7	1.0
Payment	338	11160	8.4	9.4	2.0
Dhrystone	139	4587	0.4	2.0	-2.0

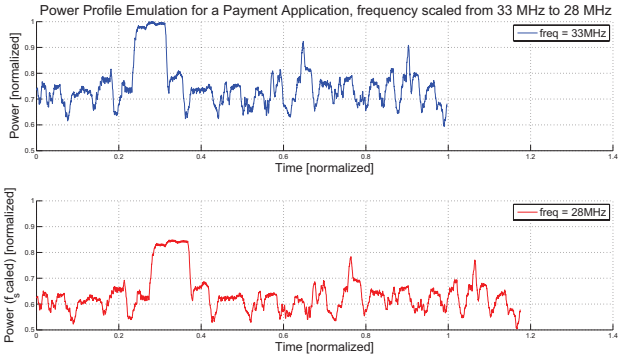

Fig. 10. Payment authentication application power profile obtained by emulation-based power profiling (stable system operation due to frequency-scaling)

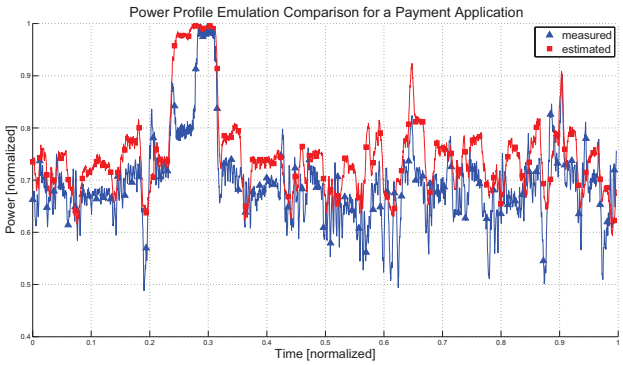

Fig. 11. Power profile comparison, gate-level power simulation profile vs. emulation-based power profile

D. Performance Evaluation

One of the major advantages of the emulation-based profiling approach is the capability to acquire power profiles in real-time. Power estimation takes no longer than application execution on the target-system. Hence, the power profile is available immediately after execution. Table III shows execution times of power profile simulations on a gate-level basis using *Magma Blastfusion* compared to the emulation-based approach. Extensively high simulation times are depicted compared to emulation run-times of a few hundreds of microseconds.

As illustrated in Table III, power emulation of the payment application takes 338 microseconds, whereas the power simulation is about 98 hours. Additional hardware introduced for power emulation contributes only to 1.5% to the overall system area.

It is obvious that the power simulation of complex applications is rendered unfeasible due to far too extensive simulation times. Therefore, power saving potential or power peaks leading to system failure as discussed in this section cannot be detected in an early design stage. By means of emulation-based power profiling, power estimates are available immediately and already when working with FPGA prototyping platforms. Countermeasures to circumvent power peaks causing system failures can be taken before the device is available on silicon and physical measurements are performed.

VI. Conclusion

Extensive run-times of power simulators render power analysis of increasingly complex embedded software applications unfeasible. The power profiling approach proposed in this work, delivers power information to the software designer in real-time. Moreover, by employing FPGA prototyping platforms, these power information is available already at early design stages. Emulation-based power profiling has proven to be an effective option to estimate a system's power consumption, delivering power information with accuracies of 90% on average. This paves the way for power-efficient embedded software design and the capability to cope with power critical events more efficiently.

TABLE III
Performance comparison of power profiling for a simulation and emulation approach

Algorithm	Performance (Cycles)	Duration Simulation[4] (hours)	Duration Emulation (microseconds)
ALU1	2336	4.1	70.8
ALU2	1458	2.2	44.2
CPU	1032	0.78	31.3
Cache	4092	14.0	12.4
RAM	1848	2.9	56
SCP-AES128	4455	17.2	13.5
SCP-AES256	5181	24.0	15.7
SCP-DES	2712	5.5	82.2
SCP-DDES	2537	6.3	76.9
Payment	11160	98.3	338
Dhrystone	4587	18.1	139

[4] Simulation is performed at a sampling rate of 33 MHz, which corresponds to the clock frequency of the smart-card system. The server system for simulations comprises 12 CPUs and 50 gigabytes of physical memory for user processes.

References

[1] E. Macii and M. Poncino, *Power Macro-Models for High-Level Power Estimation*. CRC Press, edited by C. Piguet, 2005, ch. 39 Low-Power Electronics Design, pp. 39–1—39–18.
[2] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *WMCSA*, 1999, pp. 2–10.
[3] *Analyzing Target System Energy Consumption in Code Composer Studio^{TM} IDE*, Texas Instruments, 2002.
[4] J. Flynn and B. Waldo, "Power Management in Complex SoC Design," Synopsys Inc. White Paper, Tech. Rep., 2005.
[5] V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis Of Embedded Software: A First Step Towards Software Power Minimization," in *ICCAD*, 1994, pp. 384–390.
[6] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "An instruction-level energy model for embedded VLIW architectures," in *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 21, 2002, pp. 998–1010.
[7] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Cosimulation-based power estimation for system-on-chip design," in *IEEE Trans. on Very Large Scale Integration Systems*, vol. 10, 2002, pp. 253–266.
[8] I. Lee, H. Kim, P. Yang, S. Yoo, E. Chung, K. Choi, J. Kong, and S. Eo, "PowerViP: SoC Power Estimation Framework at Transaction Level," in *ASP-DAC*, 2006, pp. 551–558.
[9] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *SIGOPS European Workshop*, 2000, pp. 37–42.
[10] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *ISLPED*, 2001, pp. 135–140.
[11] J. Haid, G. Kaefer, C. Steger, and R. Weiss, "Run-time energy estimation in system-on-a-chip designs," in *ASP-DAC*, 2003, pp. 595–599.
[12] J. Coburn, S. Ravi, and A. Raghunathan, "Power emulation: a new paradigm for power estimation," in *DAC*, 2005, pp. 700–705.
[13] M. Ghodrat, K. Lahiri, and A. Raghunathan, "Accelerating System-on-Chip Power Analysis Using Hybrid Power Estimation," in *DAC*, 2007, pp. 883–886.
[14] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation," in *ISLPED*, 2008, pp. 335–340.
[15] A. Bogliolo, L. Benini, and G. D. Micheli, "Regression-based RTL power modeling," in *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, 2000, pp. 337–372.
[16] C. Krintz and S. Gurun, "A run-time, feedback-based energy estimation model for embedded devices," in *CODES+ISSS*, 2006, pp. 28–33.
[17] "Magma Design Automation Inc., Blastfusion," (http://www.magma-da.com/), April 2009.

Automated Power Characterization for Run-Time Power Emulation of SoC Designs

Christian Bachmann*, Andreas Genser*, Christian Steger*, Reinhold Weiss* and Josef Haid†
*Institute for Technical Informatics, Graz University of Technology, Austria
†Infineon Technologies Austria AG, Design Center Graz, Austria
{andreas.genser, christian.bachmann, steger, rweiss}@tugraz.at
josef.haid@infineon.com

***Abstract*—With the advent of increasingly complex systems, the use of traditional power estimation approaches is rendered infeasible due to extensive simulation times. Hardware accelerated *power emulation* techniques, performing power estimation as a by-product of functional emulation, are a promising solution to this problem. However, only little attention has been awarded so far to the problem of devising a generic methodology capable of automatically enabling the power emulation of a given system-under-test. In this paper, we propose an automated power characterization and modeling methodology for high-level power emulation. Our methodology automatically extracts relevant model parameters from training set data and generates an according power model. Furthermore, we investigate the automation of the power model hardware implementation and the automated integration into the overall system's HDL description. For a smart card controller test-system the automatically created power model reduces the average estimation error from 11.78% to 4.71% as compared to a manually optimized one.**

I. Introduction

Power consumption has become a major design metric for electronic systems and mobile devices in particular. For these devices the power consumption severely affects operating time as well as system stability. In order to verify power requirements during the design phase of a given system, power estimation has become an essential part of the design process.

Due to the advances in process technology scaling as well as rising demands for computational performance and functionality, increasingly complex designs have to be handled in the power estimation process. Systems-on-chips (SoC) are typically composed of a large number of sub-components, each contributing to the overall power consumption. Furthermore, it can be observed that the power consumption of these devices is progressively more dependent on software applications, determining the utilization of system components as well as actuating available on-chip power management features. It is therefore favorable, to provide power estimation resources not only to system architects and hardware designers but also to power-aware software application and operating system developers.

For estimating the system's power consumption while executing elaborate program sequences (e.g., the booting sequence of an operating system), state-of-the-art gate- and RTL-level power simulators require extensive simulation times. Higher-level simulators, such as behavioral- or system-level tools, however fail at delivering cycle-accurate power estimates required for, e.g., evaluating the reaction of power management algorithms on power transients. This curtails the usability of these tools in power-aware software application development.

For this reason, hardware-accelerated power estimation approaches have been introduced, employing existing hardware counters [1–3], dedicated power estimation coprocessors [4, 5] and emulation-based approaches (i.e., *power emulation*) [6–10]. For the purpose of fast yet accurate software power estimation, high-level power emulation approaches [9, 10] as depicted in Figure 1 are considered promising. While requiring only small hardware overhead on a given FPGA already employed for functional emulation, they achieve a considerable power estimation speed-up compared to simulation-based methods. However, the time-consuming task of power model generation and required hardware description language (HDL) model adaptation for these power emulation approaches has, to the best of our knowledge, so far not been further investigated.

In the context of high-level power emulation, the novel contributions of this paper are as follows:

- We propose a systematic, automated power characterization and modeling methodology that automatically determines power model parameters for a given system-under-test.
- We develop an automated technique for implementing this power model in hardware and for integrating this generated power emulation hardware into the system-under-test.
- A case study on a smart card microcontroller test-system illustrates the benefits of our automated approach.

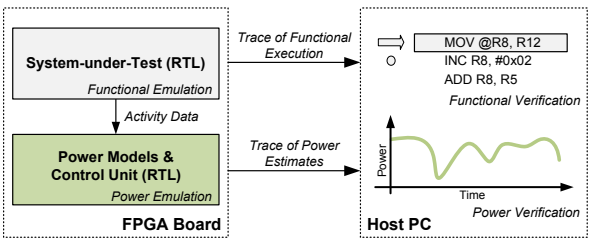

Fig. 1. Power emulation principle: Concurrent functional and power emulation (adapted from [10])

II. Related Work

Various hardware-accelerated power estimation techniques have been explored, leveraging existing hardware event counters [1–3], dedicated coprocessors [4, 5] and emulation-based approaches [6–10].

By correlating hardware events and power consumption, initial works have been using existing hardware event counters for the purpose of power estimation. The feasibility of this approach has been shown for commercially available desktop processors [1, 2] as well as for embedded processors [3].

Especially for enabling dynamic power management in mobile embedded systems, dedicated energy and power estimating coprocessors have been introduced [4, 5]. For estimating the power consumption, event-based energy and power macromodels are utilized in these works.

By augmenting the HDL model of a given system with dedicated power estimation hardware, *power emulation* can be performed as a by-product of functional emulation. Power emulation approaches using register transfer level (RTL) macromodels [6] and architectural-level component event-based power models [9] have been presented. In [8] a hybrid simulation- and emulation-based approach is introduced. This cosimulation approach can be used to simulate non-synthesizable parts of the overall system.

While RTL power emulation approaches achieve high estimation accuracy (mean error 3.4% as compared to RTL), they also suffer high area overhead (on average 3.1 times the area of the original design [6]). Furthermore, the additionally required logic negatively influences the maximum reachable operating frequency of the design. A high-level power emulation approach as presented in [9], circumvents this limitation due the use of a simplified power model. The estimation error in comparison to gate-level simulations is reported to be below 10% while requiring less than 3% of the FPGA LUTs. However, the power model proposed in [9] targets a specific implementation of a LEON3 chip multiprocessor (CMP) system introduced by the authors and is not a generic methodology.

In recent work, we have introduced our initial high-level power emulation hardware unit [10] as well as the use of this high-level power emulation methodology in power-aware software development [11]. Based on these works, we introduce and evaluate our automated power modeling and HDL implementation methodology in the following sections.

III. High-Level Power Emulation

The principle of power emulation, as initially established in [6], is based on augmenting the emulated system with special power estimation hardware. Power estimates are generated as a by-product of functional emulation during the run-time of the system. The traces of execution for both, functional and power emulation, can then be analyzed on a host computer as depicted in Figure 1.

A. Power Model

Our power model is based on the assumption that the power consumption of a given system and its sub-components can be expressed by *power states* at given points in time [12]. Furthermore we assume that for a given system-under-test, interface as well as internal signals down to a given hierarchical level are observable. This assumption is valid due to the fact that the HDL model of the system is required for synthesizing and mapping it onto the FPGA for functional emulation in the first place. Based on these assumptions, our power macromodel tries to map a number N of state- and power-relevant signals x_i to a power consumption estimate $\hat{P}$.

The total power consumption P of the system can be given as $P = P_{dyn} + P_{sta}$, where P_{dyn} resembles the well-known equation of dynamic power consumption in CMOS and P_{sta} accounts for static power consumption and leakage power. For $P_{dyn} = \alpha \cdot (f \cdot C \cdot V^2)$, α expresses the activity of a given component and the term $(f \cdot C \cdot V^2)$ captures clock frequency f, the amount of capacitance C being switched and the supply voltage V.

We model the relationship of state-dependent activity and the power consumption estimate as an additive linear equation in the form of

$$\hat{P} = c_0 + \sum_{i=i}^{N} c_i x_i = c_0 + c_1 x_1 + \ldots + c_N x_N \qquad (1)$$

where the coefficient c_0 models sources of static power consumption P_{sta} such as analog components and leakage. The coefficients c_i $(i = 1 \ldots N)$ express the non-activity-dependent term of CMOS power consumption $(f \cdot C \cdot V^2)$. The activity factor α of a given component is determined by the according state signal x_i. We can also express Equation 1 in the vector form $\hat{P} = \mathbf{x}\mathbf{c}^T$, where $\mathbf{x} = [1, x_1, \ldots, x_N]$ and $\mathbf{c} = [c_0, c_1, \ldots, c_n]$. Note that this model, while appropriately simple for implementing it in power emulation hardware, is sufficient to capture the power consumption dynamics of our system-under-test as shown in Section VI.

The problem that arises from this high-level power macromodeling approach is twofold: 1) The adequate selection of model parameters x_i, i.e., the identification of power-relevant state signals for a given system-under-test and 2) the fitting of model coefficients c_i for the selected parameters.

B. Power Emulation Unit

The power emulation (PE) unit monitors the power-relevant state signals x_i and derives cycle-accurate power estimates $\hat{P}$ from these data. It contains the hardware implementation of the power model as introduced in Equation 1. The architecture of our initial power emulation unit [10], as depicted in Figure 2, is used as the foundation for this work.

The PE unit consists of a number of power sensors, monitoring the state and activity of various system components. Each power sensor maps the observed state signals to a corresponding power value for the given module using a look-up table approach. The power estimator itself accumulates the

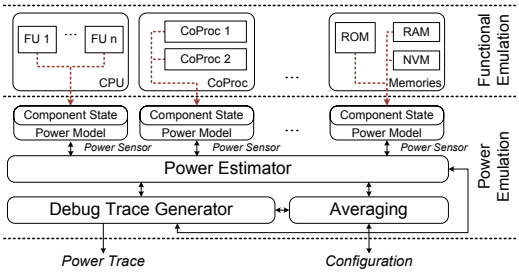

Fig. 2. Basic architecture of the power emulation unit monitoring system components and deriving power estimates according to their states (adapted from [10])

values generated by all power sensors and calculates the cycle-accurate overall power consumption estimate $\hat{P}$ of the system.

The power estimates are collected by a debug trace generation unit that assembles trace messages and delivers them to a host computer. Additionally, a reconfigurable moving average filtering module, serving the purpose of smoothing and denoising of power traces, is present in the PE unit.

While the initial power emulation unit presented in [10] resembles the manually created prototype used as the proof-of-concept for our high-level power emulation concept, in this paper[1] we present an automation approach that covers the two main obstacles for the power emulation unit's hardware implementation: 1) The automated adaptation of the system-under-test's existing HDL model for the purpose of power emulation. Internal power-relevant state signals, originating at various levels of design hierarchy, have to be connected to the power emulation unit. 2) The automated adaptation of the power emulation unit itself to the automatically generated power model.

IV. Automated Power Characterization

The manual characterization and power modeling of a given system is a time-consuming and tedious task. Therefore we employ an automated characterization methodology, consisting of multiple stages as depicted in Figure 3.

Based on a set of microbenchmarks, covering all components of the system-under-test, state-of-the-art gate-level power estimation tools are used to estimate power profiles of the system's physical implementation. At later stages of design, i.e., after the first silicon implementation is available, additional measurement profiles can be used for a *joint simulation- and measurement-based characterization*. The next step in creating a power model is the automated *selection of model parameters*. Finally, for the parameters selected before, coefficients are determined by means of a *model coefficients fitting* process.

A. Joint Simulation- & Measurement-based Characterization

For the purpose of characterizing the system-under-test and constructing a power macromodel, training set data are required [13]. The training set data tuple $\mathbf{T} = (\mathbf{X}, \mathbf{p})$, consisting of an observed signal state matrix (SSM) $\mathbf{X}$ and an observed power vector $\mathbf{p}$, are used to construct the power model. By executing m microbenchmarking applications in RTL or gate-level simulations, we obtain the signal state matrix $\mathbf{X} = [\mathbf{X}_1, \ldots, \mathbf{X}_m]^T$. $\mathbf{X}_i = [\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,K}]^T$ is the signal state matrix for a given K-cycle benchmark, containing a signal state vector $\mathbf{x}_{i,k}$ for every clock cycle k. The power vector $\mathbf{p} = [\mathbf{p}_1, \ldots, \mathbf{p}_m]^T$, where similar to above for a k-cycle benchmark $\mathbf{p}_i = [P_{i,1}, \ldots, P_{i,k}]^T$, is typically generated either in gate-level power simulations or, during later design stages, through physical power consumption measurements. In our characterization methodology we seek to improve training set quality by offering the possibility to integrate physical measurements performed on the same microbenchmarks that are used in gate-level simulations.

We address the issue of merging simulation- and measurement-derived training sets in a twofold manner: First, marker operations that serve the goal of detecting beginning and end of a test are inserted in each microbenchmark. Second, we perform a cross-correlation between the two resulting power profiles to compensate their temporal mismatch due to the non-ideal triggering conditions of the physical measurement. Afterwards, the simulated profile as well as the delay-compensated measured profile are stored in a joint measurement database that is used in the subsequent model parameter selection and coefficient fitting steps.

B. Model Parameter Selection

The automated selection of adequate model parameters represents one of the key contributions of our work. Specifically we try to minimze the number N of used power model parameters so that $N \ll N_{all}$, i.e., the number of selected parameters is by far smaller than the number of all available parameters. A small number of parameters, and thus a small number of coefficients to be fitted, vastly reduces model fitting effort and improves model efficiency.

The power model parameter selection algorithm is composed of three main sub-algorithms, as described in Algorithm 1, that are performed for the entire set of microbenchmarks. In a first *signal name pattern matching* step, the large number

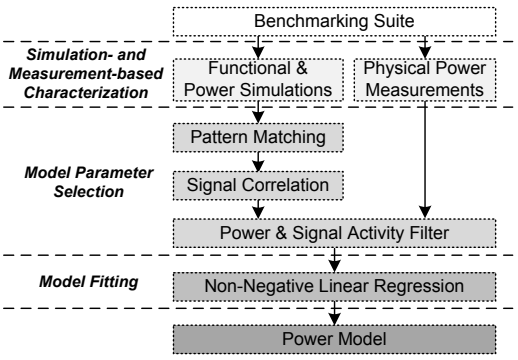

Fig. 3. Overview of the automated power characterization methodology

[1]This work is part of the POWERHOUSE project that is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT contract FFG 815193.

Algorithm 1: Power model parameter selection

Input: List of benchmarks L_{BM}, List of signals L_S, Training set data tuple **T** (containing transient power profiles for all benchmarks **p** and signal activity data for all benchmarks **X**), Signal name pattern list L_{npat}, Intra-signal correlation threshold Th_{cor}, Power activity threshold Th_{pcor}, Lag threshold Th_{lag}

Output: List of power model parameters L_{pms}

Step 1, signal name pattern matching:
List of candidate model parameters $L_{pms} := \{\}$
List of candidate signals $L_{cs} := \{\}$
foreach *Signal* $\boldsymbol{x}_i$ *in* L_S **do**
  if *Name of* $\boldsymbol{x}_i \in$ *name pattern list* L_{npat} **then**
    $L_{cs} := L_{cs} \cup \mathbf{x}_i$
foreach *Benchmark* b *in* L_{BM} **do**
  List of candidate signals per benchmark $L_c := L_{cs}$
  Step 2, intra-signal correlation:
  foreach *Signal* $\boldsymbol{x}_i$ *in* L_c **do**
    if *Signal activity* $\alpha(x_i)$ *= 0* **then**
      Remove signal $\mathbf{x}_i$ from L_c, continue;
    $\mathbf{R}_{\mathbf{X}_i} = \text{CorrCoef}(\mathbf{x}_i, \text{SSM for benchmark } \mathbf{X}_b)$
    if *Elements in* $|\boldsymbol{R}_{\boldsymbol{x}_i}| = 1$ **then**
      Remove same-cycle correlated signals from L_c except lowest hierarchical level one
    foreach *Signal* $\boldsymbol{x}_j$ *in* L_c,$i \neq j$ **do**
      if $\left|r_{(\boldsymbol{x}_i,\boldsymbol{x}_j)}(\tau)\right| > Th_{cor}$ *for* $0 \leq \tau \leq Th_{lag}$ **then**
        Remove delayed, correlated signals from L_c except lowest hierarchical level one
  Step 3, power - signal activity:
  foreach *Signal* $\boldsymbol{x}_i$ *in* L_c **do**
    if $\left|r_{(\Delta\boldsymbol{p},\Delta\boldsymbol{x}_i)}(\tau)\right| < Th_{pcorr}$ *for* $0 \leq \tau \leq Th_{lag}$ **then**
      Remove not power-related signals from L_c
  $L_{pms} := L_{pms} \cup L_c$

of potential model parameter signals x_i is largely reduced by only selecting signals matching a user-supplied list of certain name patterns. This list is typically dependent on the coding standard employed while creating the HDL implementation of the given system-under-test. A default name pattern list includes potential candidate signal name patterns, such as "enable", "read", "write", "busy", "ready", "wait", "halt", "sleep", etc., that are common to a large number of HDL designs. Note that by adapting this list of name patterns to the coding standard used while implementing the given system-under-test, the filtering result can be further narrowed down, reducing the run-time of the subsequent parameter selection algorithms.

The *intra-signal correlation analysis* represents the second sub-algorithm. It aims at analyzing signal activity statistics and removing redundant signals. The following main tasks are performed: 1) Removal of signals stuck at one logical level for the entire set of benchmarks, i.e., signals lacking any switching behavior. Note that a power benchmarking suite will also include tests varying otherwise mainly static signals, such as software configuration bits dictating the operating modes of the design. Thus, these mainly static but power-relevant configuration bits will not be eliminated by the intra-signal correlation analysis.

2) Calculation of the correlation coefficients matrix **R** for all N remaining signals **x** given as

$$\mathbf{R}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{C}(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{\mathbf{C}(\mathbf{x}_i, \mathbf{x}_i)\mathbf{C}(\mathbf{x}_j, \mathbf{x}_j)}}, \quad 1 \leq i, j \leq N \tag{2}$$

where **C** is the covariance matrix. For each row of the matrix **R** all columns containing high absolute correlation coefficient values resemble redundant signals, switching at similar points in time. A negative correlation coefficient indicates an inverted signal. These correlated signals are filtered out, only the signal originating at the highest hierarchical level is retained[2].
3) Calculation of the cross-correlation $r_{(\mathbf{x}_i,\mathbf{x}_j)}(\tau)$ of all the remaining signals within a specified maximum lag of Th_{lag}:

$$r_{(\mathbf{x}_i,\mathbf{x}_j)}(\tau) = \sum_{k=0}^{K-1} \mathbf{x}_i(k) \cdot \mathbf{x}_j(k+\tau), \quad 0 \leq \tau \leq Th_{lag} \tag{3}$$

Signals exhibiting a high maximum correlation within the maximum lag Th_{lag}, e.g., typically a small number of clock cycles, are also considered to be redundant. These redundant and delayed signals are filtered out as well.

The third sub-algorithm performs the *power - signal activity cross-correlation analysis* of the remaining signals. We calculate the cross-correlation $r_{(\Delta\mathbf{p},\Delta\mathbf{x}_i)}(\tau)$ for the forward differences $\Delta f(x) = f(x+1) - f(x)$ of both the remaining signals $\mathbf{x}_i$ and the transient power profile **p** as expressed in Equation 4.

$$r_{(\Delta\mathbf{p},\Delta\mathbf{x}_i)}(\tau) = \sum_{k=0}^{K-1} \Delta\mathbf{p}(k) \cdot \Delta\mathbf{x}_i(k+\tau), \quad 0 \leq \tau \leq Th_{lag} \tag{4}$$

This can be interpreted as a measure for the relation of signal state changes to changes in the transient power consumption of the system. Note that this is a measure similar to *power sensitivity* as introduced in [14]. All signals exhibiting low correlation within a certain lag Th_{lag}, e.g., typically a small number of clock cycles, are not considered power-relevant and are removed.

The list of candidate model parameters L_c now only contains power-related and non-redundant signals. While iterating over the whole set of available microbenchmarks, the list of

[2]Keeping the signal originating at the highest level of design hierarchy is in general beneficial for the automated HDL model integration as it reduces the amount of hierarchical levels - and therefore typically also the amount of HDL files - that have to be modified (see Section V-A).

potential power model parameters L_{pms} is compiled as the concatenation of all candidate parameter lists L_c. This list can then be used in the model coefficient fitting process.

C. Model Coefficient Fitting

Based on the power model parameters list L_{pms} compiled by the parameter selection algorithm and containing N_{pms} parameters, the complete signal state matrix $\mathbf{X}$ of size $K \times N_{all}$ can be reduced to $\mathbf{X}_{pms}$ of size $K \times N_{pms}$ by eliminating all non-used parameters. The reduced training set data tuple is then $\mathbf{T}_{pms} = (\mathbf{X}_{pms}, \mathbf{p})$, with the either simulated or measured power vector $\mathbf{p}$ as introduced above.

Determining the vector of fitting coefficients $\mathbf{c}$ requires solving the typically heavily overdetermined equation system $\mathbf{p} = \mathbf{X}_{pms}\mathbf{c}$. To this end, we employ Lawson's well known linear least squares regression technique with non-negativity constraint (*Nonnegative Least Squares*, NNLS) [15]. This algorithm aims at minimizing $\|\mathbf{X}_{pms}\mathbf{c} - \mathbf{p}\|$ so that $\mathbf{c} \geq 0$. The non-negativity constraint on the power model coefficients helps reducing power emulation hardware complexity while increasing model robustness.

V. Automated HDL Model Implementation

The automated power characterization, parameter selection and model fitting methods as introduced in Section IV allow for the rapid power modeling of a given system-under-test for the purpose of enabling power emulation. The second obstacle in this scope is constituted by the time-consuming and tedious task of implementing this automatically devised power model in hardware and integrating it into the existing HDL model of the system-under-test.

We address this obstacle in a twofold manner as depicted in Figure 4: 1) The *automated adaptation of the existing HDL model of the system-under-test*, allowing for the monitoring of internal power-relevant state signals by the power emulation unit, i.e., the "routing" of power-relevant state signals originating in different design entities at various levels of hierarchy. 2) The *automated power emulation hardware generation*, i.e., the HDL implementation of the devised power model.

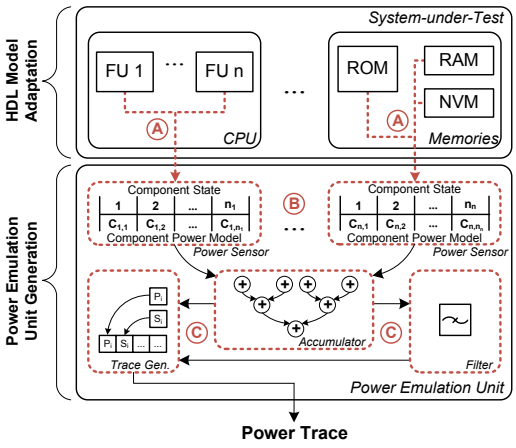

Fig. 4. Overview of the automated HDL model implementation, dotted lines indicate automatic modifications: (A) "Routing" of state signals in HDL model of system-under-test, (B) number and structure of power sensors according to the used power model and (C) internal structure of the power emulation unit

A. HDL Model Adaptation

One of the remaining challenges for automatically enabling the power emulation of the system-under-test is the vital task of adding the additional connections of power-relevant signals, originating at various hierarchical levels of the design, to the power sensors as illustrated in Figure 4. For this purpose we have implemented an algorithm that utilizes the VMAGIC (VHDL manipulation and generation interface) Java library presented in [16]. VMAGIC allows for VHDL code analysis, manipulation and generation. This library itself builds upon the ANTLR 3.1 parser generator [17]. Our adaptation algorithm consists of the following steps as illustrated in Figure 5:

- VHDL parsing: Parsing of all files associated with the system's VHDL model.
- Dependency extraction: Creation of a tree-representation of the dependencies and hierarchical levels of all design entities.
- Signal routing: Routing of all signals specified in the power model to the according power sensor. This step includes the insertion of additional intermediate signals as well as the inherent modification of the interfaces to include these signals.
- VHDL modification and write-back: Generation of synthesizable VHDL code of the modified components.

B. Power Emulation Unit Generation

After adapting the system-under-test's HDL model for power emulation, the final step is integrating the HDL model of the power emulation unit itself with the system-under-test in a common design that can be synthesized and downloaded to the FPGA platform. For this purpose, the HDL model of the PE architecture is split into several HDL template files, containing a number of tags that can be easily parsed and replaced by an adaptation algorithm. Essentially, this algorithm is adapting the following components and properties of the power emulation unit (see Figure 4):

- The number of *power sensors* as well as their internal structure, i.e., the number of power tables and their according bit-widths according to the power model and the desired resolution of the power estimates.
- The power estimates *accumulation unit*, i.e., essentially the width and height of the implemented adder tree.
- The length and bit-width of the *filtering unit*.

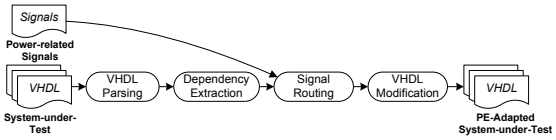

Fig. 5. HDL model adaptation algorithm

- The debug *trace generator* bit-widths according to the accumulation and filtering unit.

VI. Experimental Results

We have evaluated our automated power characterization and HDL model implementation methodology on a smart card microcontroller test system supplied by our industrial partner. This system has been further extended for enabling power emulation in an industrial setting. For different power models, with parameters selected either manually or automatically, we have evaluated the characterization effort as well as the accuracy and the hardware implementation effort.

A. Test System and Used Power Models

A smart card microcontroller based on a 16-bit pipelined cache architecture, composed of volatile and non-volatile memories as well as a number of peripherals, e.g., cryptographic coprocessors, UARTs, timers and random number generators, is used as our system-under-test (see Figure 6). This system has been extended with the power emulation unit as shown in Section V. What makes the system particularly interesting in terms of power characterization is the fact that due to its typical operating environment particular care has been taken to achieve a power-optimized design. Therefore, dedicated power-aware system states (such as low-power halt modes of certain components), also have to be considered by the characterization process.

For illustrating the benefits of using the automated power characterization process, we have compared various power models in terms of accuracy and characterization effort. To this end, four different power models have been benchmarked: 1) A *manual model*, i.e., the parameters were selected manually, devised for the first implementation of our power emulation unit. 2) A naive *brute force* approach - based power model that was created by performing linear regression separately for each microbenchmark using the entire large set of signals found by the *signal name pattern matching* algorithm. All parameters assigned significant coefficient values in this process were considered candidate model parameters. 3) A power model only using parameters retained after the *intra-signal correlation* filter. Finally, 4) a model employing only parameters retained after the *intra-signal correlation and the power-signal activity* filter. Note that for all these models coefficient fitting was performed using the same technique as described in Section IV-C.

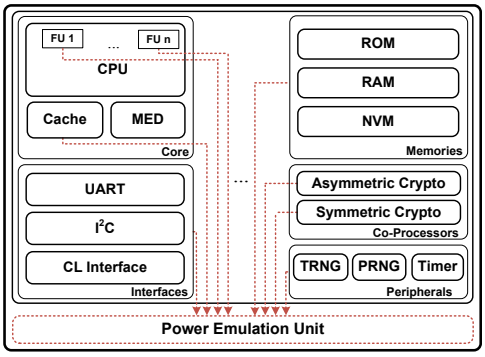

Fig. 6. 16-bit smart card microcontroller test system augmented by power emulation unit and internal state-signal connections

B. Comparison of Accuracy

We have evaluated the accuracy of the power emulation results employing these power models against gate-level power simulations using *Synopsys Primetime PX V2008.12 SP3* [18] on the basis of a cycle-by-cycle comparison. This evaluation was performed on the set of microbenchmarks used for the characterization process, i.e, the training set (TS) as well as for a different set of control benchmarks, i.e, the control set (CS). For this control set of benchmarks we have used general purpose embedded benchmarks from the MiBench suite [19] as well as Dhrystone. Due to the application-specific nature of our test system, we have employed a custom benchmarking application utilizing system components such as the cryptographic co-processors and other peripherals that are not covered by standard CPU benchmarks.

Figure 7 illustrates the average estimation error for these benchmarks. Figure 8 further summarizes these numbers

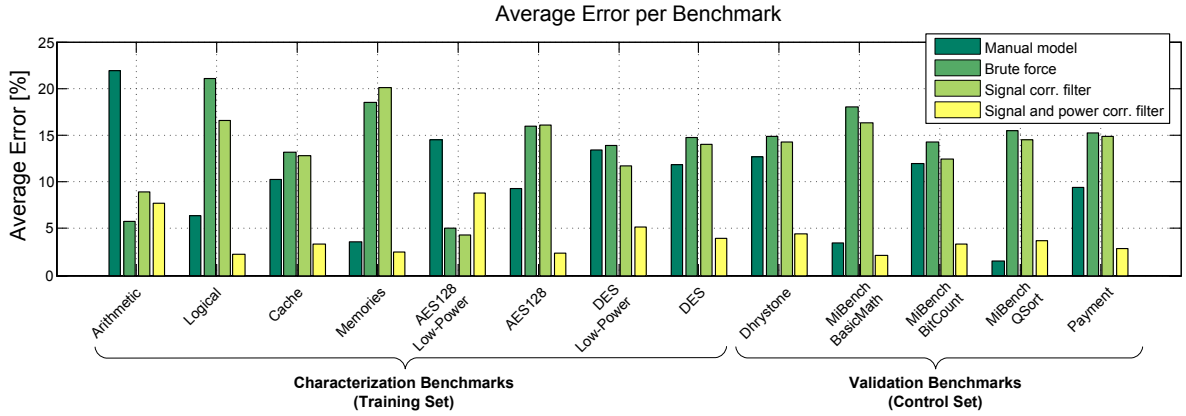

Fig. 7. Comparison of average estimation error for various benchmarks

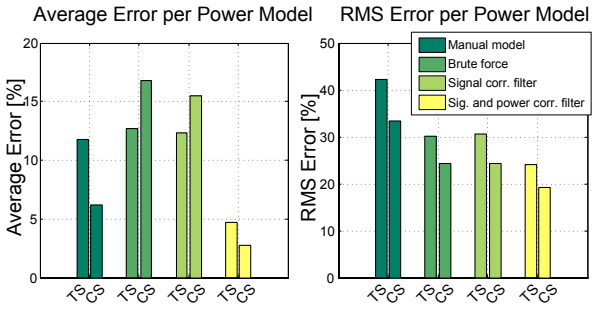

Fig. 8. Comparison of average and RMS estimation error for different power models over all benchmarks from the training set (TS) and the control set (CS)

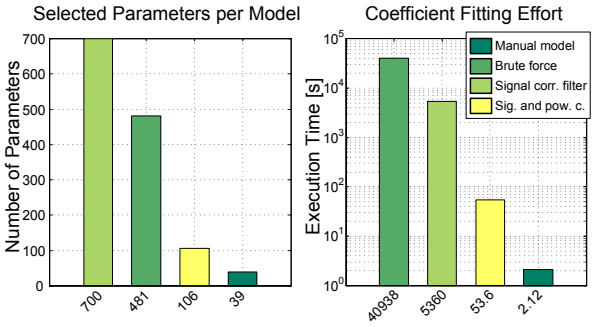

Fig. 9. Comparison of number of coefficients and required execution time for coefficient fitting (Note semilogarithmic scale for second plot)

across all benchmarks and additionally illustrates root-mean-square error (RMSE) values used as a measure for the cycle-by-cycle accuracy. It can be seen that the power model created by the intra-signal correlation and the power-signal activity algorithm reduces the average error as compared to the manual model from 11.78% to 4.71% on the training set and from 6.18% to 2.78% on the control set. The same is valid for the RMSE that is brought down from 42.24% to 24.04% (TS) and from 33.46% to 19.07% (CS). Furthermore, this approach also outperforms the brute force and the stand-alone intra-signal correlation approaches with regard to the same metrics. The relatively high RMSE at this point is due to fact that we are utilizing a *single* power model representing the entire test system in order to decrease the impact of power emulation onto the FPGA. We expect to further decrease this number by using a model splitting approach that models on-chip co-processors and memories seperately.

C. Power Modeling and HDL Adaptation Effort

Next, we compare the effort required for selecting the parameters of the models compared above, for performing the linear regression based coefficient fitting and for the required HDL adaptation. Note that all methods operate on the same training set data. Hence, this portion of the total time required for characterization is constant for these models and is not reflected in Table I.

Due to the different nature of the used parameter selection methods, the number of obtained parameters varies (see Figure 9). Likewise, the execution time for performing linear regression - based coefficient fitting varies with the number of coefficients. Note, however, that the execution time is not only dependent on the number of parameters but also on the conditioning of the input data. Even though the number of parameters determined by the signal correlation filter and the naive brute force approach is by far larger than for the combination of signal and power correlation filter, these models fail to deliver the same level of accuracy. Another interesting observation is that the percentage of remaining non-zero coefficients after performing the NNLS fitting algorithm is the highest for the signal and power correlation filter method as shown in Table II. When considering the lower model parameter selection and coefficient fitting efforts (Figure 9), the automated parameter selection process is even more favorable.

To give an understanding of the required efforts for the automatic HDL implementation as introduced in Section V, we have compared the efforts for adapting the HDL model of this particular system-under-test for the automatically generated

TABLE I
COMPARISON OF PARAMETER SELECTION EFFORT FOR DIFFERENT POWER MODELS

Parameter Selection Method	Effort / Execution Time
Manual selection[a]	several days
Brute force[b,c]	$\sim$ 12 days
Signal corr. filter[c]	8.2 min
Signal and power corr. filter[c]	8.3 min

[a]Iterative parameter selection process by the power emulation unit HW designer.
[b]Sequential linear regression for all benchmarks, candidate parameters from *signal name pattern matching*.
[c]Executed on a 3 GHz AMD Opteron system.

TABLE II
COMPARISON OF SELECTED PARAMETERS AND REMAINING NON-ZERO COEFFICIENT AFTER NNLS FOR DIFFERENT POWER MODELS

Power Model	# Sel. Param.	# Rem. Coeff.	%
Manual model	39	15	38.5
Brute force	481	196	40.7
Signal corr.	700	186	26.6
Sig. & pow. corr.	106	54	50.9

TABLE III
COMPARISON OF HDL IMPLEMENTATION EFFORT

HDL Implementation Method	Effort / Execution Time
HDL analysis[a]	28.9 s
Dependency extraction[a]	4 s
Signal routing & Write-back[a]	24.3 s
PE unit HDL generation[a]	0.2 s
Automatic implementation total[a]	57.4 s
Manual implementation	several hours

[a]Executed on a 3.2 GHz Intel Xeon system.

TABLE IV
COMPARISON OF NUMBER OF COEFFICIENTS AND IMPACT ON EMULATION PLATFORM FOR DIFFERENT POWER MODELS

Power Emulation	Coefficients[a]	Utilization[b]
Manual model	15	0.8%
Brute force	196	4.2%
Signal corr. filter	186	4.1%
Signal and power corr. filter	54	1.6%
Functional Emulation	*n.a.*	66%

[a]Remaining non-zero coefficients after performing linear regression.
[b]Percentage of total available ALUTs on Altera Stratix II platform.

model in Table III. The table summarizes the execution times for adapting the HDL model comprising 400 files in total, ∼10% thereof are being modified. Note that the amount of manual effort will certainly vary largely with the designer's expertise.

D. Impact on Emulation Platform

The power emulated test system was implemented on an Altera Stratix II platform. The microcontroller itself, as used for functional emulation only, employs approximately 66% of the platform's available adaptive look-up tables (ALUTs). Table IV lists the additionally required ALUT percentage for implementing the respective power model.

The power model automatically generated using the signal and profile correlation filter requires additional 1.6% of the platform's ALUTs as opposed to the 0.8% of the manual model. Considering the overall increase in accuracy, we consider the impact of the larger power emulation hardware on the emulation platform as minor. Due to this minor impact, the system can be operated at its targeted clocking frequency of 33 MHz. Note that this relatively low clocking frequency resembles a requirement of the test system and not a limitation of the power emulation unit.

VII. CONCLUSIONS

With the increasing complexity of integrated circuits, power simulations are becoming infeasible due to extensive simulation times. Hardware-accelerated power emulation approaches promise to be a solution to this issue. However, only little attention has been awarded so far to the problem of devising a generic methodology capable of automatically enabling the power emulation of a given system-under-test.

In this paper we have outlined a method for the automatic characterization and power model creation of a given system-under-test for the purpose of power emulation. Using this method a power model has been automatically established for a smart card controller test-system that reduces the estimation error from 11.78% to 4.71% as compared to a manually derived one. Furthermore, we have illustrated how the automatic hardware integration of this power model and the required HDL model adaptation can be achieved. Both, the automated power model creation as well as its automated hardware integration, drastically decrease the overall effort for enabling the power emulation of a given system-under-test. We believe that this reduction of effort will allow the power emulation technique to prove its benefits also in industrial settings.

VIII. ACKNOWLEDGEMENTS

We would like to thank our industrial partners Infineon Technologies Austria AG and Austria Card GmbH for their enduring support in the course of this project. Furthermore, we would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology for providing us with funding for the POWERHOUSE project under the FIT-IT contract FFG 815193.

REFERENCES

[1] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proc. of the 9th ACM SIGOPS European workshop*, 2000.
[2] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *Proc. ISLPED*, 2001.
[3] G. Contreras and M. Martonosi, "Power prediction for intel XScale processors using performance monitoring unit events," in *Proc. of the ISLPED*, 2005.
[4] J. Haid, G. Kaefer, C. Steger, and R. Weiss, "A co-processor for real-time energy estimation of system-on-a-chip," in *Proc. of the 45th MWSCAS*, 2002.
[5] J. Peddersen and S. Parameswaran, "Low-impact processor for dynamic runtime power management," *Design & Test of Computers, IEEE*, vol. 25, pp. 52–62, 2008.
[6] J. Coburn, S. Ravi, and A. Raghunathan, "Power emulation: a new paradigm for power estimation," in *Proc. 42nd DAC*, 2005.
[7] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. M. Mendias, "A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip," in *Proc. 43rd DAC*, 2006.
[8] M. Ghodrat, K. Lahiri, and A. Raghunathan, "Accelerating system-on-chip power analysis using hybrid power estimation," in *Proc. of the 44th DAC*, 2007.
[9] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-system chip multiprocessor power evaluations using FPGA-based emulation," in *Proc. of the ISLPED*, 2008.
[10] A. Genser, C. Bachmann, J. Haid, C. Steger, and R. Weiss, "An emulation-based real-time power profiling unit for embedded software," in *Systems, Architectures, Modeling, and Simulation (SAMOS)*, 2009.
[11] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Accelerating embedded software power profiling using run-time power emulation," in *19th PATMOS*, 2009.
[12] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proc. of the ISLPED*, 1998.
[13] A. Raghunathan, N. Jha, and J. Dey, *High-Level Power Analysis and Optimization*. Kluwer Academic, 1998, ch. 3, p. 43.
[14] Z. Chen, K. Roy, and T.-L. Chou, "Power sensitivity—a new method to estimate power dissipation considering uncertain specifications of primary inputs," in *Proc. of the ICCAD*, 1997.
[15] C. Lawson and R. Hanson, *Solving Least Squares Problems*. Prentice-Hall, 1974, ch. 23, p. 161.
[16] C. Pohl, C. Paiz, and M. Porrmann, "vMAGIC - Automatic code generation for VHDL," *International Journal of Reconfigurable Computing*, vol. 2009, 2009.
[17] http://www.antlr.org/.
[18] http://www.synopsys.com/.
[19] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

An Emulation-Based Platform for Power- and Performance-Aware HW/SW Development of Embedded Multi-Core Systems

Christian Bachmann[1], Andreas Genser[1], Michael Lackner[1], Christian Steger[1], Reinhold Weiß[1], and Josef Haid[2]

[1]Institute for Technical Informatics, Graz University of Technology, Austria
[2]Infineon Technologies Austria AG, Design Center Graz, Austria

The rising complexity of embedded multi-core systems renders purely simulation-based design space exploration increasingly difficult. Even for the execution of moderate software workloads, traditional cycle-accurate power consumption and performance simulation approaches fail to deliver results in a timely adequate manner. In this paper we present an emulation-based embedded multi-core evaluation platform that estimates the power consumption and monitors performance indicators of a given system-under-test during its run-time. This information is collected and sent to a host computer for evaluation, allowing for truly power- and performance-aware design phase analysis and optimization of the system. We present the automated generation of multi-core power emulation and performance evaluation hardware and its automated insertion into the system-under-test. In an exemplary case study on a LEON3-based embedded multi-core test system we illustrate the applicability of our approach to novel designs as well as the benefits of the detailed power and performance profiles generation for the design process. For this test system, the power consumption of each core is individually estimated with an average accuracy of more than 95%. Furthermore, for each core 18 performance indicators are monitored.

1 Introduction

In the design space exploration process HW/SW designers have traditionally relied on simulation-based approaches for exploring power consumption and performance trade-offs. However, the rising complexity of embedded multi-core systems that is fueled by the demand for ever increasing performance and new functionality, hinders purely simulation-based design space exploration. Even for the execution of moderate software workloads, traditional power consumption and performance simulation approaches fail to deliver cycle-accurate results in a timely adequate manner. This is even more critical considering the huge number of degrees of freedom for these systems that HW/SW designers have to cope with.

To overcome this problem two main research directions can be identified: First, the derivation of *simulation frameworks* [1–6], typically operating at higher levels of abstraction in order to speed up the simulation process of large multi-core systems. While this higher level of abstraction on the one hand allows for shorter simulation times, on the other hand it also decreases simulation accuracy and in general renders cycle-accuracy infeasible. Therefore, high-level simulations imply the risk of concealing the low-level implications and side effects of design decisions and optimizations (e.g., timing errors due to deficient power management).

Second, with the availability of large but moderately priced field programmable gate arrays (FPGAs), capable of holding entire multi-core designs, *emulation-based approaches* [7–13] for the

Submitted for publication / under review.

hardware-accelerated design evaluation of a system-under-test have emerged. With vast increases in FPGA resources in recent years, these approaches have become a promising alternative to software simulators, allowing for cycle-accurate design exploration and verification at high speeds.

In the context of emulation-based full-system power consumption and performance evaluation, the novel contributions of this paper are as follows:

- We extend our previously introduced high-level power emulation methodology for the use in an embedded multi-core environment, allowing for the individual power estimation of heterogeneous system components.
- We additionally enable the monitoring of performance indicators of the given system-under-test to enable a deeper understanding of trade-offs between power/energy consumption and performance during the design phase. This joint approach greatly facilitates the identification not only of the source of excessively high power consumption or poor execution performance but also the causing mechanisms.
- A case-study on an embedded multi-core test system illustrates the applicability of our approach to novel systems as well as the benefits of our power emulation and performance monitoring platform in the power- and performance-aware design process of these systems.

This paper is structured as follows. In Section 2 we review previous work in the field of simulation- as well as emulation-based HW/SW design space exploration for embedded multi-core architectures. We introduce our multi-core power emulation and performance monitoring platform as well as its automated adaptation to new systems-under-test in Section 3 . To illustrate its applicability in the design of future embedded multi-core architectures we present a multi-core system case study in Section 4 and additional experimental results obtained using this system in Section 5. Conclusions are drawn in Section 6.

2 Related Work

Previous work on power- and performance-aware HW/SW design space exploration of embedded multi-core architectures has mainly focused on *software simulators*. With the availability of sufficiently large but moderately priced FPGAs, the benefits of hardware-accelerated power and performance *emulation platforms* have become a promising alternative to software simulators, offering emulation speeds close to real-time as well as cycle-accuracy.

2.1 Simulation-based Methods

Simulation-based methods seek to increase simulation speed by employing higher levels of abstraction in the simulation process. Full-system single- as well as multi-core simulators have been proposed on various levels of abstraction. SimpleScalar [1] is a widely used architecture-level simulator, upon which the Wattch [2] power simulator builds. SimpleScalar supports multiple instruction-sets, e.g., Alpha, Power PC, x86 and ARM. Simics [3] is another well known full-system simulator at the instruction-set level, offering simulation models for various microprocessor architectures. MPARM [4] is a SystemC-based simulation platform for entire multi-processor systems-on-chip (MPSoC), containing models of the AMBA bus and memories, as well as a cycle-accurate ARM processor instruction set simulator (ISS). MC-Sim [5] represents a heterogeneous multi-core simulator framework, consisting of a functional ISS for the processor cores, cycle-accurate structural models for the interconnect and behavioral models for coprocessors. A native MPSoC co-simulation framework is presented in [6]. It implements a transactional level simulation environment with focus on software timing and performance estimation. A given embedded software application is compiled and annotated on a host PC to reflect the execution behavior of a specific target processor.

In general, simulation-based approaches offer the benefit of early availability during the HW/SW design process. Furthermore, they allow for easy instrumentation of simulated modules at various hierarchical levels for extracting power and performance statistics. However, the higher level of abstraction - required for achieving tolerable simulation times - bears the hazard of concealing crucial low-level effects. Furthermore, with increasing hardware complexity (i.e., increasing number of cores, accelerators and other peripherals) and with an increasing number of statistics to be collected, the simulation speed drastically decreases.

2.2 Emulation-based Methods

Several emulation-based approaches have been proposed for overcoming the limitations of SW simulators. Initial power emulation, i.e., the hardware-accelerated power estimation on an emulation platform, for smaller circuit designs was presented in [7]. While the low-level (RTL) power models employed in this approach result in high estimation accuracy, they also entail large area overheads (on average a factor of three) requiring area and latency reduction techniques. A full-system MPSoC emulation framework has been introduced in [8] and has been employed for the purpose of thermal estimation [9, 10]. An extended version of this framework was later used to study operating system thermal management strategies [11]. The framework utilizes manually inserted sniffers to collect run-time statistics of different system components and estimates the system's power consumption by utilizing power figures of already existing cores. A power emulation approach for a LEON3-based chip multiprocessor system was shown in [12] using a high-level power model in order to reduce the required hardware overhead. The approach utilizes manually inserted hardware performance event counters related to power consumption and a power model implemented in software. However, this approach cannot be regarded as truly non-invasive as the evaluation of the power model in software both slightly modifies the execution behavior of the system as well as consumes additional execution time. An emulation-based framework for the design space exploration of embedded multi-core systems is presented in [13]. Based on the extraction of various metrics of the emulated system, technology-dependent power consumption and area requirements are being estimated. The framework integrates with Xilinx-proprietary FPGA tools [14] and puts its emphasis on network-on-chip (NoC)-based systems. In a case study its applicability to different multi-processor NoC architectures is illustrated but the estimation accuracy is not being quantified.

In recent work we have presented a high-level power emulation framework for single-core smart card architectures [15] as well as its automated power characterization process [16]. This work serves as the foundation for our design phase power consumption and performance evaluation platform for future embedded multi-core architectures, presented in this paper. In contrast to previously published emulation-based approaches we aim at automatically enabling the high-level power- and performance emulation methodology, in order to avoid time-consuming and error-prone manual modifications. Furthermore, the profiling approach is completely transparent both to the system-under-test's hardware as well as to the software applications being profiled. The generation of power estimates of the system and its sub-components as well as the monitoring of performance indicators both take place during the run-time of the system-under-test without affecting or disturbing the system in any way.

3 Embedded Multi-Core Power Emulation and Performance Monitoring Platform

The principle structure of our proposed power emulation and performance monitoring platform is illustrated in Figure 1. A given embedded multi-core system-under-test is implemented on an FPGA board in the same way as required for, e.g., emulation-based functional verification. Additionally, power emulation and performance counting units as well as a power and performance data aggregation units are automatically instantiated in the design-under-test.

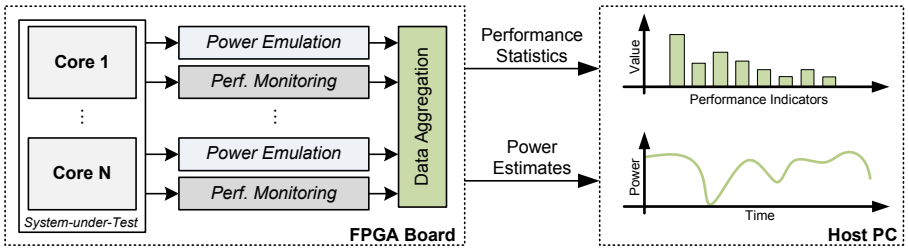

Figure 1: Power and performance evaluation platform

We can identify a number of requirements that have to be fulfilled for enabling the productive design phase use of a power and performance emulation methodology in an industrial setting:

- *Automated instantiation*: The instantiation of hardware units required for power emulation and performance monitoring in the HDL model has to be performed automatically.
- *Transparent and non-invasive*: The additional HW may not influence the system's execution behavior and should ideally be completely transparent to the system.
- *SW controllable*: Optionally, the control as well as the accessibility of statistics through software applications running on the system should be possible.
- *Low impact*: The impact of the additional HW must be as low as possible to allow for the emulation of complex multi-core architectures.
- *Run-time data transfer*: Power and performance statistics should be transferred to a host PC during the system's run-time to avoid the need for large buffer memories.

3.1 High-Level Multi-Core Power Emulation Principle

The power emulation technique is based on augmenting the emulated system with the hardware implementation of its associated power model [7]. Power estimates are generated while the functional emulation of the system-under-test is performed. In contrast to initial low-level power emulation approaches that exhibit a large hardware overhead [7], high-level approaches [12, 15] reduce this overhead by raising the level of abstraction of the used power models.

For CMOS-based systems the total power consumption of a component can be expressed as $P = P_{dyn} + P_{sta}$, consisting of dynamic power consumption P_{dyn} and static power consumption P_{sta}. For the dynamic power consumption term $P_{dyn} = \alpha \cdot (f \cdot C \cdot V^2)$, α represents the activity of the given component whereas the term $(f \cdot C \cdot V^2)$ models the well-known relationship with operating frequency f, switching capacitance C and the supply voltage V. The static power consumption P_{sta} account for the static, i.e., non-activity-dependent power consumption as well as the leakage power. In our high-level power emulation approach the cycle-accurate power consumption $P_i[t]$ of the given i-th system component $(i = 1 \ldots N)$ is approximated using an additive linear equation.

$$\hat{P}_i[t] = \hat{P}_{sta,i} + \hat{P}_{dyn,i}[t] = c_{i,0} + \sum_{j=1}^{N_i} c_{i,j} x_{i,j}[t] = c_{i,0} + c_{i,1} x_{i,1}[t] + \ldots + c_{i,N_i} x_{i,N_i}[t] \qquad (1)$$

In Equation 1, the coefficient $c_{i,0}$ models sources of static power consumption $\hat{P}_{sta,i}$ for the given component such as analog sub-components and leakage. The coefficients $c_{i,j}$ $(j = 1 \ldots N_i)$ express the non-activity-dependent term of CMOS power consumption $(f \cdot C \cdot V^2)$. The activity factor α_i of a given sub-component is determined by the according state signal $x_{i,j}[t]$. The power estimate

of the entire multi-core system can then be simply derived as the sum of all component power estimates as expressed in Equation 2.

$$\hat{P}[t] = \sum_{i=1}^{K} \hat{P}_i[t] = \hat{P}_1[t] + \ldots + \hat{P}_K[t] \quad (2)$$

The power model for each component is derived using an automated power characterization methodology as outlined in the following section.

3.2 Multi-Core Power Characterization

For enabling the power emulation of a future multi-core system-under-test we need to derive its power model, i.e., a set of power models P_i for all major power-relevant system components. The manual creation of these models - the system's *power characterization* - is a time-consuming and error-prone endeavor. For this reason we have introduced an automated characterization methodology that determines power model parameters as well as fitting coefficients [16]. We have further extended this characterization method to establish power models for the multi-core system-under-test as depicted in Figure 2:

1. The system-under-test is implemented for a specific technology node, resulting in a gate-level netlist representation.
2. For each major component the execution of a set of M microbenchmarks is simulated, generating an internal component activity data matrix $\mathbf{X}_i = [\mathbf{X}_{i,1}, \ldots, \mathbf{X}_{i,M}]^T$.
3. The activity data is used in conjunction with a state-of-the-art power estimation tool to generate power consumption training set data vector $\mathbf{p}_i = [\mathbf{p}_{i,1}, \ldots, \mathbf{p}_{i,M}]^T$ for each component.
4. By analyzing the activity and power consumption data tuple $\mathbf{T}_i = (\mathbf{X}_i, \mathbf{p}_i)$, a power model parameter selection process selects candidate model parameters $x_{i,j}$ [16].
5. A linear regression-based coefficient fitting process determines power model coefficients $c_{i,j}$ for all parameters [16].

Once the characterization process is completed for the entire system-under-test, the derived power models are fed to the subsequent power and performance emulation hardware generation process as outlined in Section 3.5.

3.3 Performance Monitoring Principle

Built-in system *performance monitoring units*, also called *hardware performance counters* (HPC), have become a vital tool for performance profiling, analysis and tuning of HW/SW systems. Modern microprocessors typically support a certain number of these HPCs that can be configured to monitor performance-relevant events such as cache misses/hits, bus activity, pipeline stalls, etc. In contrast to software simulators relying on simplified performance models, the HPCs allow designers to evaluate the impact of optimizations on real hardware.

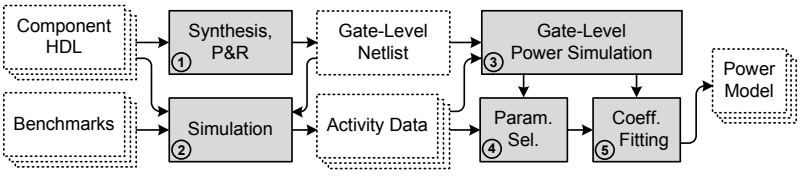

Figure 2: Power characterization flow

While the number of HPCs as well as the range of events that can be profiled is limited in standard microprocessors, our emulation-based approach offers a greater degree of flexibility by allowing the insertion of arbitrary, user-defined performance counters. We employ a set of modular performance monitoring units, observing the behavior of each core and its sub-components. Based on internal signal activity, a set of predefined events $e[t] = (e_1[t], \ldots, e_N[t])$ is detected and logged by the performance monitoring unit for the entire system-under-test.

$$e_i[t] = f_i(x_{i,1}[t], \ldots, x_{i,N_i}[t]); \tag{3}$$

For each performance event $e_i[t]$, an activation function f_i has to be defined that is specifying which combination and state of different signals corresponds to the occurrence of the given event. The activation function in Equation 3 could be a simple logical conjunction of input signals such as $f = x_1[t] \wedge \ldots \wedge x_N[t]$ but also a more complex logical connective, e.g., $f = x_1[t] \wedge x_2[t] \vee \ldots \vee \neg x_N[t]$. Furthermore, logical connectives of partly time-delayed signals such as $f = x_1[t] \wedge x_2[t-2] \vee \neg x_3[t-1]$ could as well be of interest for a given system-under-test, e.g., for detecting certain changes in a component's state or operating mode.

The number and type of events to be profiled by the performance monitoring unit is determined by a system-specific configuration supplied by the user. This configuration specifies the hierarchical level and name of all signals to be monitored as well as the logical connective that defines the occurrence of a performance event.

3.4 Multi-Core Power Emulation and Performance Monitoring Architecture

The multi-core power and performance evaluation architecture as shown in Figure 3 serves the purpose of joint power emulation and performance monitoring. For each major system component that shall be power and performance profiled, both, a *power emulation unit* and a *performance monitoring unit* are instantiated. Furthermore, a *data aggregation unit* collects power emulation as well as performance monitoring data and aggregates the data until it can be transferred to the host computer.

3.4.1 Power Emulation Unit

The power emulation (PE) unit [15] represents the hardware implementation of the given component power model as expressed in Equation 1. This unit monitors the power-relevant state signals $x_{i,j}[t]$ and derives cycle-accurate power estimates $\hat{P}_i[t]$ from these data. Each PE unit consists of a number of power sensors, monitoring the state and activity of various sub-components. Each power sensor maps the observed state signals to a corresponding power value for the given module using a look-up table approach. The power estimator itself accumulates the coefficient values output by each power sensor and calculates the cycle-accurate overall power consumption estimate $\hat{P}_i[t]$ of the component. A data aggregation unit collects all component power estimates for post-processing and transfer to the host.

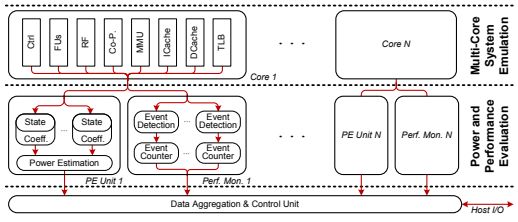

Figure 3: Power emulation and performance monitoring architecture

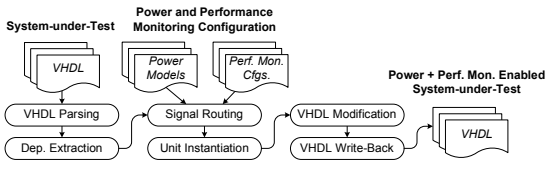

Figure 4: Automated HDL modification for enabling the power and performance monitoring of a given system

3.4.2 Performance Monitoring Unit

The performance monitoring unit serves the purpose of performance profiling of a given system component. It is split into two major parts: (1) The performance-relevant event detection stage and (2) the event logging functionality.

The event detection stage monitors performance-relevant signals $x_{i,j}[t]$ and compares these signals to predefined event patterns. For each performance event a single- or multi-bit signal is monitored. Depending on the activation function f_i for the given event $e_i[t]$ to be detected, either certain signals states or signal state transitions are being detected as an event. After the detection of a performance event, a trigger signal is activated.

In the event logging stage a counter process is instantiated for each performance event. The counter value is incremented when the trigger signal is activated. All counter values are being reset when their current values have been collected by the data aggregation unit for the transfer to the host computer.

3.4.3 Data Aggregation Unit

The data aggregation unit collects power and performance profiling data from the according units. The amount of profiling data is dependent on the number of components monitored, the monitoring granularity (i.e., the number of power models and performance events profiled) and the system's clock rate. Hence, the data volume can exceed the maximum data transfer bandwidth to the host. For this reason, the data aggregation unit accumulates profiling values in sum-registers for the timespan between consecutive data transfers to the host.

3.5 Automated HDL Modification for Power and Performance Monitoring

The manual implementation of the power emulation units as well as the required adaptation of the original HDL model of the system-under-test entails a considerable level of effort. Therefore we have devised an automated HDL model analysis and modification methodology [16] based on the VMAGIC VHDL manipulation Java library [17] that is itself relying on the ANTLR [18] parser generator. For the purpose of constructing the multi-core power and performance evaluation platform, we have further extended this approach to support per-component instantiation and connection of both the power emulation units and the performance monitoring units. The adaptation algorithm is composed of the following steps as shown in Figure 4:

- VHDL parsing and dependency extraction: The entire set of files representing the system's VHDL model is being parsed. A hierarchical dependency tree of all design entities is created.
- Signal routing and instantiation of the power emulation units as well as the performance monitoring units: The units are being instantiated as defined by the power models and the performance monitoring configurations. All internal component signals that are required by either the power emulation, the performance monitoring units or both are being routed from their origin to the respective units.
- VHDL modification and write-back: Synthesizable VHDL code is being generated for all modified components and being written back to the affected VHDL files.

4 Case Study: Multi-Core LEON3 Power and Performance Evaluation Platform

As a case study we have implemented a joint power and performance evaluation platform based on the Gaisler LEON3 [19] architecture. This LEON3 system has been further extended for

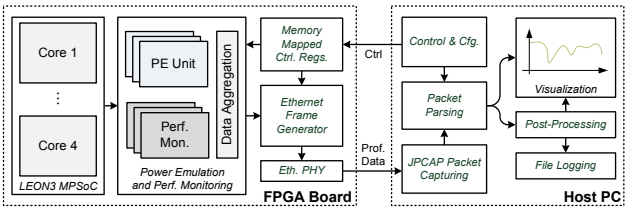

Figure 5: Test system consisting of LEON3 MPSoC augmented by power emulation and performance monitoring functionality

Architecture	32-bit SPARC V8
Pipeline	7-stage, single-issue
Funct. Units	Add, Shift, Mul, Div
I/D Cache	1-way, 8KB/2KB
Clock Rate	35 MHz
Technology[1]	90 nm

Table 1: Architectural parameters chosen for the cores of the test system

enabling (1) multi-core power emulation as well as (2) multi-core performance event monitoring. Furthermore, an Ethernet-based power and performance data transfer interface as well as the host data processing software have been devised for this platform.

4.1 Test System

Our test system for evaluating the power and performance evaluation platform comprises an FPGA board where the actual platform is implemented and a host PC that collects data generated by the platform via an Ethernet-link as shown in Figure 5.

4.1.1 FPGA Board

We employ a Xilinx ML507 development board that is equipped with a Virtex5 XC5VFX70 FPGA. On the FPGA a Gaisler LEON3 multi-core system is implemented, consisting of four 32-bit pipelined SPARC V8 compliant cores with 8 KB of instruction and 2 KB of data caches each. Table 1 lists the architectural parameters chosen for our implementation.

The LEON3 architecture has been further extended with power emulation and performance monitoring units using our methodology as outlined in Section 3.5. The aggregated power and performance profiling data generated during the emulation run is collected by an Ethernet frame generator unit that uses the on-board Ethernet physical layer TX unit to transfer the data to the host PC.

4.1.2 Host PC

On the host PC a Java-based application captures and parses incoming Ethernet frames. The received profiling data is being post-processed and either being visualized in a GUI or logged to a file for later evaluation.

4.2 Implemented Power Model

Alongside with the performance monitoring units, each core has been equipped with a power emulation unit incorporating the core's power model. For obtaining reference power consumption data required by the power modeling process, the LEON3 core architecture has been synthesized for a 90 nm process. Afterwards, gate-level power simulations of short microbenchmarks have been conducted using Synopsys Primetime PX V2008.12 SP3 to generate reference power data. The power models have been derived from these gate-level reference power data by performing the automated characterization process as outlined in Section 3.2. For the current implementation of the LEON3's core architecture, we have derived a power model utilizing a total of 53 parameters and their according coefficients. Furthermore, the caches and the register file have been modeled

[1]Target technology node used for power modeling.

Models	Core (IU, FUs, MMU) I/D Caches, RF
Charact. Methods	NNLS (Core) eCACTI (Caches, RF)
# Model Parameters	53 (Core) 6 (Caches), 2 (RF)

Table 2: Power model parameters

General Clock cycles Stall cycles (I/D-Cache,Tot.)	**Register File** Read (S/D) / Write
Execution Stage Add, Logic, Shift Op. Mul, Div Op.	**Inst. Cache** Read Hit / Miss
	Data Cache R/W. Hit/Miss

Table 3: Monitored performance events

μBench.[1]	AA	AL	CA	DR	PD	RA	RO	TI
AE[%]	3.9	2.7	3.0	0.9	0.8	1.5	3.1	2.8
RMSE[%]	15.8	15.9	18.2	16.2	8.5	13.6	14.5	14.2

Benchmarks	Coremark	Quicksort	Bitcount
AE[%]	2.3	4.5	4.5
RMSE[%]	17.7	16.9	22.8

Table 4: Power model average and RMS error for characterization and evaluation benchmarks

in the eCACTI power estimation tool [20] for the same technology node. Table 2 summarizes the used power models and the number of parameters used.

We have evaluated the accuracy of the implemented core power model both for the training microbenchmarks as well as for a set of control benchmarks against reference gate-level power simulations as shown in Table 4. For the characterization microbenchmarks[2] the average error is below 4% while the root-mean-square error (RMSE), which is a measure for the cycle-by-cycle accuracy, is below 19%. For the training set, these values are below 5% and 23% respectively. Taking the high-level nature of our power modeling approach into account, we consider this accuracy to be sufficiently high for the early design phase applicability of our power emulation and performance monitoring platform.

4.3 Implemented Performance Monitoring

For the given LEON3 test system, a total of 18 performance event detection units and associated event counters have been instantiated as summarized in Table 3. The monitored performance events are extracted from each core's pipeline, register file as well as instruction and data cache units. The collected performance profiling data contains information on the type (Add, Shift, etc.) and number of executed instructions as well as the total number of clock and stall cycles. Register file accesses and I/D cache hit/miss statistics are also collected. Furthermore, program counter (PC) values are recorded to enable the correlation of power and performance profiles to the system's execution trace.

4.4 Impact on Emulation Platform

The quad-core LEON3 system-under-test including the power emulation and performance monitoring functionality was synthesized with Xilinx ISE 12.3 [14] for a target frequency of 40 MHz. Of the available 44800 look-up tables (LUTs) on the used Virtex5 emulation platform, a total of 35232 (78.64%) are used. Of these LUTs, the test system requires 80.8% for the four cores (∼17% each) and the peripherals (12.2%) as shown in Table 5. The power emulation units and the performance monitoring units for all cores account to 7.8% and 1.0% respectively. Furthermore, the power

[2]Used microbenchmarks: ALU-Arithmetic (AA), ALU-Logical (AL), Cache (CA), Dhrystone (DR), Power-Down (PD), RAM (RA), ROM (RO), Timer (TI)

Test System	**LUTs [%]**		**Power Em. & Perf. Mon.**	**LUTs [%]**
Core 1	17.3		Power Emulation Units	7.8
Core 2	16.8		Performance Monitoring Units	1.0
Core 3	16.8		**Run-Time Data Transfer**	**LUTs [%]**
Core 4	17.7		Data Aggregation	6.9
Peripherals	12.2		Run-Time Ethernet Transmitter	3.5
Total TS	*80.8*		*Total PE, PM & Data Transfer*	*19.2*

Table 5: FPGA utilization

and performance data aggregation functionality consumes 6.9% and the run-time Ethernet data transfer controller 3.5% of the totally used LUTs.

4.5 Emulation Performance

For illustrating the speed-up by using our emulation-based approach we compare the emulation time of benchmarking applications with RTL simulations and instruction-set simulator (ISS) performance. RTL simulations are performed with Modelsim 6.6b while as ISS the TSIM/LEON3 SPARC simulator 2.0.18 (as an evaluation version) is employed. Both simulators are running on an AMD Phenom II 3.2 GHz server system. Table 6 illustrates the speedup in profiling time by using our emulation-based platform in comparison to RTL and IS-level simulations for three benchmarking applications. Note that while the RTL simulation incorporates the power emulation and performance monitoring functionality as outlined in Sections 4.2 and 4.3, the ISS only collects general performance statistics and does not provide power estimation functionality.

For the profiled benchmarks, the speedup of the emulation platform is almost a factor of five whereas for the RTL simulation the speedup is in the range of 50 to 60 thousand, according to application characteristics. Furthermore, to highlight the benefit of employing an emulation-based approach to profile longer workloads, we have used average speedup values to extrapolate RTL and IS simulation times for the booting process of an embedded operating system that requires 11 seconds on the emulation platform. While an IS-level simulation could still be completed within one minute, the RTL simulation would require more than 7 days.

5 Experimental Results

We illustrate the benefits of the joint power emulation and performance monitoring platform in an embedded HW/SW development process by profiling prototypical applications.

Benchmarks	Bitcount	Coremark	Quicksort	OS Booting
RTL Simulation Time	26.46 s	144.56 s	30.71 s	7.39 d[b]
IS Simulation[a]	2.45 ms	11.81 ms	2.51 ms	54.23 s[b]
Emulation Time	0.50 ms	2.39 ms	0.50 ms	11 s
Speedup vs RTL Sim.	52744	60507	60890	58047[c]
Speedup vs IS Sim.[a]	4.9	4.9	5.0	4.9[c]

Table 6: Simulation vs emulation time

[a] Purely functional simulation, no power estimation process.
[b] Value is extrapolated from average speedup of emulation vs RTL/IS sim.
[c] Average value.

5.1 Software Optimization

5.1.1 Compiler Optimizations

Figure 6 illustrates the profiling of the Coremark benchmark on one core of our power emulation and performance monitoring platform. The benchmark has initially been compiled using the SPARC-ELF GCC 3.4.4 with all optimizations disabled (level 0, i.e., no compiler optimizations). Afterwards, optimization level 3 was chosen, allowing the compiler to use common optimizations as well as more expensive ones such as function inlining. The performance profiles nicely illustrate how the compiler manages to optimize data access, hence, strongly reducing the amount of data cache stall cycles and therefore also more than halving the execution time.

5.1.2 Manual Optimizations

An example of a manual optimization approach is shown in Figure 7, illustrating the manual optimization of the cache usage of loops. In the original source code as shown in Listing 1 in a first loop all elements on an array are incremented by a value `op1`, afterwards in a second loop they are multiplied by a value `op2`. Due to the fact that the size of the array is larger than the available data cache, old cache entries of the array items are already overwritten during the execution of the first loop. Hence, even though the same array is modified in the second loop, all entries have to be loaded again from the memory, which is indicated by a constant high number of D-cache stalls during the loop's execution.

In the manually optimized version as illustrated in Listing 2, both loops are split into smaller parts that are only operating on smaller blocks of the array that are completely fitting into the D-cache. The profiling result shown in Figure 7 clearly illustrates how the execution is accelerated for the smaller loops that can operate on the cached array data values. Even though the power consumption is higher during these phases due to the faster execution, the overall energy consumption can be reduced by 8.6% due to the shorter execution time.

5.2 Profiling of Multi-Core Applications: Task Migration

One of the main advantages of our emulation-based profiling architecture presented in this paper is the speedup in comparison to software simulators. This allows for the profiling of applications exhibiting long execution times, such as, the execution of an embedded operating system. As operating system for our LEON3 multi-core test system we have employed an SMP-enabled SnapGear

[1]Data normalized due to existing NDA.

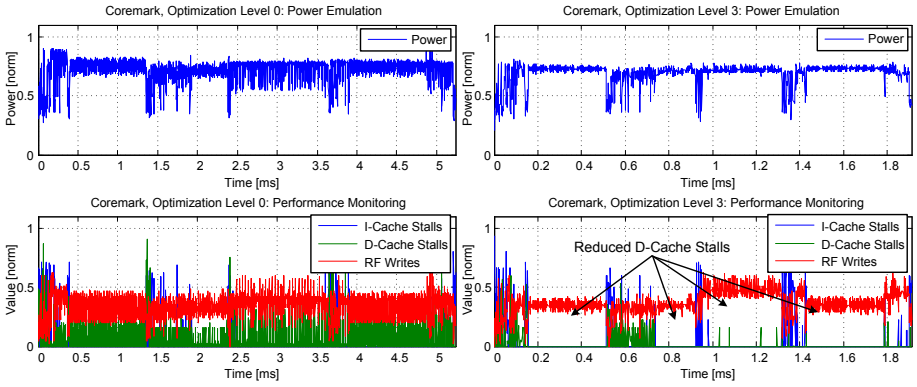

Figure 6: Profiling of benchmark application illustrating the impact of different compiler optimization levels[1]

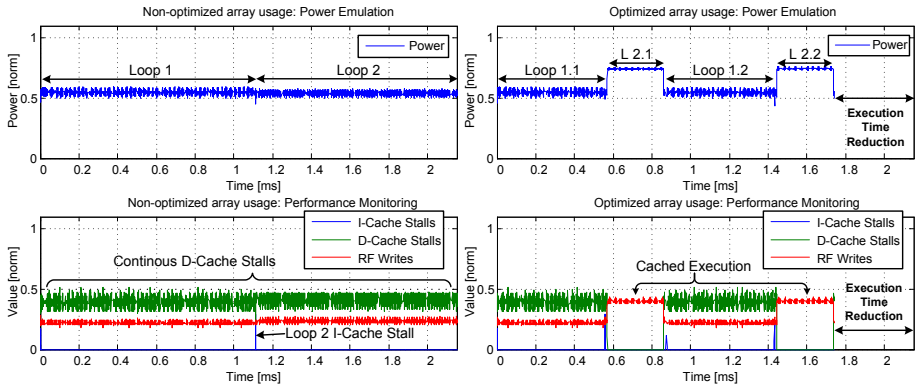

Figure 7: Profiling of benchmark application illustrating the impact of manual array access optimizations[1]

```
//Loop 1
for(i=0..N)
{
  array[i] += op1;
}

//Loop 2
for(i=0..N)
{
  array[i] *= op2;
}
```

Listing 1: Source code example of two consecutive loops

```
for(i=0..N/2) //L1.1
  array[i] += op1;

for(i=0..N/2) //L2.1
  array[i] *= op2;

for(i=N/2..N) //L1.2
  array[i] += op1;

for(i=N/2..N) //L2.2
  array[i] *= op2;
```

Listing 2: Optimized source code considering D-cache size

Linux 2.6-P42 incorporating the Linux kernel 2.6.21.

Using the Linux scheduler, we can now migrate the execution of tasks from one core to the other in a way as it would be required to achieve, e.g., load balancing or thermal-aware scheduling. Our task consists of a simple increment operation executed in a loop. During the tasks execution we force the Linux scheduler to migrate the running loop task from core 1 to core 2 by changing the tasks processor affinity using the `sched_setaffinity` system call. The migration process is clearly visible both in the cores' power profiles as well as in the performance profiles monitoring cache activity as depicted in Figure 8.

6 Conclusions

Simulation-based design space exploration and verification of embedded multi-core systems is becoming increasingly difficult due to the ever increasing complexity both in terms of hardware as well as software functionality. Hardware-accelerated emulation approaches have becoming a promising alternative in filling this design productivity gap.

In this paper we have introduced an emulation-based platform that derives power estimates during the run-time of a given system-under-test and at the same time monitors various performance indicators. For facilitating the productive design phase use of our power emulation and performance monitoring platform in an industrial setting, we have investigated the automated power characterization and HDL model adaptation of future embedded multi-core systems-under-test.

[1]Data normalized due to existing NDA.

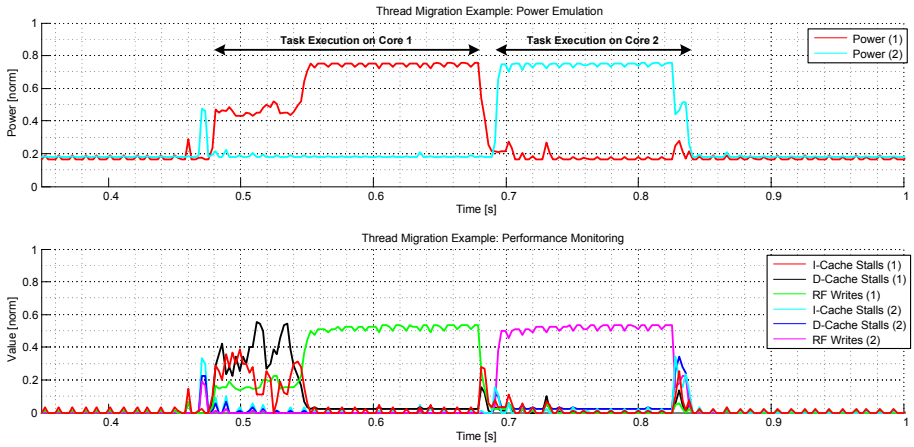

Figure 8: Profiling of Linux task migration procedure between two cores[1]

In a case study of a LEON3-based multi-core system, we illustrate the applicability of our power emulation and performance monitoring approach. The system has been automatically augmented with power emulation units that enable a per-core average power estimation accuracy within 95% of reference gate-level simulations. Furthermore, for each core 18 performance indicators are profiled. The entire system-under-test consisting of four cores and including the power emulation and performance monitoring functionality has been implemented on a Xilinx Virtex5 FPGA. On this platform single- as well as multi-core SW applications can be easily profiled even when exhibiting longer execution times that limit the use of simulation-based approaches.

The collected run-time power and performance profiles enable a deeper understanding of trade-offs between power/energy consumption and performance. This joint approach greatly facilitates the identification not only of the source of excessively high power consumption or poor execution performance but also the causing mechanisms, therefore enabling a truly power- and performance-aware HW/SW development process.

Acknowledgment

We would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology for providing us with funding for the POWERHOUSE project under FIT-IT contract FFG 815193, as well as our industrial partners Infineon Technologies Austria AG and Austria Card GmbH for their enduring support.

References

[1] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, pp. 59 –67, 2002.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pp. 83–94, 2000.

[3] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50 –58, 2002.

[1] Data normalized due to existing NDA.

[4] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "Mparm: Exploring the multi-processor soc design space with systemc," *J. VLSI Signal Process. Syst.*, vol. 41, no. 2, pp. 169–182, 2005.

[5] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik, and G. Reinman, "Mc-sim: an efficient simulation tool for mpsoc designs," in *Proceedings of the 2008 IEEE/ACM ICCAD '08*, 2008, pp. 364–371.

[6] P. Gerin, M. M. Hamayun, and F. Pétrot, "Native mpsoc co-simulation environment for software performance estimation," in *Proceedings of the 7th IEEE/ACM CODES+ISSS '09*, 2009, pp. 403–412.

[7] J. Coburn, S. Ravi, and A. Raghunathan, "Power emulation: a new paradigm for power estimation," in *Proc. 42nd DAC*, 2005.

[8] P. Del Valle, D. Atienza, I. Magan, J. Flores, E. Perez, J. Mendias, L. Benini, and G. De Micheli, "Architectural Exploration of MPSoC Designs Based on an FPGA Emulation Framework," in *Proc. XXI DCIS*, 2006, pp. 12–18.

[9] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. M. Mendias, "A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip," in *Proc. 43rd DAC*, 2006.

[10] D. Atienza, P. G. Della Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, J. M. Mendias, and R. Hermida, "HW-SW Emulation Framework for Temperature-Aware Design in MPSoCs," *ACM TODAES*, vol. Vol. 12, pp. pp. 1 – 26,, 2007.

[11] S. Carta, A. Acquaviva, P. G. Del Valle, M. Pittau, D. Atienza, F. Rincon, G. De Micheli, L. Benini, and J. M. Mendias, "Multi-Processor Operating System Emulation Framework with Thermal Feedback for Systems-on-Chip," in *17th ACM GLSVLSI*, 2007, pp. 311–316.

[12] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-system chip multiprocessor power evaluations using FPGA-based emulation," in *Proc. of the ISLPED*, 2008.

[13] P. Meloni, S. Secchi, and L. Raffo, "An fpga-based framework for technology-aware prototyping of multicore embedded architectures," *IEEE Embedded Systems Letters*, 2010.

[14] http://www.xilinx.com/.

[15] A. Genser, C. Bachmann, J. Haid, C. Steger, and R. Weiss, "An emulation-based real-time power profiling unit for embedded software," in *SAMOS*, 2009.

[16] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Automated power characterization for run-time power emulation of soc designs," in *13th Euromicro DSD*, 2010, pp. 587–594.

[17] C. Pohl, C. Paiz, and M. Porrmann, "vMAGIC - Automatic code generation for VHDL," *International Journal of Reconfigurable Computing*, vol. 2009, 2009.

[18] http://www.antlr.org/.

[19] http://www.gaisler.com/.

[20] M. Mamidipaka and N. Dutt, "ecacti: An enhanced power estimation model for on-chip caches," In Technical Report TR-04-28, CECS, UCI, Tech. Rep., 2004.

Power Emulation: Methodology and Applications for HW/SW Power Optimization

J. Haid
Infineon Technologies Austria AG
Design Center Graz
Graz, Austria

C. Bachmann, A. Genser, C. Steger, R. Weiss
Graz University of Technology
Institute for Technical Informatics
Graz, Austria

Power profiling methods are indispensible in the power-aware design of HW/SW systems. By extending functional emulators with power estimation hardware, high-level power information can be derived during run-time, yielding a considerable speed-up as compared to simulation based approaches. A key enabler for the widespread use of the power emulation methodology is the automation of both power model creation and HDL adaptation. In this paper, we outline our system-level power emulation technique as well as its automatic power modeling and hardware adaptation. Furthermore, applications in the field of HW/SW power management are illustrated.

I. Introduction

Low power hardware platforms offer a wide range of power saving features. The challenge for software developers is to use them in the most efficient way when executing an application. This is not a trivial task as software developers often do not have a deep understanding of silicon technology and vice versa hardware developers are often not deeply familiar with complex software design techniques.

Power emulation is one of the most promising techniques to close this gap and enable software developers to directly check the impact of their code on the corresponding hardware platform. The basic idea is to use emulators not only for functional verification, but also for providing detailed information about the power consumption of the executed code. Compared to measurement-based approaches power emulation does not require expensive measurement equipment as well as allows a more precise correlation between instructions and power consumption.

Functional emulation of a system by means of an FPGA prototyping platform has become a widespread technique for functional verification. By additionally adding power estimation hardware to the system and by coupling this approach with a software development environment on a host computer, real-time power verification is feasible. The early design stage applicability of this approach makes the HW/SW co-design process more efficient and hence increases power efficiency of software and hardware. The principle behind the power emulation approach is depicted in Figure 1.

This paper gives a comprehensive overview on FPGA-based power emulation. It addresses the following aspects:

- An automated power model generation methodology for a given system-under-test
- Implementation of power estimator hardware and its automatic adaptation to the power model
- An automated software optimization approach exploiting results from the power emulator

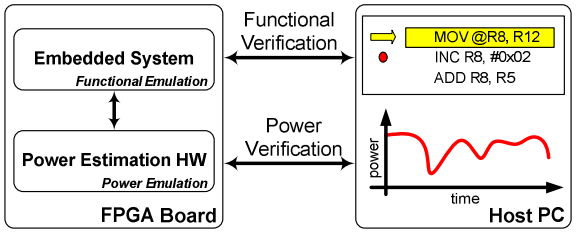

Figure 1: Power emulation principle, obtained from [8]

Finally the paper shows how power emulation can be used to investigate and verify new power management policies with the goal to reduce the power management efforts required in software.

II. Related Work

Numerous works have studied *power profiling* methods for use in SW/HW systems development and verification. Recently, hardware-accelerated *power emulation* approaches have been researched in order to speed up the power estimation process. Based on these approaches, *power management* techniques can be implemented and verified, seeking to increase the battery lifetimes as well as the general system stability of power- and energy-constrained devices.

A. Power Profiling

Power profiling based on simulation approaches acquires activity and state information of given SoCs during program execution. Power estimation can be carried out on various levels of abstract yielding different estimation accuracies and simulation times. However, simulation-based approaches are generally limited to non-real-time execution.

In [1], the authors define an instruction level power model to estimate the application's power consumption. The model incorporates the power consumption while executing instructions (i.e., base costs) as well as the power

consumption while switching between instructions (i.e., circuit state overhead costs). A pipeline-aware power model for VLIW architectures improving instruction-level estimation accuracy is proposed in [2]. In [3], the authors describe a co-simulation based power estimation approach for SoCs. System power estimation is carried out on system-level, while components of interest can be co-simulated on a lower level of abstraction to enhance estimation accuracy.
Moreover, commercial power estimation tools operating on a low level of abstraction (i.e., register transfer or gate level) are shown in [4]. The execution of power estimation on a low abstraction layer leads to accurate results while exhibiting long simulation times that can become extensively long for complex applications. To circumvent this limitation, present trends tend to move from simulation-based approaches to hardware-accelerated approaches.

B. Power Emulation

Hardware-accelerated power estimation speeds up power analysis methods up to real-time behavior by shifting the evaluation of power models from software to hardware. To derive power and energy consumption information from an SoC, existing hardware performance counters [5] or additional hardware counters are added to the system [6].
A special form of hardware-accelerated power estimation is power emulation. The SoC is augmented with power estimation functionality on a typical FPGA prototyping platform. Alongside functional verification, this approach also allows for power verification in an early design stage. The power emulation principle has first been proposed in [7]. The authors achieve run-time reductions between 10x and 500x by performing power emulation on register transfer level. However, the induced hardware overhead accounts up to 3x as compared to the base system.
In recent work we introduced a system-level power emulation approach providing power consumption in real-time at less hardware overhead [8] that can be used to enable power-aware software development [9].
A power characterization methodology that allows for the set up of accurate and low-effort power models forms the basis for estimation based power analysis. To feed power models with SoC activity and state information, access to internal system signals is required. In [10], we introduced methodologies in order to automatically set up accurate power models and to enable the automatic routing of power-relevant system signals.

C. HW/SW Power Management

Power management techniques aim at optimizing the average as well as the peak power consumption of power-constrained devices. Especially for energy harvesting devices such as RF-powered smart cards, power consumption peaks can lead to critical supply voltage drops threatening the system's stability and hence its functionality. Previous attempts on SW power peak reduction have focused on instruction reordering to minimize the switching activity [11] as well as non-functional instruction (NFI) insertion [12]. In security applications the prevention of power analysis attacks by means of power profile flattening has been studied using NFI insertion. Both software and hardware implementations were shown in [13]. A HW based current-injection unit for real-time power consumption flattening was introduced in [14]. This approach has been further extended in [15] using a voltage scaling method to improve the power profile flattening.

III. Power Model and Characterization

A. System-Level Power Modeling

System-level power estimation can be set up by linear regression models, which can be expressed as

$$y = \sum_{i=1}^{n} c_i x_i + \varepsilon, \qquad (1)$$

where $\boldsymbol{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots \boldsymbol{x_n}]$ gives the vector of model parameters. x_i represent power-relevant system states, such as CPU operating modes (run, idle).
$\boldsymbol{c} = [c_1, c_2, \ldots c_n]^T$ gives the vector of model coefficients to be determined during a power model characterization process. The result y represents the estimated power value, which differs from the real power value by the estimation error $\boldsymbol{\varepsilon}$.

A linear regression based power model is set up following three major steps: (i) *Parameter selection*, (ii) *training-set design* and (iii) *least squares fit method*. Power-relevant system states are determined during model parameter selection. The selection of model parameters influences the complexity and the accuracy of the power model.
The training-set incorporates a set of microbenchmarks that are executed on the SoC. For a number of *i* model parameter vectors $\boldsymbol{x^i} = [\boldsymbol{x^i}_1, \boldsymbol{x^i}_2, \ldots \boldsymbol{x^i}_n]$ excited by the microbenchmarks, power measurements are carried out. For all vector sets $\boldsymbol{X} = [\boldsymbol{x^0}, \boldsymbol{x^1}, \ldots \boldsymbol{x^m}]^T$ the corresponding power measurements can be given as $y = [y_0, y_1, \ldots y_m]^T$.
The input vector sets $\boldsymbol{X}$ and associated power values y can be summarized as expressed in (2).

$$y = \boldsymbol{c}\mathbf{X} \qquad (2)$$

In general, the linear system of equations given in (2) is not exactly solvable, since the number of power measurements is usually higher than the number of model parameters. Hence, a least squares fit method finally determines the model coefficients $\boldsymbol{c}$ while trying to minimize the square error for each of c_i.

B. Automated Power Modeling Methodology

For the high-level power modeling approach employed in our power emulation methodology, a challenge that arises is the identification of power-relevant state signals x_i. For the given system-under-test these state signals shall then be used as power model parameters.

We seek to automate this selection process by devising an automated model parameter selection algorithm previously introduced in [10]. This algorithm is integrated into the power characterization flow as depicted in Figure 2 and consists of three stages.

In the first stage, a *signal name pattern matching* is performed. The typically very large number of signals available in the design is largely reduced by only selecting signals matching a list of user-supplied name patterns. A default name pattern list should be derived from the coding standard used while implementing the system-under-test and will contain name patterns such as “enable”, “busy”, “ready”, “halt”, “sleep”, etc.

The second stage consists of the *intra-signal correlation analysis*, aiming at the further reduction of potential parameter signals by removing redundant signals based on their activity statistics. In the final third stage the *power-signal cross-correlation analysis* is carried out, measuring the relation of signal state changes to changes in the transient power consumption of the system. All signals exhibiting only low correlation within a certain lag, e.g., typically a small number of clock cycles, are not considered power-relevant and are removed.

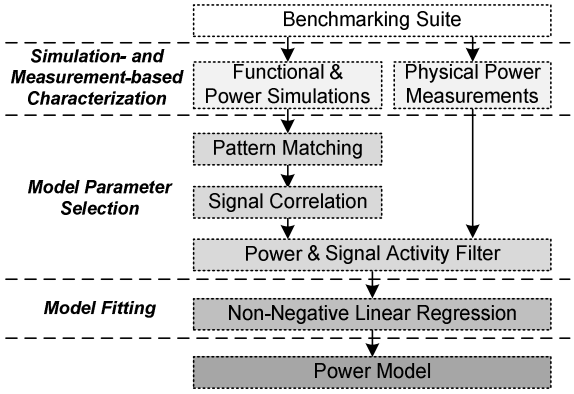

Figure 2: Automated power characterization methodology, obtained from [10]

IV. Power Estimation Hardware Generation and Adaption

A. Power Estimation Hardware

The power estimation hardware that is a key component of the power emulation approach is depicted in Figure 3.

Power-relevant signals from the SoC are fed to power sensors that track state information of system modules. Power model coefficients $\boldsymbol{c}$ are stored in the power sensors and state information is mapped towards model coefficients using a table-lookup approach. The power estimation unit accumulates power sensor outputs yielding the power estimate y. The debug-trace generator collects power information from the power estimation unit, which are then transmitted to a host computer. A configurable averaging filter allows for power information smoothening and de-noising.

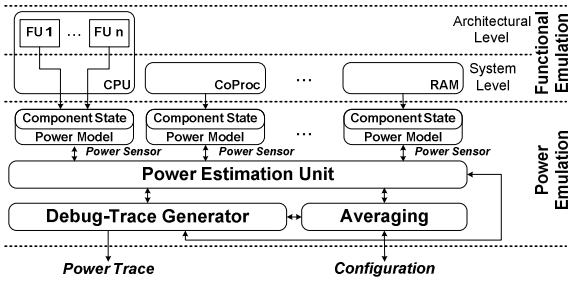

Figure 3: Power estimation hardware, obtained from [8]

B. Automated HDL Model Adaption

Now that power model parameters, i.e., power-related state signals, have been defined and used to create a power model of the given system-under-test, the last obstacle for enabling power emulation is the hardware implementation of this model. We address this issue in a twofold manner as previously presented in [10] and depicted in Figure 4.

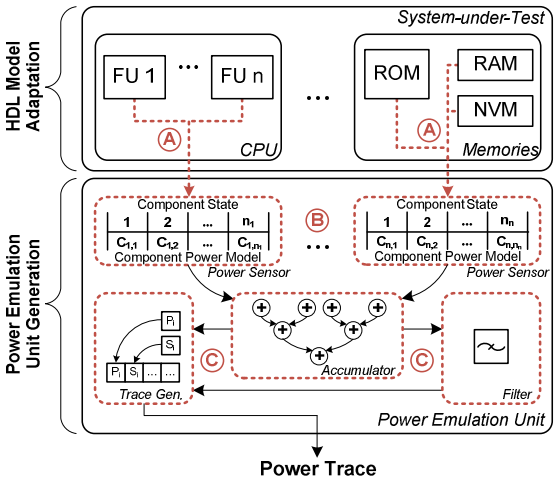

Figure 4: Automated HDL model implementation, obtained from [10]

First, in an *HDL model adaptation* stage, we modify the original HDL model of the system-under-test by adding internal connections of the power-relevant state signals identified by the model parameter selection algorithm outlined above. This step can be interpreted as the "routing" of signals originating in different design entities at various levels of hierarchy to the power sensor units. Our approach

builds upon the VMAGIC (VHDL manipulation and generation interface) Java library [16] that itself builds upon the ANTLR 3.1 parser generator [17].
Second, the HDL model of the power emulation unit has to be integrated into the adapted HDL of the system-under-test to allow for the FPGA synthesis and download. Furthermore, the power emulation unit itself has to be adapted to the used power model, affecting the internal structure of the power sensors, the power estimates accumulation unit, the filtering unit as well as the debug trace generator.

V. HW/SW Power Management Based on Power Emulation

A. Case Study: Smart-Card SoC

The power estimation hardware that is a key component of the power emulation approach can also be exploited for estimation-based power management strategies.
We illustrate the effectiveness of power management approaches on a smart card SoC based on a 16-bit pipelined cache architecture, composed of volatile and non-volatile memories as well as a number of peripherals, e.g., cryptographic coprocessors, UARTs, timers and random number generators. This system has been extended with the power estimation hardware and is used as the test system for evaluating the power emulation approach itself as well as power optimization methods enabled by the power emulation technique.

B. Automated Software Power Optimization Framework for Smart-Card SoCs

One approach for minimizing power consumption peaks, and hence supply voltage drops, is the offline optimization of the software application. This is particularly useful for systems where no run-time power estimates are available and run-time power management techniques are not applicable.
Figure 5 depicts an automated software power optimization approach based on power emulation: 1) A given software application is power-profiled by means of power emulation. 2) Based on these power estimates, a supply voltage simulation of the system's supply voltage circuitry is performed. By analyzing the results of this simulation, power peaks that actually lead to supply voltage drops below a critical level can be identified. 3) The code regions causing the critical power peaks are identified and optimized by means of power management features available on the given system, e.g., by applying frequency scaling around the problematic code regions.

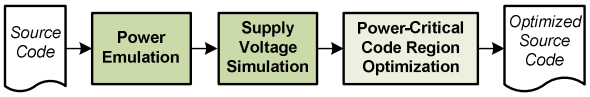

Figure 5: Principle of automated software power optimization

We have applied the software power optimization approach to embedded software applications executed on the smart card microcontroller test system. Figure 6 illustrates the optimization results for a power-critical part of an authentication benchmarking software applications. While in the original version of the application power peaks occur that lead to supply voltage drops below a given limit, in the optimized version these peaks are reduced by means of inserting frequency scaling and non-functional instructions[1]. Hence also the supply voltage drops are diminished and the limit is not being violated.

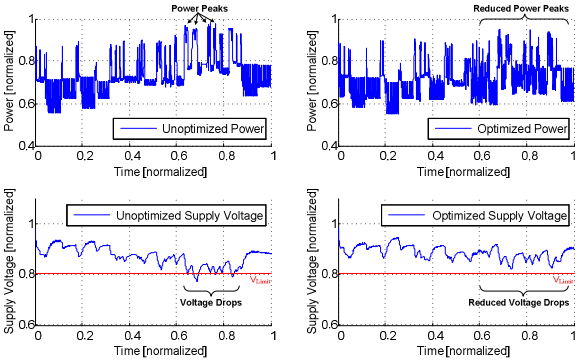

Figure 6: Non-power-optimized vs. power-optimized authentication benchmarking software application[2]

C. Estimation-Based Power Management for Smart-Card SoCs

In this section we exploit power profile smoothening based on the power estimation hardware as a run-time power management variant to minimize the number of power peaks within a power profile. This yields a higher reliability of the overall system due to the minimization of harmful supply voltage drops. The principle of this power management approach is depicted in Figure 7.

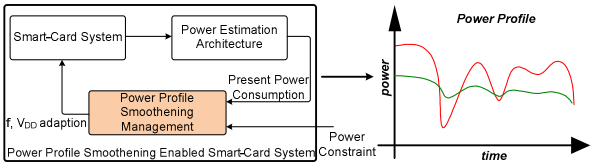

Figure 7: Power profile smoothening principle

We derive the present power consumption information by means of the power estimation architecture in a purely digital manner.
Based on this power information, dynamic voltage and frequency scaling (DVFS) adaptions are applied to the system to stay within given power constraints. DVFS

[1] Non-functional instructions, such as NOPs, typically exhibit a lower power consumption.
[2] Data normalized due to confidentiality agreement with industrial partners.

adaption decisions are taken by power profile smoothening policies that compare the actual power consumption of the system with given power constraints. The availability of the present system's power consumption data as well as given power constraints in the digital domain substitute D/A conversions and allow for a simplified design of run-time power profile smoothening.

Figure 8 illustrates the power profile smoothening result for the execution of a benchmarking algorithm on the smart-card system under given power constraints. First, the system is operated at full operating speed and the maximum supply voltage. The power consumption exceeds the maximum affordable level of 0.6. Second, the estimation-based power profile smoothening approach is applied. Frequency and supply voltage adaptions are carried out in a way that the power consumption stays below the given power constraint. This is illustrated in the middle and bottom subplot of Figure 8.

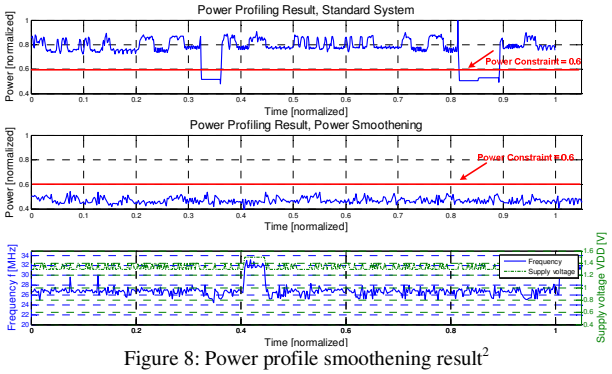

Figure 8: Power profile smoothening result[2]

VI. Conclusions

Power emulation represents a promising alternative to simulation-based power estimation techniques, providing run-time power estimates together with functional emulation.

In this paper we have shown a comprehensive overview on the automation of power model creation and hardware generation for power emulation. Furthermore, we have illustrated the applicability of this technique in software as well as hardware based power management approaches.

Acknowledgment

We would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology for providing us with funding for the POWERHOUSE project under the FIT-IT contract FFG 815193.

References

[1] Tiwari, V.; Malik, S. & Wolfe, A. Power Analysis of Embedded Software: A First Step Towards Software Power Minimization Proc. IEEE/ACM International Conference on Computer-Aided Design, pp. 384-390, 1994

[2] An instruction-level energy model for embedded VLIW architectures. Sami, M.; Sciuto, D.; Silvano, C. & Zaccaria, V. IEEE (J-CAD), Vol. 21, pp. 998-1010, 2002

[3] Lajolo, M.; Raghunathan, A.; Dey, S. & Lavagno, L. Cosimulation-based power estimation for system-on-chip design IEEE (J-VLSI), Vol. 10, pp. 253-266, 2002

[4] Flynn, J. & Waldo, B. Power Management in Complex SoC Design. Synopsys Inc. White Paper, 2005

[5] Joseph, R. & Martonosi, M. Run-time power estimation in high performance microprocessors. Proc. Low Power Electronics and Design, International Symposium on, 2001

[6] Bellosa, F. The benefits of event: driven energy accounting in power-sensitive systems. Proceedings of the 9th workshop on ACM SIGOPS European workshop, 2000

[7] Coburn, J.; Ravi, S. & Raghunathan, A. Power emulation: a new paradigm for power estimation. Proc. 42nd Design Automation Conference, pp. 700-705, 2005

[8] A. Genser, C. Bachmann, J. Haid, C. Steger & R. Weiss: An Emulation-Based Real-Time Power Profiling Unit for Embedded Software, *SAMOS,* 2009

[9] C. Bachmann, A. Genser, C. Steger, R. Weiss, J. Haid: Accelerating Embedded Software Power Profiling Using Run-Time Power Emulation, *PATMOS,* 2009

[10] C. Bachmann, A. Genser, C. Steger, R. Weiss, J. Haid: Automated Power Characterization for Run-Time Power Emulation of SoC Designs, DSD 2010.

[11] Grumer, M., Wendt, M., Lickl, S., Steger, C., Weiss, R., Neffe, U., Muehlberger, A.: Software power peak reduction on smart card systems based on iterative compiling. Emerging Directions in Embedded and Ubiquitous Computing 2007.

[12] Wendt, M., Grumer, M., Steger, C., Weiss, R., Neffe, U., Muehlberger, A.: System level power profile analysis and optimization for smart cards and mobile devices, SAC 2008.

[13] Muresan, R., Gebotys, C.: Current flattening in software and hardware for security applications. CODES + ISSS 2004.

[14] Li, X., Vahedi, H., Muresan, R., Gregori, S.: An integrated current flattening module for embedded cryptosystems. ISCAS 2005.

[15] Vahedi, H., Muresan, R., Gregori, S.: On-chip current flattening circuit with dynamic voltage scaling. ISCAS 2006.

[16] C. Pohl, C. Paiz, and M. Porrmann, "vMAGIC - Automatic code generation for VHDL," International Journal of Reconfigurable Computing, vol. 2009, 2009.

[17] http://www.antlr.org/

Accelerating Embedded Software Power Profiling Using Run-Time Power Emulation

Christian Bachmann[1], Andreas Genser[1],
Christian Steger[1], Reinhold Weiß[1] and Josef Haid[2]

[1] Institute for Technical Informatics, Graz University of Technology, Austria
[2] Infineon Technologies Austria AG, Design Center Graz, Austria

Abstract. Power-aware software development of complex applications is frequently rendered infeasible by the extensive simulation times required for the power estimation process. In this paper, we propose a methodology for rapidly estimating the power profile of a given system based on high-level power emulation. By augmenting the HDL implementation of the system with a high-level power model, a power profile is generated during run-time. We evaluate our approach on a deep-submicron 80251-based smart-card microcontroller-system. The additional hardware effort for introducing the power emulation functionality is only 1.5% while the average estimation error is below 10% as compared to gate-level simulations.

1 Introduction

Power consumption has emerged as the most important design metric for embedded systems, influencing operating time as well as system stability. Therefore, power estimation has become an essential part of today's embedded system design process. In this process, increasingly complex designs have to be handled. Systems-on-chip (SoC) contain large numbers of components, each contributing to the overall power consumption. For these systems, the power consumption is increasingly dependent on software applications determining the utilization of components as well as controlling available power management features.

Currently available simulation tools are usually operating on low levels of abstraction and fail to deliver power estimates in admissible time. Higher levels of abstraction are favourable in order to speed up the estimation process. Furthermore, the greatest power reduction potential can be identified on high levels, e.g., the application layer [1]. However, for estimating the power consumption of elaborate program sequences such as the booting sequence of an operating system (OS), software simulators require extensive runtimes. This curtails the usability of these tools for power-aware software development of complex applications.

Hardware-accelerated methods employing existing hardware counters [2, 3], dedicated power estimation co-processors [4–7] and emulation-based approaches [8–10] have therefore been explored. While low-level emulation-based approaches

suffer from high hardware overhead (on average 3x increased area requirements) [8], high-level event counter-based methods often require additional software processing overhead for evaluating the counters and converting their values to power estimates [10].

We propose a solution to the problem of software power profiling by using an estimation approach based on high-level power models implemented alongside the original system design on a hardware emulation platform.[3] Power consumption estimates are generated during run-time by special power estimation hardware added to the functionally emulated system. By analyzing these estimates, a truly power aware software design methodology can be enabled.

This paper is structured as follows. In Section 2 we discuss related work on software power estimation. Section 3 introduces our contributions to the area, whereas Section 4 evaluates experimental results. Conclusions drawn from our current work are presented in Section 5.

2 Related Work

Previous work on software power estimation can be grouped in two distinct categories: (i) simulation-based and (ii) hardware-accelerated (run-time) estimators.

Simulation-based software power estimation methods determine activity and state data through simulated program execution on a model of the system-under-test. Different levels of abstraction used in describing these models yield different estimation accuracies and variably long simulation times.

Many commercial power estimation tools, e.g., [11], operate on low levels of abstraction such as gate or register transfer level (RTL). These low levels of abstraction allow for very high estimation accuracies but slow down the estimation process. At higher levels, instruction-level power models have been explored. Models consisting of base costs per instruction as well as overhead costs for switching between different instructions have been defined [12]. By considering microarchitectural effects in a pipeline-aware model, the estimation accuracy has been improved [13]. At the system level, a transaction-based framework for complex SoCs has been introduced in [14].

Hardware-accelerated power estimation methods leverage existing or specially added hardware blocks. Using existing hardware event counters, the thread-specific power consumption of operating systems can be determined [15]. Using a similar approach, run-time power estimation in high performance microprocessors using hardware counters is shown in [2] and [3]. Dedicated power estimation co-processors are utilized in [4–7].

By using a standard FPGA platform and a HDL model of a given system that has been augmented with power estimation hardware, *power emulation* and functional emulation can be performed concurrently. RTL [8, 9], high-level event counter-based [10] as well as hybrid methods using simulation and emulation

[3] The PowerHouse project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FIT-IT contract FFG 815193. Project partners are Infineon Technologies Austria AG, Austria Card GmbH and TU Graz.

have been explored [16]. In [17] we have introduced a system-level power profiling unit based on the emulation principle. It allows for generating run-time power estimates and forms the basis of our accelerated software power estimation methodology as first outlined in [18].

3 Embedded Software Power Profiling using High-Level Power Emulation

Our embedded software power profiling method is based on the *power emulation principle* as initially introduced in [8]. A *high-level power model*, created by an adaptable *characterization procedure*, is utilized to estimate the current power consumption as a function of the overall system state.

This model is implemented in hardware as a *power emulation unit* monitoring the states of system components and generating according power estimates during run-time [17]. Out of the power estimates a power trace is recorded and stored on the emulation platform alongside the standard functional trace. These traces are transferred to a host computer running an *integrated software development and debugging environment* (IDE) that is used for visualizing and analyzing the recorded traces.

3.1 Principle of High-Level Power Emulation

Hardware emulation on prototyping platforms, typically on FPGA boards, has become a widespread technique for functional verification. The principle of power emulation is to augment the emulated circuit with special power estimation hardware [8]. By doing so, power consumption estimates can be generated as a by-product of functional emulation.

Emulation-based power estimation offers a number of advantages as compared to power profiling using physical measurements or software simulators. In contrast to physical measurements that are often very coarse-grained and limited to the entire system due to packaging, the emulator allows for calculating cycle-accurate estimates for the whole chip as well as for system subcomponents. Simulation-based estimators offer a high degree of accuracy. However, this accuracy comes at the cost of large simulation times. Due to the calculation of power models in hardware, this simulation time is reduced vastly by the power emulation technique.

Unlike RTL power emulation [8], we employ a high-level model that, due to its high level of abstraction, significantly decreases the complexity of the estimation hardware and therefore reduces its implementation effort. Thus, long combinational paths in the power emulation unit are avoided, enabling the system-under-test to be emulated in or near the targeted clocking frequency of the system's physical implementation. In contrast to high-level event counter-based power emulation approaches, e.g. [10], no further post-processing of counter values in software is required to obtain the power estimates. Our power model is outlined in the following section.

3.2 Power Model

High-level power models, such as system- or component-level power models, often derive power consumption estimates from the state of the system and its components respectively. For a system-level state-based power model this is shown, e.g., in [19]. In our high-level model, the power consumption estimate $\hat{P}_{total}$ of the system takes the form of

$$\hat{P}_{total}(t) = \hat{P}_{idle} + \sum_{i=1}^{N} \hat{P}_i(\tilde{S}_i(t)) = \sum_{i=0}^{N} \hat{P}_i(S_i(t)) \tag{1}$$

where $\hat{P}_i$ denotes the sets of power estimates for a number of N different system components (e.g., CPU, coprocessors, peripherals) for their respective time-dependent states $S_i(t)$. The idle power coefficient $\hat{P}_{idle}$ comprises system components with a constant, i.e., not state-dependent, power consumption such as certain analog components or non-clock-gated parts of the system. We can include the idle term into our power model as a power estimate set containing only one constant power value.

Hence, the state parameters $S_i(t)$ given as

$$S_i(t) = \begin{cases} S_{idle}, & i = 0 \\ \tilde{S}_i(t) = f_i(\mathbf{x_i}(t)), & 0 < i \leq N \end{cases} \tag{2}$$

represents the state of each component including the idle power component. It is a function of the time-dependent component state signal vector $\mathbf{x_i}(t) = [x_{i,0}(t) \dots x_{i,K_i-1}(t)]$, containing the K_i binary control signals x_i that contribute to the state information of the individual module. The mapping function f_i maps the binary control signals x_i to a state value that is used to select the power estimate in the set $\hat{P}_i$. Note that the mapping functions f_i differ for each component, based on the number of control signals and the meaning of each individual signal in terms of state information.

We establish the power model for a given system-under-test by applying our characterization methodology (see Section 3.4) to the HDL model of the system. In the characterization process the granularity of the model is determined by varying the number of included components and the number of states considered, influencing the model's accuracy and its complexity.

3.3 Power emulation unit architecture

Our estimation architecture, as introduced in a previous publication [17], resembles the implementation of the power model in hardware and generates power consumption estimates that are transmitted to and evaluated by a host computer. It is composed of a number of sub-modules as depicted in Figure 1. Power sensors are used to observe the activity and the state of all system components (CPU, coprocessors, memories, peripherals, etc.). Each power sensor monitors the K_i control signals of the according system component and maps the observed

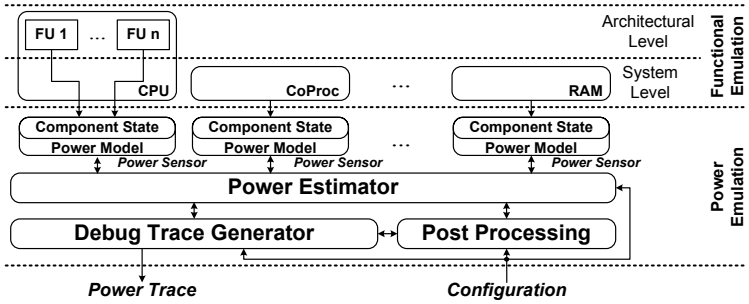

Fig. 1. Architecture of the power emulation unit [17]

state vector $\mathbf{x}_i$ to a corresponding state value S_i by means of the mapping function f_i. The component's state information is then translated to a corresponding power value using a table-lookup approach based on a set of power tables P_i. The power estimator accumulates the outputs generated by the power sensors. The result of a sequence of additions constitutes the instantaneous, cycle-accurate power estimate $\hat{P}_{total}$ for the overall system as pointed out in Equation 1. It is worth noting that the power tables can also be reconfigured during program run-time in order to adapt to, e.g., operating frequency or voltage changes and to allow for the tracking of single components or certain groups of components.

3.4 Characterization Methodology

The straightforward adaptability of the power model and the power emulation architecture to a given system-under-test is one of the key goals of our accelerated power estimation approach. We use a comprehensive characterization methodology for determining a system's power consumption and mapping these results to our power model and the power emulation unit. The basic structure of the characterization process is outlined in Figure 2.

The starting point for this process is a HDL model of the system to be power-emulated. This HDL model is fed to a standard synthesis and place & route design flow, yielding a model of the physical implementation of the system. Based on the physical model, state-of-the-art power estimation tools, in our case *Magma Blastfusion* 5.2.2, can be used to generate power profiles of the system.

A power modelling process derives model coefficients from a set of benchmarking applications covering all system components that contribute significantly to the overall power consumption. Our methodology automatically extracts a list of state signals based on user-defined name patterns, e.g., *busy*, *ready*, *action*, etc. from the HDL model of the system. By analyzing the switching activity data for each signal across all benchmarks, highly correlated (i.e., redundant) and nonrelevant signals are removed. Using regression analysis, power consumption weights are assigned to the remaining control signals. For each system component, all occurring control signal combinations are determined and uniquely mapped to states.

The information obtained in the power modelling step is used in a threefold manner: First, the architecture of the power estimation unit is adapted to suit the requirements of the power model. Therefore, the number of available modules and number of states within each module is adjusted. Second, the original HDL model of the system is adapted in order to allow the power estimation unit to track the internal state of every significant component. This means that signals that can be used to monitor various power states of a component are routed to the power estimation unit. Third, configuration information is generated that is used to setup the power coefficients table inside the power estimation unit according to the system's power model.

The modified HDL model of the system-under-test, augmented with the power emulation functionality, and the set of configuration data obtained in the characterization process can then be used to enable power emulation in a standard software development process.

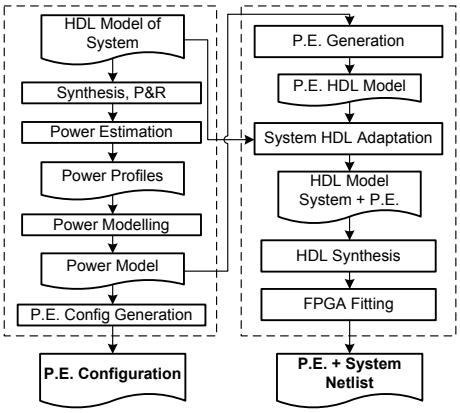

Fig. 2. Characterization methodology for high-level power emulation [18]

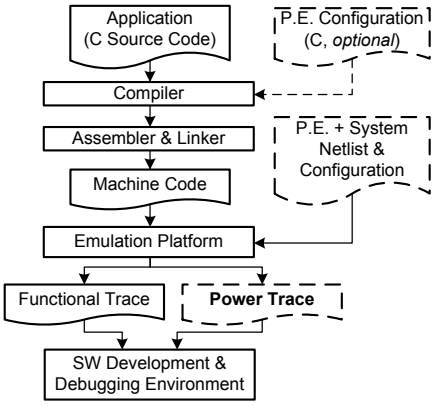

Fig. 3. Software development flow utilizing power emulation

3.5 Power-Aware Software Development Flow

We use a standard software development flow augmented with the power emulation methodology as depicted by Figure 3 to enable power-aware software development.

A software application in C is processed using a C compiler, assembler and linker tool-chain. The resulting machine code is then loaded onto and executed on the power-emulated system. Note that this process of loading and executing an application is not affected by the use of the power emulation methodology. Optionally, the behaviour of the power estimation unit can also be changed by modifying the power configuration tables during normal program execution using routines implemented in C.

The only required change as compared to the standard development flow is the use of a modified netlist for the emulation platform. A netlist resembling the FPGA implementation of the system including the power estimation unit is loaded onto the emulation platform instead of the standard netlist. While executing the machine code on the emulated system a functional trace as well as a power trace are being generated. These traces are stored within an on-board trace memory until they are transferred to the host computer. The software development and debugging environment on the host computer controls the program execution on the emulator (e.g., breakpoint insertion) and evaluates the trace messages received from the emulator.

The power traces can then be plotted alongside the functional trace within the IDE. We use the functional trace to derive the program counter (PC) status as well as the operation code (Opcode) of the current instruction. The functional trace could also be used to correlate the estimated power profile to higher-level debugging information, e.g., for displaying the profile alongside C source code.

4 Experimental Results

We have evaluated our approach on software applications written for a commercially available 80251-based 16-bit microcontroller architecture supplied by our industrial partner. The microcontroller is composed of volatile and non-volatile memories as well as a number of peripherals, e.g., cryptographic coprocessors, UARTs, timers and random number generators.

4.1 Impact on Emulation Platform

An Altera Stratix II emulation platform was used for implementing the microcontroller as well as the power estimation hardware. The microcontroller itself, as it can be used for purely functional emulation, employs approximately 66% of the platform's available adaptive look-up tables (ALUTs). After augmenting the system with the power estimation hardware, roughly 67.5% of the ALUTs are required. Hence, with the additional 1.5% ALUTs we consider the impact of the power emulation hardware on the emulation platform as minor. Note that our approach is generic and could also be implemented on another emulation platform as we do not employ any platform-specific processing elements (e.g., DSP blocks).

4.2 Speed Up

Due to the fact that our estimation unit has only minor impact on the total system and no additional critical paths are introduced, we are able to operate the emulated microprocessor and the estimation unit at the targeted clocking frequency of 33 MHz. Therefore, cycle-accurate power estimates can be generated during run-time. Note that the current implementation of the power estimation unit only employs single-cycle arithmetic operators due to the relatively low

clocking frequency of the system-under-test. For higher frequencies, pipelined multi-cycle operators could be introduced to reduce the critical path.

Compared to the extensive runtime of gate- and RTL simulations the runtime estimation represents a major speed-up. The significant timely difference can be illustrated on a test-run of the Dhrystone benchmark: While a gate-level power simulation on a state-of-the-art server system takes 18.1 hours, the emulation is finished within 139 μs. This vast reduction of simulation time, of course, comes with losses in accuracy which are discussed below.

4.3 Comparison of Accuracy

Figure 4 depicts a comparison between the reference gate-level simulations and the emulated power estimates for a benchmarking application.[1] The magnitude of the average estimation error for the whole application is below 10%. The given results were achieved using a power model taking 5 system modules into account: Two 16 state modules (CPU, cache+memories) and four peripheral modules consisting of two states each. This represents a rather small overall model that could be further extended to include more modules and states, hence improving the estimates. Note that, as pointed out in Section 4.1, considerable FPGA ressources are still available for increasing the accuracy of the model.

4.4 Power-Aware Application Examples

We illustrate the usability of our high-level power emulation approach on two prototypical examples.[1] In these examples the system's power consumption while executing an authentication application exceeds a given maximum power limit. These power-critical events are easily detected by our power emulation methodology and, hence, can be avoided.

The first example, as depicted in Figure 5, shows the invocation of a cryptographic coprocessor for an encryption operation. While waiting for the cryptographic operation to finish, the CPU polls a status register of the coprocessor.

[1] Data normalized due to existing NDA.

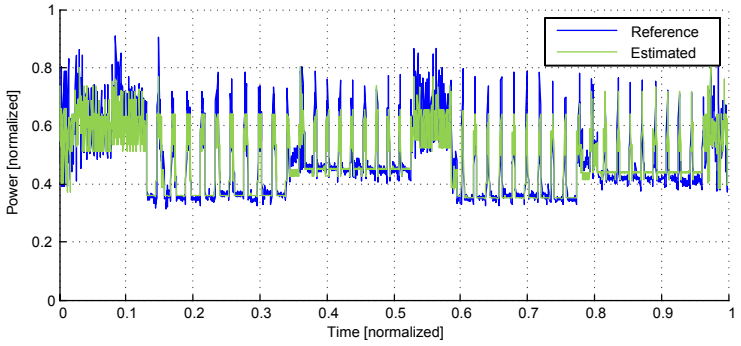

Fig. 4. Comparison of gate-level simulation and power emulation results

During this time, the estimated power consumption indicates that the limit will be exceeded. Hence, in the second implementation, the CPU is set to a special low-power sleep mode and is only reactivated after the cryptographic operation has finished. By emulating the power profile of the power-aware, second implementation we clearly see that the limit is not exceeded any more. Note that slight variations in program execution exist due to different coprocessor setup algorithms and are also visible in the power trace.

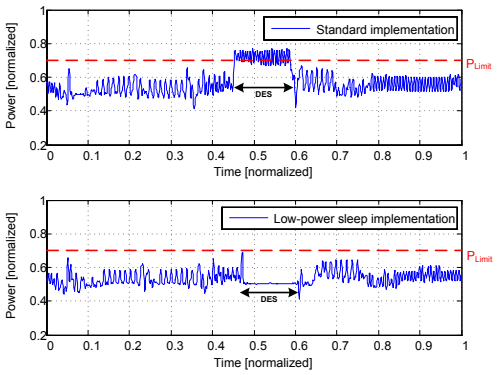

Fig. 5. Profiling of an application utilizing a component's low-power mode.

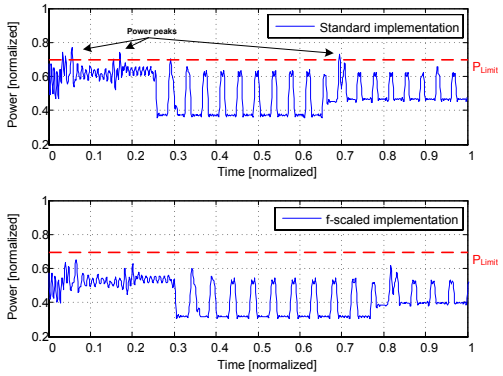

Fig. 6. Profiling of an application utilizing frequency scaling.

In the second example (see Figure 6), the cryptographic results from the example above are further processed. In this phase, additional power peaks above the limit are identified with the help of power emulation. In the power-optimized implementation, these peaks can be diminished by scaling the clock frequency from 33 MHz to 29 MHz. The power emulation result of the modified implementation indicates an acceptable execution time increase while observing the permissible power limit.

5 Conclusions

Power-aware software development is often hindered by the lack of quick power profiling methods. The rapid high-level power emulation approach presented in this paper circumvents this limitation by exploiting hardware acceleration techniques. By combining the functional emulation of a system's HDL model with power estimation hardware, run-time power estimates can be generated on a standard FPGA emulation platform.

We have evaluated our approach on an 80251-based microcontroller system, yielding cycle-accurate power estimates with an average estimation error below 10% in magnitude. These numbers were reported for a high-level power model requiring only 1.5% of the available FPGA ressources. We believe that by mar-

rying these estimates with a standard software development environment truly power-aware software engineering is enabled.

References

1. Macii, E., Poncino, M.: Power macro-models for high-level power estimation. In Piguet, C., ed.: Low-Power Electronics Design. CRC Press (2005)
2. Joseph, R., Martonosi, M.: Run-time power estimation in high performance microprocessors. In: Proc. of the ISLPED 2001. 135–140
3. Contreras, G., Martonosi, M.: Power prediction for intel xscale processors using performance monitoring unit events. In: Proc. of the ISLPED 2005, New York, NY, USA, ACM 2005. 221–226
4. Haid, J., Kaefer, G., Steger, C., Weiss, R.: A co-processor for real-time energy estimation of system-on-a-chip. In Proc. 45th MWSCAS 2002. II–99–II–102
5. Haid, J., Kaefer, G., Steger, C., Weiss, R.: Run-time energy estimation in system-on-a-chip designs. In Proc. of the ASP-DAC 2003. 595–599
6. Peddersen, J., Parameswaran, S.: Clipper: Counter-based low impact processor power estimation at run-time. In Proc. of the ASP-DAC 2007. 890–895
7. Peddersen, J., Parameswaran, S.: Low-impact processor for dynamic runtime power management. Design & Test of Computers, IEEE **25**(1) (Jan.-Feb. 2008) 52–62
8. Coburn, J., Ravi, S., Raghunathan, A.: Power emulation: a new paradigm for power estimation. In Proc. 42nd DAC 2005. 700–705
9. Coburn, J., Ravi, S., Raghunathan, A.: Hardware accelerated power estimation. In Proc. DATE 2005. 528–529 Vol. 1
10. Bhattacharjee, A., Contreras, G., Martonosi, M.: Full-system chip multiprocessor power evaluations using fpga-based emulation. In Proc. of the ISLPED 2008.
11. Flynn, J., Waldo, B.: Power management in complex soc design. Technical report, Synopsys Inc. White Paper 2005
12. Tiwari, V., Malik, S., Wolfe, A.: Power analysis of embedded software: A first step towards software power minimization. In Proc. IEEE/ACM International Conference on Computer-Aided Design 1994. 384–390
13. Sami, M., Sciuto, D., Silvano, C., Zaccaria, V.: An instruction-level energy model for embedded vliw architectures. IEEE (J-CAD) **21**(9) (2002) 998–1010
14. Lee, I., Kim, H., Yang, P., Yoo, S., Chung, E.Y., Choi, K.M., Kong, J.T., Eo, S.K.: Powervip: Soc power estimation framework at transaction level. In Proc. of the ASP-DAC 2006.
15. Bellosa, F.: The benefits of event: driven energy accounting in power-sensitive systems. In: EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop, New York, NY, USA, ACM 2000. 37–42
16. Ghodrat, M., Lahiri, K., Raghunathan, A.: Accelerating system-on-chip power analysis using hybrid power estimation. In: Proc. DAC 2007. 883–886
17. Genser, A., Bachmann, C., Haid, J., Steger, C., Weiss, R.: An emulation-based real-time power profiling unit for embedded software. In: Proc. SAMOS 2009. 67–73
18. Bachmann, C., Genser, A., Haid, J., Steger, C., Weiss, R.: Rapid system-level power estimation for power-aware embedded software design. In: Proc. DSD Work In Progress Session 2009.
19. Benini, L., Hodgson, R., Siegel, P.: System-level power estimation and optimization. In Proc. of the ISLPED 1998. 173–178

An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software

Christian Bachmann[1], Andreas Genser[1],
Christian Steger[1], Reinhold Weiß[1] and Josef Haid[2]

[1] Institute for Technical Informatics, Graz University of Technology, Austria
[2] Infineon Technologies Austria AG, Design Center Graz, Austria

Abstract. In power-constrained mobile systems such as RF-powered smart-cards, power consumption peaks can lead to supply voltage drops threatening the reliability of these systems. In this paper we focus on the automated detection and reduction of power consumption peaks caused by embedded software. We propose a complete framework for automatically profiling embedded software applications by means of the power emulation technique and for identifying the power-critical software source code regions causing power peaks. Depending on the power management features available on the given device, an optimization strategy is chosen and automatically applied to the source code. In comparison to the manual optimization of power peaks, the automatic approach decreases the execution time overhead while only slightly increasing the required code size.

1 Introduction

The power consumption of embedded systems is increasingly dependent on software applications determining the utilization of system components and peripherals. Furthermore, the embedded software actuates power management features such as voltage and frequency scaling as well as dedicated sleep or hibernation states. Hence, software applications impact the average as well as the peak power consumption that is in turn affecting the reliability, stability and security of embedded systems. Especially for RF-powered devices such as contactless smartcards, power peaks threaten the system reliability by impacting the power supply circuit and leading to supply voltage drops [1]. These supply voltage drops can in turn result in system resets or, even worse, in erroneous system states. Therefore, power peak reduction and elimination methods for embedded software have been proposed [2–4]. Furthermore, power peak reduction techniques have been studied for the purpose of power profile flattening in hardware implementations [5–7]. For security applications, the profile flattening resembles a countermeasure against power analysis attacks.

In this paper we propose an automated methodology for profiling a software application's power consumption and deriving a power peak optimized imple-

mentation. Based on an integrated supply voltage simulation, critical code regions are detected and optimized. While existing software optimization methods employ either instruction-level power simulators [2–4] or physical on-chip power measurements [5–7] to obtain power profiles, our approach utilizes a high-level power emulation technique previously introduced in [8]. Using this technique, cycle-accurate run-time power estimates are derived from the system-under-test's functional emulation. In comparison to measurement-based approaches, the joint functional and power emulation offers the advantage of inherent power profile to functional execution trace correspondence, i.e., a power consumption value can be determined for each executed instruction. Furthermore, the emulation is cycle-accurate while still allowing for rapid profiling of long program sequences. This constitutes an advantage over simulation-based approaches that are either lacking simulation detail and hence accuracy or simulation speed.

In contrast to hardware power profile flattening approaches, no additional on-chip measurement and control hardware is required. Furthermore, opposed to power peak reduction methods modifying intermediate language representations of the given software application [2, 3], our approach operates on and modifies the original C or assembler source code. The resulting power peak optimized source code can afterwards still be manually modified by the software engineer if required. In the context of embedded software power peak optimization, the novel contributions of this paper are as follows:

- We present a framework for detecting source code regions causing power peaks by analyzing the power consumption as well as the functional debug information obtained during software execution.
- We derive an optimization algorithm, actuating power management features for these power-critical source code regions and hence reducing the number of power peaks.
- Finally, we illustrate the feasibility of our approach on a power-constrained deep-submicron smart-card controller system.

This paper is structured as follows. In Section 2 we discuss related work on power peak optimization and power profile flattening. Section 3 presents our automated framework for power-critical code region detection and optimization. We illustrate the effectiveness of our approach in Section 4. Finally, conclusions drawn from our current work are summarized in Section 5.

2 Related Work

Due to the large influence of software on both average as well as peak power consumption of embedded systems, numerous works have studied power- and energy-aware software optimization methods. With regard to power-constrained devices, the power profile flattening and the optimization of power consumption peaks, is of increased interest. These power peaks are often caused due to the occurrence of power-critical events during software execution. Especially in battery- and RF-powered devices these peaks can severely impact the power

supply circuit and can lead to supply voltage drops [1]. These supply voltage drops seriously jeopardize the stability and hence the reliability of the given system. Power profile flattening hardware implementations have been studied in the context of security-related applications. In the security domain, the reduction of profile variability is of increased interest as a countermeasure against power analysis attacks [9].

For the purpose of reliability enhancements, the reduction of power peaks has been investigated in [3] by means of a simulation-based peak elimination framework using iterative compilation. Other attempts on power peak reduction have focused on instruction reordering to minimize the switching activity due to circuit state changes [2] as well as non-functional instruction (NFI) insertion [4].

Power profile flattening in security applications, aiming at hindering power analysis attacks by means of NFI insertion, was studied in [5]. Both software and hardware implementations were shown. In [6] a current-injection-based real-time flattening method has been proposed. This approach has been extended in [7] by a voltage scaling capability for improved flattening performance.

3 Automated Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software

Our automated power profiling and power-critical code region detection methodology as depicted in Figure 1 builds upon a *standard software development flow* (A) and our *run-time power profiling* approach (B). The power estimates, alongside with the functional traces are being analyzed to *detect power-critical code regions* (C). After these regions have been detected, an *optimization algorithm* is used to reduce the power consumption and hence the power peaks during these critical code regions (D).

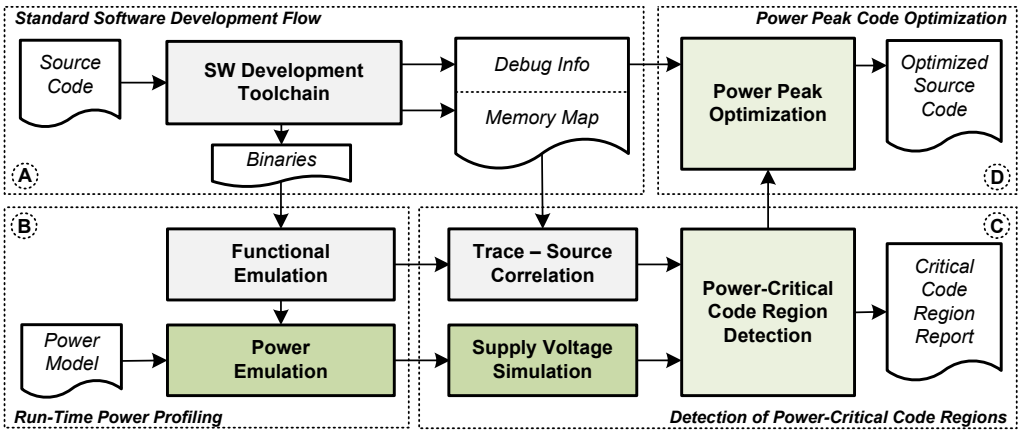

Fig. 1. Automated flow for power profiling, power-critical code region detection and optimization

3.1 Run-Time Power Profiling Based on Power Emulation

For the purpose of detecting power-critical code regions, power profiling of the given software application has to be performed in the first place. In contrast to existing software power peak optimization approaches, we employ the power emulation technique previously introduced in [8] to obtain power profiles for the software application's execution. The principle of power emulation as depicted in Figure 2, is to augment the functionally emulated system-under-test with special power estimation hardware. This power estimation hardware monitors the state of the system and its subcomponents. Based on these state data, the power estimator derives cycle-accurate run-time power estimates according to an integrated high-level power model.

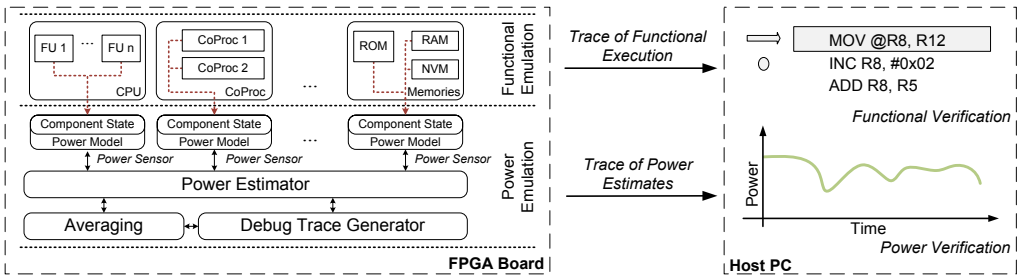

Fig. 2. Embedded software power profiling utilizing power emulation: Run-time power estimation and functional execution trace generation (adapted from [8])

As compared to low-level simulation-based power profiling, the power emulation technique largely reduces profiling time. This allows for the profiling of complex software applications and elaborate program sequences, such as the booting process of an operating system. In contrast to high-level simulators, power emulation offers the benefit of cycle-accuracy that instruction- or system-level-simulators fail to deliver. Furthermore, power emulation offers the advantage of inherent power profile to functional execution trace correspondence as compared to measurement-based approaches.

3.2 Power-Critical Code Region Detection

Our power critical code region detection approach as depicted in Figure 1 consists of multiple stages. First, the functional execution trace obtained in the joint functional and power emulation step is used to establish the source code correlation, i.e., identifying the source code region corresponding to each execution trace message. Second, using the power emulation trace as input data, a supply voltage simulation employing a numerical model of the RF-supply is performed[3]. Third, the resulting supply voltage profile is utilized to identify

[3] Due to the limited computational complexity of the numerical RF-supply model, a simulation-based implementation is adequate.

power peaks leading to critical voltage drops and finding the source code regions causing these drops.

Figure 3 depicts the inductively coupled power supply of a contact-less smart-card device. The impact of power peaks on the supply voltage level, however, is dependent on the duration, power level and rate of these peaks as shown in Figure 4. We define power-critical source code regions as parts of an embedded software application resulting in power peaks that lead to supply voltage drops below a critical limit. These peaks can be caused by, e.g., phases of high processor activity, a number of consecutive memory read or write accesses and co-processor as well power-intensive peripheral activity. In order to identify power peaks that actually lead to critical supply voltage drops on the given system, a supply voltage simulation based on the emulated power profile is performed.

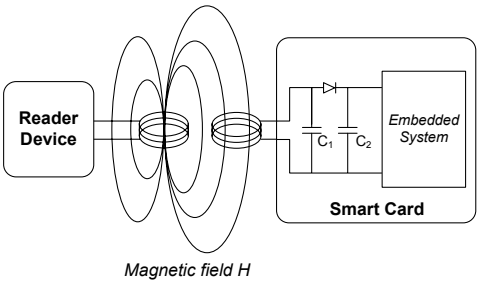

Fig. 3. Inductively coupled power supply of RF-powered smart-card embedded system (adapted from [10])

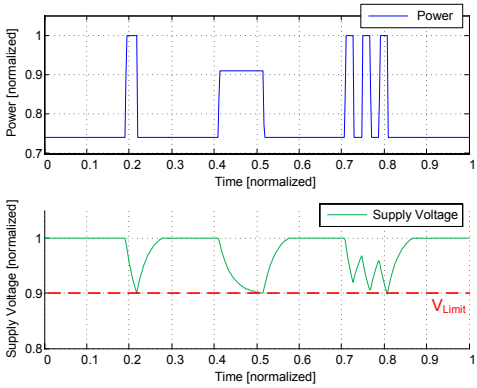

Fig. 4. Impact of different power peaks on the supply voltage (voltage drops)

3.3 Optimization of Power-Critical Source Code Regions

The subsequent power-critical code region optimization algorithm as shown in Algorithm 1 aims at applying code modifications for power peak reduction to the original C or assembler source code. Depending on the power management features available on the given system, the frequency scaling and the NFI insertion techniques are applied to these power-critical regions. Listing 1.1 illustrates the insertion of frequency scaling control instructions around the call-site[4] of a function causing power peaks, whereas Listing 1.2 shows the use of NFI insertion within a loop causing short power peaks.

The algorithm operates in three major stages: (1) The power-critical code regions for each function are determined. If a large part of a function constitutes the power-critical code region, the algorithm chooses to optimize the entire function.

[4] The source code line calling a particular function.

```
start_f_scaling();

power_critical_function();

stop_f_scaling();
```

Listing 1.1. f-scaling example

```
while(loop_condition)
{
   short_loop_instruction;
   nop(); //NFI
}
```

Listing 1.2. NFI insertion example

In this case the call-sites of the function are searched and marked for modification instead of the function itself. (2) Consecutive source code lines marked for modification are grouped into modification clusters. For each of those clusters, the algorithm chooses an optimization strategy based on the cluster's number of power peaks and their respective duration: Short power peaks are likely to be resolved by NFI insertion, longer power peaks or longer groups of peaks can be reduced by applying frequency scaling. (3) Each of the found source code clusters is then modified in the chosen way and the modified code is written back to the source files.

Algorithm 1: Power-Critical Source Code Region Optimization

Input: Set of application source code S, List of power-critical code regions L, Threshold of max. percentage of power-critical lines per function Th_{clpf}, Threshold of f-scaling time penalty $Th_{f-scale}$
Output: Set of optimized application source code S_o
Step 1, group by function:
List of affected source code lines $L_{sl} := \{\}$
foreach *Function f in S* **do**
  Find source code lines of f in L
  if *Found source code lines > 0* **then**
    Calculate percentage of power-critical code region in function
    if *Percentage* $> Th_{clpf}$ **then**
      Find call-sites of function f, add source code lines of call-sites to L_{sl}
    else
      Add source code lines to L_{sl}
Step 2, cluster lines to modify & choose optimization strategy:
L_{slc} := Cluster consecutive source code lines in L_{sl}
foreach *Source code cluster C in L_{slc}* **do**
  if *Duration $C > Th_{f-scale}$* **then**
    Mark cluster C for f-scaling
  else
    Mark cluster C for NFI insertion
Step 3, perform modification:
$S_o := S$
foreach *Source code cluster C in L_{slc}* **do**
  Modify S_o by inserting selected optimization instructions

4 Experimental Results

For evaluating our framework, a smart-card microcontroller test-system supplied by our industrial partner was employed. For different benchmarking applications, power profiles were recorded using the power emulation technique. Afterwards, these benchmarks were optimized both in a manual as well as in an automated way utilizing the presented framework. This allows for evaluating the effectiveness of our method.

4.1 Test System for Power Peak Optimization

The used smart-card microcontroller test system consists of a 16-bit pipelined cache architecture. It comprises volatile and non-volatile memories as well as a number of peripherals, e.g., cryptographic coprocessors, timers, and random number generators. The system has been augmented with a power emulation unit as depicted in Figure 5 to allow for the generation of run-time power estimates.

For detecting power peaks leading to problematic supply voltage drops, we have implemented an RF power supply equivalent circuit model as proposed in [1] and depicted in Figure 6. Based on power consumption changes in the microcontroller test-system, the load current $i_l(t)$ changes and affects the load voltage $v_l(t)$. In phases of high power consumption and thus high load currents when the required load current is higher than the supplied source current $i_s(t)$, the energy storage capacitor delivers the missing fraction $i_c(t)$. However, for longer power peaks or a longer series of short power peaks, the capacitor fails to deliver the required current resulting in a critical supply voltage drop.

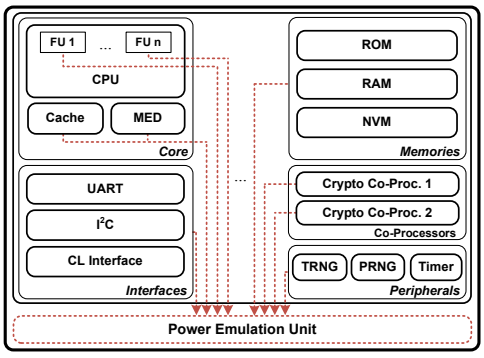

Fig. 5. 16-bit smart-card microcontroller test system augmented by power emulation unit (adapted from [11])

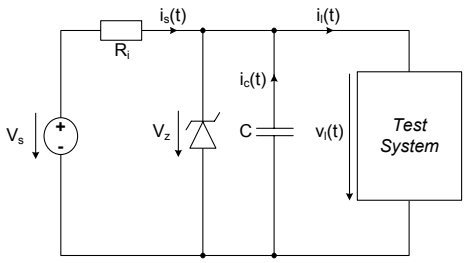

Fig. 6. Equivalent circuit of the RF power supply of the test system (adapted from [1])

4.2 Comparison of Original and Optimized Power Consumption and Supply Voltage Profiles

We illustrate the optimization result by comparing the power consumption and the respective supply voltage profiles of a given software application. Figure 7 resembles the results obtained during profiling of the original application. After the power-critical code region detection and optimization, the power profiling and supply voltage simulation was repeated yielding the profiles depicted in Figure 8.

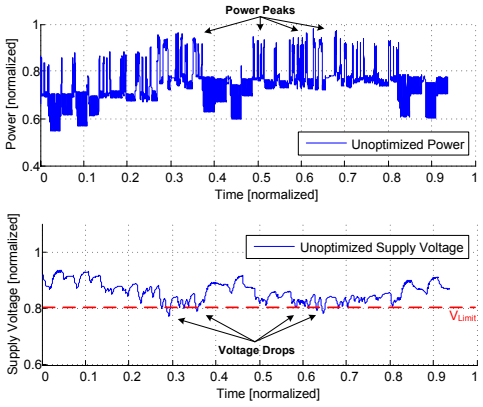

Fig. 7. Unoptimized power consumption and resulting supply voltage profiles of authentication benchmarking application[1]

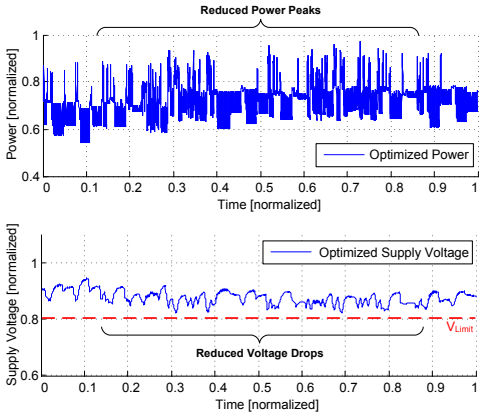

Fig. 8. Optimized power consumption and resulting supply voltage profiles of authentication benchmarking application[1]

The results illustrate how a number of power peaks result in supply voltage drops below the critical limit. By applying frequency scaling and NFI insertion to the code regions causing these peaks, their power consumption and hence their supply voltage impact can be diminished. Note that this modification, while improving system stability and reliability, comes at the cost of a slightly increased execution time. However, as illustrated in the subsequent section, the additionally required execution time is smaller for the automatically than for the manually optimized version because the frequency scaling and the NFI insertion are applied more selectively.

4.3 Impact of Power Peak Optimization on Execution Time and Code Size

We have applied the power peak optimization algorithm to various benchmarking applications in order to evaluate its impact on the execution time and the code

[1] Data normalized due to existing NDA.

size. For comparison we have also manually optimized the given benchmarking applications by applying frequency scaling to the entire benchmark. For both the manual and the automatic approach, all power peaks resulting in critical supply voltage drops have been eliminated. Figure 9 illustrates these results for two general purpose microcontroller benchmarks (Coremark [12] and Dhrystone) as well as for two domain-specific ones (Authenthication and Crypto).

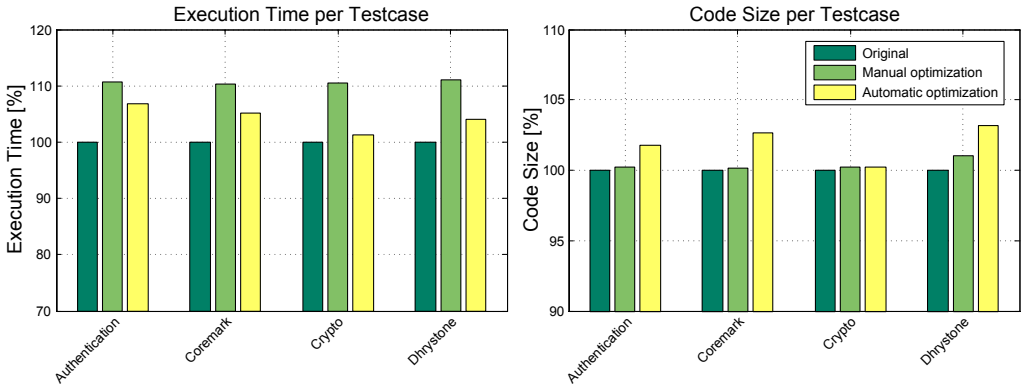

Fig. 9. Execution time and code size of original, manually as well as automatically modified benchmarks[1]

The results show that in terms of execution time the automatic approach outperforms the manual optimization due to the finer granularity of code modifications. For the manual optimization approach the execution time increases by ∼10% due to the minimally required frequency reduction of ∼10% for eliminating all critical supply voltage drops. However, for the automatic approach this increase is in the range of only 1.2% (Crypto) up to 6.8% (Authentication) depending on the number and duration of power peaks. Note that the increase in execution time also depends on the ratio of code regions affected by power peaks that need to be optimized to regions requiring no optimization.

Furthermore, we compare the increase in code size caused by the insertion of frequency scaling control instructions and NFIs. This increase is almost negligible for the manual approach (smaller than or ∼1% for all testcases). For the automatic approach, the increase is slightly higher and in the range of 0.2% (Crypto) up to 3.2% (Dhrystone).

5 Conclusions

The power consumption of embedded systems is to a large extent determined by software applications, actuating power management features as well as controlling the overall system activity. Power peaks, caused by power-critical software

[1] Data normalized due to existing NDA.

events, can seriously impact the supply voltage and lead to critical supply voltage drops. These voltage drops pose a threat to the reliability of power-constrained mobile devices such as RF-powered smart cards.

In this paper we have outlined an automated framework aimed at the power peak detection utilizing the emulation-based power profiling of given embedded software applications. By identifying the software code regions causing power peaks, the framework is able to selectively apply power reduction strategies, such as frequency scaling and non-functional instruction insertion, to the affected regions. Furthermore, we have evaluated the effectiveness of this automated power peak optimization framework on a number of benchmarking applications. For these benchmarks the inherent execution time increase is in the range of only 1.2% up to 6.8% for the automatic modifications as compared to ∼10% for the manual ones.

6 Acknowledgements

We would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology for providing us with funding for the POWERHOUSE project under FIT-IT contract FFG 815193, as well as our industrial partners Infineon Technologies Austria AG and Austria Card GmbH for their enduring support.

References

1. Haid, J., Kargl, W., Leutgeb, T., Scheiblhofer, D.: Power management for RF-powered vs. battery-powered devices. In: TMCS. (2005)
2. Grumer, M., Wendt, M., Steger, C., Weiss, R., Neffe, U., Muehlberger, A.: Automated software power optimization for smart card systems with focus on peak reduction. AICCSA (2007)
3. Grumer, M., Wendt, M., Lickl, S., Steger, C., Weiss, R., Neffe, U., Muehlberger, A.: Software power peak reduction on smart card systems based on iterative compiling. Emerging Directions in Embedded and Ubiquitous Computing (2007)
4. Wendt, M., Grumer, M., Steger, C., Weiss, R., Neffe, U., Muehlberger, A.: System level power profile analysis and optimization for smart cards and mobile devices. In: SAC. (2008)
5. Muresan, R., Gebotys, C.: Current flattening in software and hardware for security applications. In: CODES+ISSS. (2004)
6. Li, X., Vahedi, H., Muresan, R., Gregori, S.: An integrated current flattening module for embedded cryptosystems. In: ISCAS. (2005)
7. Vahedi, H., Muresan, R., Gregori, S.: On-chip current flattening circuit with dynamic voltage scaling. In: ISCAS. (2006)
8. Genser, A., Bachmann, C., Haid, J., Steger, C., Weiss, R.: An emulation-based real-time power profiling unit for embedded software. In: SAMOS. (2009)
9. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO. (1999)
10. Finkenzeller, K.: RFID Handbook. John Wiley & Sons Ltd. (2003)
11. Bachmann, C., Genser, A., Steger, C., Weiss, R., Haid, J.: Automated power characterization for run-time power emulation of soc designs. In: 13th Euromicro DSD. In press. (2010)
12. http://www.coremark.org/

Estimation-Based Run-Time Power Profile Flattening for RF-Powered Smart Card Systems

Andreas Genser[1], Christian Bachmann[1], Christian Steger[1], Reinhold Weiss[1], Josef Haid[2]
[1]Institute for Technical Informatics, Graz University of Technology
{andreas.genser, christian.bachmann, steger, rweiss}@tugraz.at
[2]Design Center Graz, Infineon Technologies AG
josef.haid@infineon.com

***Abstract*—Power-constrained systems, such as RF-powered smart cards are gaining increased significance in the embedded system's domain. These systems are highly susceptible to supply voltage drops caused by power peak regions that impact on the system stability. Power profile flattening mechanisms have emerged as an effective power peak countermeasure to enhance system reliability. In this paper we present a hardware power profile flattening approach by employing system-level DVFS adaptions coupled with a hardware power estimation architecture. The exploitation of hardware-accelerated real-time power estimation techniques replaces costly analog on-chip power measurements and enables the dynamic control of the system's power consumption in a purely digital manner. We demonstrate the effectiveness of our approach by conducting power profiling and voltage drop analysis of a deep-submicron RF-powered smart card system.**

I. Introduction

RF-powered smart cards acquire power via an externally generated RF-field that provides only a limited amount of power dependant on the field strength and the distance between the smart card and the RF-field generating reader device. Power peaks (i.e., regions that exceed the power limit) that are often caused by large power consuming system modules (e.g., cryptographic coprocessors) can cause the supply voltage of the system to drop below a critical limit, in a way that stable system operation can not be guaranteed. A number of countermeasures to ensure reliable system operation despite occurring power peaks have been proposed. These works can be grouped in two categories: (i) *software power profile flattening* and (ii) *hardware power profile flattening*.

Software power profile flattening techniques aim at the insertion of non functional instructions (NFIs) by modifying program code sections causing power peak regions [1], [2]. An automated software power peak optimization method has been proposed in [3] by performing optimizations on the compiler level and by instruction reordering. In [4], an automated framework for power peak optimization has been presented based on NFI insertion and frequency scaling. However, the number of NFIs, their locations or the required system frequency to avoid power peaks is not known during compile time, which might require multiple optimization iterations until satisfying results are achieved.

With the advent of *hardware power profile flattening techniques*, these limitations are circumvented by utilizing dedicated flattening hardware that performs run-time NFI insertion based on power profile analysis. Power profiles are obtained by on-chip analog power measurements, while power constraints are held in registers in a digital manner. This requires D/A converters to perform the decision making whether NFI insertion is necessary [1].

A second variant of *hardware power profile flattening* is done based on *current injection methods* as proposed in [5]. Digital and analog solutions are proposed. Power profile flattening is achieved by forcing redundant switching activity or by adjusting a dedicated current sink depending on the present system's power consumption. In [6], a *current injection method* combined with the scaling of the supply voltage is presented. *Current injection methods* require a current reserve to sustain phases of high power consumption, while keeping the overall power consumption constant. This current reserve reduces system efficiency.

In this paper, we present a hardware power profile flattening technique based on a power estimation architecture that has been proposed in [7]. The availability of real-time power estimates in a purely digital manner replaces costly on-chip analog measurements and D/A conversions, which simplifies the overall system design. We implement dynamic voltage and frequency scaling (DVFS) as a promising power profile flattening alternative to NFI insertion and current injection methods. By varying the system frequency and the supply voltage, the provided power can be exploited more efficiently compared to current injection methods. We demonstrate the effectiveness of our approach for a typical RF-powered smart card system. The system-level implementation of the estimation-based power profile flattening technique accounts only to 2.8% to the total area and to approx. 2% to the overall power consumption of the smart card system.

II. Hardware-Accelerated Power Estimation Prerequisites

Hardware-accelerated power estimation allows to derive cycle-accurate power consumption information in real-time in a purely digital way. In our previous work, we proposed a hardware-accelerated power estimation approach that is implemented on system-level to keep hardware and power overhead low, while still providing high estimation accuracies [7]. The basic essentials of this work are discussed within this section.

A. Power Model

Power models intended for hardware integration typically operate on high abstraction layers (e.g., system-level) and are frequently based on linear regression methods [8]. A linear regression model can be established by considering a linear combination of a number of model parameters x_i and model coefficients c_i yielding the estimate $\hat{y}$. This can be given as

$$\hat{y} = c_0 + \sum_{i=1}^{n} c_i x_i + \epsilon. \quad (1)$$

$\mathbf{x} = [x_1, x_2, ...x_n]$ gives the vector of model parameters, each of which representing system states, such as CPU operating modes, memory accesses or coprocessor activity, etc. The entirety of model coefficients can be written as $\mathbf{c} = [c_1, c_2, ...c_n]^T$. Each of the model coefficients c_i contain power consumption information corresponding to a system state x_i. c_0 resembles a special model coefficient that represents the leakage power consumption of the system. All model coefficients in $\mathbf{c}$ are determined by a power model characterization process [9]. The evaluation of (1) finally gives the power estimate $\hat{y}$, which deviates from the true power value y by the estimation error ϵ. If computed in hardware, the model can be evaluated on a clock cycle basis allowing for real-time power consumption information derivation.

B. Power Estimation Architecture

The power estimation architecture as illustrated in Fig. 1 implements the power model established according to (1) and holds power model coefficients $\mathbf{c}$ determined during the power characterization process. Power sensors implement lookup tables that map system state information x_i towards model coefficients c_i. Each of the sensors observes system states of a system component and derives its power consumption. The power accumulation unit sums up power consumption information from the power sensors and assembles cycle-accurate power samples $P(\mathbf{x}(t))$.

Power model evaluations in hardware benefit from a huge speed up over low-level power simulations (e.g., gate-level simulations) in terms of simulation time. As reported in [7], speedups of several orders of magnitude could be achieved compared to gate-level simulations.

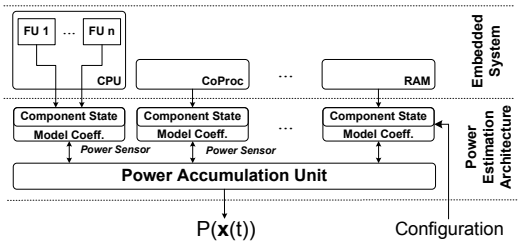

Fig. 1. Power estimation architecture, power consumption information are derived from system state information by power sensors. Obtained and modified from [7]

III. Estimation-Based Power Profile Flattening

A system overview of our proposed estimation-based hardware power profile flattening architecture is given in Fig. 2.

A. DVFS Power Scaling Extensions

The DVFS power scaling extensions block solves the issue that power consumption information $P(\mathbf{x}(t))$ lacks the dependency from $f(t)$ and $V_{DD}(t)$. $P(\mathbf{x}(t))$ provides power consumption information for a reference system frequency and a reference supply voltage at which the power characterization process has been performed. DVFS power scaling extensions as depicted in Fig. 3, extend $P(\mathbf{x}(t))$ with $f(t)$ and $V_{DD}(t)$ information yielding the real power consumption of the system $P(\mathbf{x}(t), f(t), V_{DD}^2(t))$. The DVFS power scaling extensions block consists of a configurable lookup table (LUT) providing $V_{DD}(t)$ information depending on the system frequency $f(t)$. These (f, V_{DD})-pairs correspond to the supported f- and V_{DD}-settings of the considered system. The scaling unit conducts the multiplication of $P(\mathbf{x}(t))$, $f(t)$ and $V_{DD}(t)$ according to (2).

$$P(\mathbf{x}(t), f(t), V_{DD}^2(t)) = P(\mathbf{x}(t)) \cdot f(t) \cdot LUT^2(f(t)) \quad (2)$$

B. Power Profile Flattening Controller

The actual power profile flattening is performed by the power profile flattening controller. The present power consumption $P_t = P(\mathbf{x}(t), f(t), V_{DD}^2(t))$ is compared to given power constraints P_c. The discrepancy between P_c and P_t is given as P_ϵ, which determines the decision criteria whether the system frequency $f(t)$ is to be increased or decreased. By performing DVFS adaptions, the employed power profile flattening policy aims at minimizing P_ϵ according to (3).

$$FlatteningPolicy : \min\{P_\epsilon\} = \min\{\|P_t - P_c\|\} \quad (3)$$

Fig. 4 depicts the state-machine of a greedy power profile flattening policy. Starting from the $Init$-state, a state transition

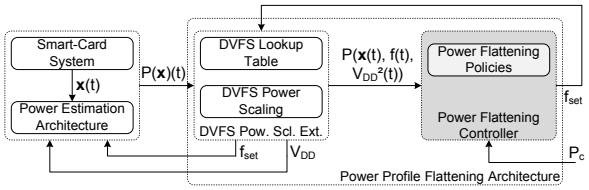

Fig. 2. Power profile flattening concept based on the power estimation architecture, the power profile flattening architecture incorporating the power profile flattening controller and DVFS power scaling extensions

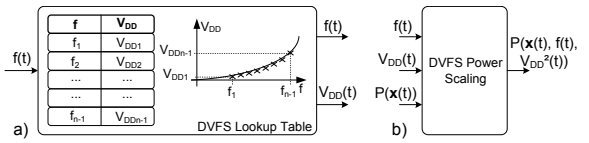

Fig. 3. DVFS power scaling extensions that incorporate a) system frequency and supply voltage dependency and b) a power scaling unit

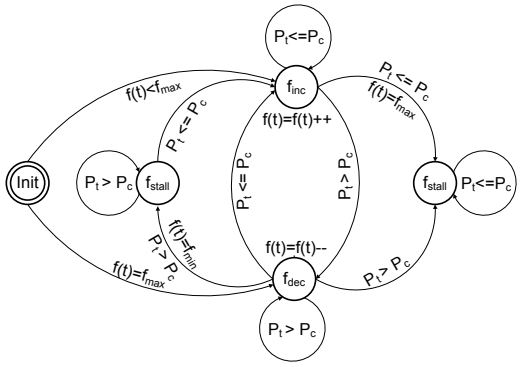

Fig. 4. System-level power profile flattening state-machine based on a greedy flattening policy

to f_{dec} or f_{inc} depending on the present value of $f(t)$ is performed. In state f_{inc}, $f(t)$ is increased as long as the present power consumption P_t is less than or equal to P_c, else the state-machine enters state f_{dec} and $f(t)$ is decreased to lower the system's power consumption. If the minimum frequency f_{min} or the maximum frequency f_{max} is reached, two f_{stall} states exist to prevent f from running out of bounds. Note that each f-adaption outputs a new frequency set value f_{set} at the power profile flattening controller causing an according lookup in the DVFS lookup table implemented in the DVFS scaling extensions block. Both, f_{set} and the corresponding supply voltage V_{DD} steer the system's operating point in a way that the power profile is flattened.

IV. Case Study: Power Profile Flattening for an RF-powered Smart Card System

We demonstrate the results of the proposed estimation-based power profile flattening approach based on an RF-powered smart card system that is highly susceptible to power peaks. If power peaks occur, the source voltage of the RF energy source might drop below a critical limit, which leads to power shortages at the smart card system.

To reproduce the behavior of the RF energy source, we exploit SPICE simulations of *LTSpice IV* [10] based on an RF energy source equivalent circuit [11]. By means of a typical smart card application, results of the power profile flattening approach are illustrated.

A. System Setup

Fig. 5 illustrates the coupling of the reader device to a contact-less smart card via an RF-field. The available power at the smart card device is dependent on the strength of the magnetic RF-field H as well as the distance between the reader and the smart card. Moreover, Fig. 5 illustrates the structure of a typical contact-less smart card system. It is based on a 16-bit pipelined cache architecture incorporating a memory encryption-decryption (MED) unit and volatile and non-volatile memories. Communication interfaces, such as UART, I2C or a contact-less interface are provided. Moreover, coprocessors to accelerate symmetric and asymmetric cryptographic algorithms as well as peripherals such as random

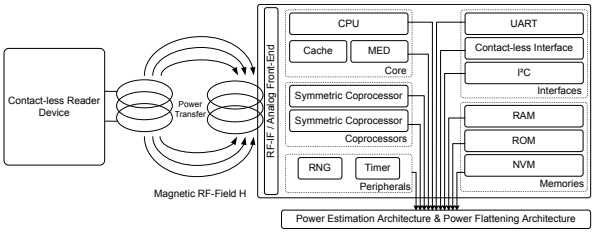

Fig. 5. Block diagram of a 16-bit contact-less smart card system incorporating coprocessors to enhance system performance for cryptographic algorithms

number generators (RNG) or timers are built in. An analog front-end powers system components with energy gathered from the RF-field. System state information are provided to the power estimation architecture by routing power-relevant signals that have been determined during the power characterization process.

In order to apply DVFS adaptions for power profile flattening, the lookup table in the DVFS power scaling extensions block has to be set up based on the frequency-supply voltage relationship of the given smart card system. The supported (f, V_{DD})-pairs are in the range of 1-38MHz and 0.5-1.7V, and frequency and supply voltage steps are 1MHz and 0.1V, respectively. The smart card system incorporating the power estimation architecture and the power profile flattening architecture has been synthesized on an *Altera StratixII FPGA*.

B. Power Profile Flattening Results

A typical smart card application to obtain system power profiles has been executed for the following scenarios:

- Scenario 1: Standard configuration without power profile flattening hardware
- Scenario 2: Power profile flattening configuration incorporating the power estimation and the power profile flattening architecture exploiting a *greedy flattening policy*. Power from the RF-field is constraint to $P_c = 0.4$.

Fig. 6 and 7 show power profile flattening results of Scenario 1 and 2, respectively. The power profile for a typical smart card application is depicted in Scenario 1 in the upper subplot of Fig. 6. During the occurrence of *Peak 1* and *Peak*

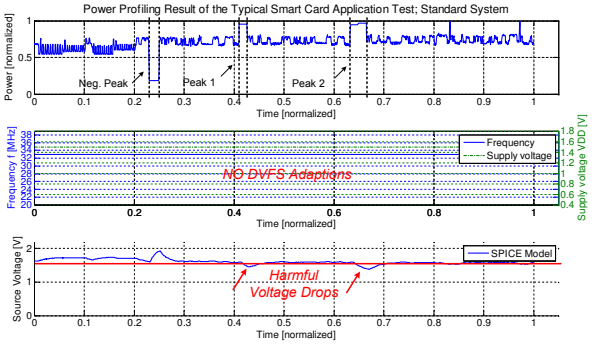

Fig. 6. Scenario 1: Smart card application power profile, f and V_{DD}, as well as the behavior of the RF energy source voltage without power profile flattening (data have been normalized for confidentiality reasons)

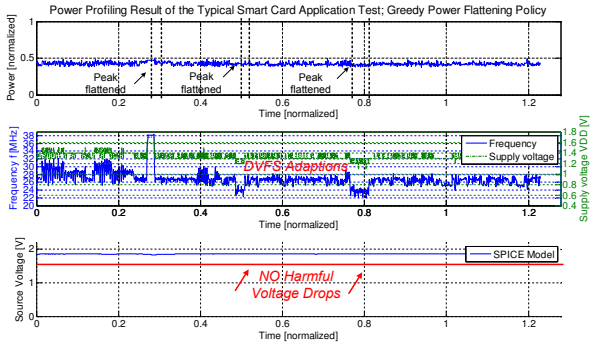

Fig. 7. Scenario 2: Smart card application power profile, f and V_{DD}, as well as the behavior of the RF energy source voltage with power profile flattening by the *greedy policy* (data have been normalized for confidentiality reasons)

2, the CPU and the coprocessor are working simultaneously, while during the *negative peak* phase the CPU is put to sleep mode and the coprocessor is active. The standard implementation executes the smart card application at a system frequency of f=33MHz and a supply voltage of V_{DD}=1.5V (see second subplot in Fig. 6). The bottom subplot of Fig. 6 illustrates the source voltage drop simulated by *LTSpice IV*. It can be clearly observed that power peaks caused by coprocessor activity make the source voltage to drop below the critical limit of V_{DD}=1.5V.

The achieved power profile flattening results of Scenario 2 are illustrated in Fig. 7. Power peaks caused by the coprocessor are flattened by adapting the system frequency f and the supply voltage V_{DD} (see middle subplot of Fig. 7). As a consequence, the source voltage stays above the critical voltage limit. Moreover, the 'negative' power peak phase is flattened by speeding up the system to f=38MHz and V_{DD}=1.7V. This behavior exploits the available power more efficiently compared to previously proposed works based on NFI insertion or current injection methods that can not accelerate system execution.

Tab. I gives a general overview of the average to peak power (A2P) and algorithm execution time (ET) for different applications. A comparison between Scenario 1 and Scenario 2 shows that the average to peak power is reduced between 42% and 71% for the given applications, which on the other hand comes at the cost of an execution time increase of 17-23%.

TABLE I
COMPARISON OF AVERAGE TO PEAK POWER (A2P) IN % AS WELL AS EXECUTION TIME (ET) IN % FOR DIFFERENT EXECUTED ALGORITHMS (ALL VALUES NORMALIZED FOR CONFIDENTIALITY REASONS)

	Smart card appl.		Dhrystone		CPU	
	A2P	ET	A2P	ET	A2P	ET
Scen.1	100	100	100	100	100	100
Scen.2	29	123	58	122	47	117

C. Area and Power Overhead

Synthesized on an *Altera StratixII FPGA*, the entire system allocates 84785 ALUTs, while power estimation and power profile flattening hardware occupy 2416 ALUT resources. Hence, hardware extensions contribute to 2.8% to the total system area.

TABLE II
NORMALIZED AVERAGE POWER OF THE POWER PROFILE FLATTENING HARDWARE (PPF-HW) COMPARED TO THE CURRENT FLATTENING CIRCUIT PROPOSED IN [6]

	Sys.	Sys. + PPF-HW	Overhead (%)
Our architecture	100	102	**2**
Vahedi et al. [6]	100	122	**22**

[1] Power estimates have been derived from *Altera PowerPlay*. Data have been normalized for confidentiality reasons.

As depicted in Tab. II, the proposed estimation-based power profile flattening architecture accounts only to approx. 2 % to the overall power consumption of the system. In contrast, the current flattening technique proposed in [6] performs the flattening at an average power overhead of approx. 22 %.

V. CONCLUSION

RF-powered smart cards rely on energy sources that provide power at a high variability over time. This yields a high susceptibility of these systems to supply voltage drops caused by power shortages.

The proposed estimation-based power profile flattening approach aims at minimizing power peaks by employing DVFS system adaptions based on a greedy power profile flattening policy. For a typical smart card application executed on a 16-bit RF-powered smart card system, power peak reductions are achieved in a way that harmful voltage drops can be avoided. The hardware overhead for power profile flattening hardware extensions contributes only to 2.8% to the overall system area and adds approx. 2% of power overhead. The presented work provides an effective run-time countermeasure against harmful power peaks that can significantly enhance system reliability.

REFERENCES

[1] R. Muresan and C. Gebotys. Current flattening in software and hardware for security applications. In *CODES+ISSS*, 2004.
[2] M. Wendt, M. Grumer, C. Steger, R. Weiss, U. Neffe, and A. Muehlberger. System level power profile analysis and optimization for smart cards and mobile devices. In *SAC*, 2008.
[3] M. Grumer, M. Wendt, C. Steger, R. Weiss, U. Neffe, and A. Muhlberger. Automated software power optimization for smart card systems with focus on peak reduction. In *AICCSA*, 2007.
[4] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid. An automated framework for power-critical code region detection and power peak optimization of embedded software. In *PATMOS*, 2010.
[5] X. Li, H. Vahedi, R. Muresan, and S. Gregori. An integrated current flattening module for embedded cryptosystems. In *ISCAS*, 2005.
[6] H. Vahedi, R. Muresan, and S. Gregori. On-chip current flattening circuit with dynamic voltage scaling. In *ISCAS*, 2006.
[7] A. Genser, Ch. Bachmann, J. Haid, Ch. Steger, and R. Weiss. An Emulation-Based Real-Time Power Profiling Unit for Embedded Software. In *SAMOS*, 2009.
[8] A. Bogliolo, L. Benini, and G. De Micheli. Regression-based RTL power modeling. In *ACM Trans. on Design Autom. of Elect. Sys.*, volume 5, 2000.
[9] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid. Automated power characterization for run-time power emulation of soc designs. In *DSD*, 2010.
[10] Linear Technology, LTSpice IV. (http://www.linear.com/designtools/software/), April 2010.
[11] Josef Haid, Walter Kargl, Thomas Leutgeb, and Dietmar Scheiblhofer. Power management for rf-powered vs. battery-powered devices, 2005.

Bibliography

[1] G. E. Moore, "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.," *Solid-State Circuits Newsletter, IEEE*, vol. 20, no. 3, pp. 33 –35, 2006.

[2] J. Rabaey, "A brand new wireless day," in *Asia and South Pacific Design Automation Conference (ASPDAC '08)*, 2008.

[3] H. Varadarajan, V. Tiwari, R. Patel, H. Ramamurthy, S. Jamshidi, S. Jariwala, and W. Jiang, "Low-Power Design Issues," in *The Computer Engineering Handbook* (V. G. Oklobdzija, ed.), CRC Press, 2002.

[4] G. De Micheli and R. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349 –365, 1997.

[5] C. Piguet, "History of Low-Power Electronics," in *Low-Power Electronics Design* (C. Piguet, ed.), CRC Press, 2005.

[6] C. H. Gebotys, "Low-Power Software Techniques," in *Low-Power Electronics Design* (C. Piguet, ed.), CRC Press, 2005.

[7] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and S. Kaijian, *Low Power Methodology Manual*. Springer, 2007.

[8] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *International Symposium on Low Power Electronics and Design (ISLPED '04)*, pp. 32 – 37, 2004.

[9] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: pushing power management to logic synthesis/design," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 17, pp. 1051 –1060, Oct. 1998.

[10] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *Solid-State Circuits, IEEE Journal of*, vol. 27, pp. 473 –484, Apr. 1992.

[11] "The International Technology Roadmap for Semiconductors (ITRS), 2010 Update, System Drivers," 2010. http://www.itrs.net/Links/2010ITRS/Home2010.htm. Last accessed 2010-01-18.

[12] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*. Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2008.

[13] L. Zhao, R. Iyer, J. Moses, R. lllikkal, S. Makineni, and D. Newell, "Exploring Large-Scale CMP Architectures Using ManySim," *Micro, IEEE*, vol. 27, no. 4, pp. 21 –33, 2007.

[14] J. Coburn, S. Ravi, and A. Raghunathan, "Power emulation: a new paradigm for power estimation," in *42nd Design Automation Conference (DAC '05)*, 2005.

[15] J. Coburn, S. Ravi, and A. Raghunathan, "Hardware accelerated power estimation," in *Design, Automation and Test in Europe (DATE '05)* (S. Ravi, ed.), pp. 528–529 Vol. 1, 2005.

[16] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation," in *International Symposium on Low Power Electronics and Design (ISLPED '08)*, 2008.

[17] "Keil® μVision® IDE." http://www.keil.com/uvision/. Last accessed 2010-02-22.

[18] "Eclipse® Language IDE." http://www.eclipse.org/. Last accessed 2010-02-22.

[19] C. Bachmann, A. Genser, J. Hulzink, M. Berekovic, and C. Steger, "A Low-Power ASIP for IEEE 802.15.4a Ultra-Wideband Impulse Radio Baseband Processing," in *Design, Automation and Test in Europe (DATE '09)*, 2009.

[20] C. Bachmann and A. Genser, "POWERHOUSE - Power-Aware, Hardware-Supported Operating System and Ubiquitous Application Software Development Environment, Technical Report," *Institute for Technical Informatics, Graz University of Technology*, 2011.

[21] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," in *International Conference on Computer-Aided Design (ICCAD '94)* (S. Malik, ed.), pp. 384–390, 1994.

[22] V. Tiwari, S. Malik, A. Wolfe, and M. Tien-Chien Lee, "Instruction level power analysis and optimization of software," in *International Conference on VLSI Design* (S. Malik, ed.), pp. 326–328, 1996.

[23] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '99)* (M. Satyanarayanan, ed.), pp. 2–10, 1999.

[24] Texas Instruments, "Analyzing Target System Energy Consumption in Code Composer Studio IDE," *Tech. Rep.*, 2002.

[25] "Hitex® PowerScale®." http://www.hitex.com/. Last accessed 2010-02-22.

[26] H. F. Hamann, A. Weger, J. A. Lacey, Z. Hu, P. Bose, E. Cohen, and J. Wakil, "Hotspot-Limited Microprocessors: Direct Temperature and Power Distribution Measurements," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 56–65, 2007.

[27] F. J. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau, "Power model validation through thermal measurements," in *International Symposium on Computer Architecture (ISCA '07)*, ISCA '07, (New York, NY, USA), pp. 302–311, ACM, 2007.

[28] W. Nebel and D. Helms, "High-level power estimation and analysis," in *Low-Power Electronics Design* (C. Piguet, ed.), CRC Press, 2005.

[29] "Synopsys®, Inc." http://www.synopsys.com/. Last accessed 2010-02-08.

[30] "Magma® Design Automation, Inc." http://www.magma-da.com/. Last accessed 2010-02-08.

[31] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "An instruction-level energy model for embedded vliw architectures," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 21, pp. 998 – 1010, Sept. 2002.

[32] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "The design and use of simplepower: a cycle-accurate energy estimation tool," in *Design Automation Conference (DAC '00)*, pp. 340 –345, 2000.

[33] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *International Symposium on Computer Architecture (ISCA '00)*, pp. 83–94, 2000.

[34] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *SIGARCH Comput. Archit. News*, vol. 25, pp. 13–25, June 1997.

[35] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol. 35, pp. 59 –67, 2002.

[36] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proc. of the 9th ACM SIGOPS European workshop*, 2000.

[37] R. Joseph and M. Martonosi, "Run-time power estimation in high performance microprocessors," in *International Symposium on Low Power Electronics and Design (ISLPED '01)*, 2001.

[38] G. Contreras and M. Martonosi, "Power prediction for intel XScale processors using performance monitoring unit events," in *International Symposium on Low Power Electronics and Design (ISLPED '05)*, 2005.

[39] J. Haid, G. Kaefer, C. Steger, and R. Weiss, "A co-processor for real-time energy estimation of system-on-a-chip," in *Midwest Symposium on Circuits and Systems (MWSCAS '02)*, 2002.

[40] J. Haid, G. Kaefer, C. Steger, and R. Weiss, "Run-time energy estimation in system-on-a-chip designs," in *Asia and South Pacific Design Automation Conference (ASP-DAC '03)* (G. Kaefer, ed.), pp. 595–599, 2003.

[41] J. Peddersen and S. Parameswaran, "Clipper: Counter-based low impact processor power estimation at run-time," in *Asia and South Pacific Design Automation Conference (ASP-DAC '07)* (S. Parameswaran, ed.), pp. 890–895, 2007.

[42] J. Peddersen and S. Parameswaran, "Low-Impact Processor for Dynamic Runtime Power Management," *Design & Test of Computers, IEEE*, vol. 25, pp. 52–62, 2008.

[43] D. Atienza, P. G. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. M. Mendias, "A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip," in *43rd ACM/IEEE Design Automation Conference (DAC '06)*, 2006.

[44] M. Ghodrat, K. Lahiri, and A. Raghunathan, "Accelerating System-on-Chip Power Analysis Using Hybrid Power Estimation," in *Design Automation Conference (DAC '07)*, 2007.

[45] Aeroflex Gaisler AB, "LEON3/GRLIB SoC IP Library Product Brief," 2008. http://www.gaisler.com/doc/Leon3%20Grlib%20folder.pdf. Last accessed 2010-02-11.

[46] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50 –58, 2002.

[47] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC," *J. VLSI Signal Process. Syst.*, vol. 41, no. 2, pp. 169–182, 2005.

[48] J. Cong, K. Gururaj, G. Han, A. Kaplan, M. Naik, and G. Reinman, "MC-Sim: an efficient simulation tool for MPSoC designs," in *International Conference on Computer-Aided Design (ICCAD '08)*, pp. 364–371, 2008.

[49] P. Gerin, M. M. Hamayun, and F. Pétrot, "Native MPSoC co-simulation environment for software performance estimation," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '09)*, pp. 403–412, 2009.

[50] P. Del Valle, D. Atienza, I. Magan, J. Flores, E. Perez, J. Mendias, L. Benini, and G. De Micheli, "Architectural Exploration of MPSoC Designs Based on an FPGA Emulation Framework," in *Conference on Design of Circuits and Integrated Systems (DCIS '06)*, pp. 12–18, 2006.

[51] D. Atienza, P. G. Della Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, J. M. Mendias, and R. Hermida, "HW-SW Emulation Framework for Temperature-Aware Design in MPSoCs," *ACM Transactions on Design Automation of Electronic Systems TODAES*, vol. Vol. 12, pp. pp. 1 – 26,, 2007.

[52] P. Meloni, S. Secchi, and L. Raffo, "An FPGA-Based Framework for Technology-Aware Prototyping of Multicore Embedded Architectures," *IEEE Embedded Systems Letters*, 2010.

[53] J. Haid, W. Kargl, T. Leutgeb, and D. Scheiblhofer, "Power Management for RF-Powered vs. Battery-Powered Devices," in *TMCS*, 2005.

[54] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO*, 1999.

[55] M. Grumer, M. Wendt, C. Steger, R. Weiss, U. Neffe, and A. Muehlberger, "Automated Software Power Optimization for Smart Card Systems with Focus on Peak Reduction," *International Conference on Computer Systems and Applications (AICCSA '07)*, 2007.

[56] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: an overview," in *Symposium Low Power Electronics, Digest of Technical Papers* (S. Malik, ed.), pp. 38–39, 1994.

[57] M. Grumer, M. Wendt, S. Lickl, C. Steger, R. Weiss, U. Neffe, and A. Muehlberger, "Software Power Peak Reduction on Smart Card Systems Based on Iterative Compiling," *Emerging Directions in Embedded and Ubiquitous Computing*, 2007.

[58] M. Wendt, M. Grumer, C. Steger, R. Weiss, U. Neffe, and A. Muehlberger, "System level power profile analysis and optimization for smart cards and mobile devices," in *ACM Symposium on Applied Computing (SAC '08)*, 2008.

[59] R. Muresan and C. Gebotys, "Current flattening in software and hardware for security applications," in *International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS '04)*, 2004.

[60] X. Li, H. Vahedi, R. Muresan, and S. Gregori, "An integrated current flattening module for embedded cryptosystems," in *International Symposium on Circuits and Systems (ISCAS '05)*, 2005.

[61] H. Vahedi, R. Muresan, and S. Gregori, "On-chip current flattening circuit with dynamic voltage scaling," in *International Symposium on Circuits and Systems (ISCAS '06)*, 2006.

[62] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Automated Power Characterization for Run-Time Power Emulation of SoC Designs," in *Proceedings of the 13th Euromicro Conference on Digital System Design (DSD '10)*, pp. 587–594, 2010.

[63] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Accelerating Embedded Software Power Profiling Using Run-Time Power Emulation," in *Power and Timing Modeling, Optimization and Simulation, 19th International Workshop, PATMOS '09*, 2009.

[64] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software," in *Power and Timing Modeling, Optimization and Simulation, 20th International Workshop, PATMOS '10*, pp. 11–20, 2010.

[65] Infineon Technologies AG, "SLE 78CLX1440P - Short Product Overview," 2010. http://www.infineon.com/. Last accessed 2010-02-11.

[66] "SnapGear Embedded Linux Distribution." http://www.snapgear.org/. Last accessed 2010-02-08.

[67] M. Mamidipaka and N. Dutt, "eCACTI: An enhanced power estimation model for on-chip caches," *Technical Report TR-04-28, CECS, UCI*, 2004.

[68] "CoreMark an EEMBC benchmark." http://www.coremark.org/. Last accessed 2010-02-08.

[69] "Hitex® HiTOP® IDE/Debugger." http://www.hitex.com/. Last accessed 2010-02-22.

[70] "POWERMODES - Power Emulator and Model Based Dependability and Security Evaluation Platform, Project Proposal," *Institute for Technical Informatics, Graz University of Technology*, 2010.

[71] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *International Symposium on Computer Architecture (ISCA '03)*, pp. 2 – 13, 2003.