Accurate and Fast System-Level Power Modeling: An XScale-Based Case Study

ANKUSH VARMA
University of Maryland and Intel Research Labs
ERIC DEBES, IGOR KOZINTSEV, and PAUL KLEIN
Intel Research Labs
and
BRUCE JACOB
University of Maryland

Accurate and fast system modeling is central to the rapid design space exploration needed for embedded-system design. With fast, complex SoCs playing a central role in such systems, system designers have come to require MIPS-range simulation speeds and near-cycle accuracy. The sophisticated simulation frameworks that have been developed for high-speed system performance modeling do not address power consumption, although it is a key design constraint. In this paper, we define a simulation-based methodology for extending system performance modeling frameworks to also include power modeling. We demonstrate the use of this methodology with a case study of a real, complex embedded system, comprising the Intel XScale embedded microprocessor, its WMMX SIMD co processor, L1 caches, SDRAM, and the on-board address and data buses. We describe detailed power models for each of these components and validate them against physical measurements from hardware, demonstrating that such frameworks enable designers to model both power and performance at high speeds without sacrificing accuracy. Our results indicate that the power estimates obtained are accurate within 5% of physical measurements from hardware, while simulation speeds consistently exceed a million instructions per second (MIPS).

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids—*Simulation*; C.0 [**Computer Systems Organization**]: *System Architectures*

General Terms: Design; Experimentation; Measurement; Performance

Additional Key Words and Phrases: Power modeling; embedded systems; SystemC

ACM Reference Format:

Varma, A., Debes, E., Kozintsev, I., Klein, P., and Jacob, B. 2008. Accurate and fast system-level power modeling: An XScale-based case study. ACM Trans. Embedd. Comput. Syst. 7, 3,

This work was done by Ankush Varma as an intern at the Intel Labs, Santa Clara, CA.

Authors' address: Ankush Varma and Bruce Jacob; University of Maryland, College Park, Maryland 20740; email: ankush@eng.umd.edu. Eric Debes, Igor Kozintsev, and Paul Klein, Intel Research Labs, Santa Clara, California 95052; email: eric.debes@intel.com.

Article 25 (April 2008), 20 pages. DOI = 10.1145/1347375.1347378 http://doi.acm.org/10.1145/1347375.1347378

1. INTRODUCTION

As platforms and SoC designs become increasingly complex, high-speed system-level simulation has become an indispensable design tool. Exploration of the large design spaces involved often depends upon the ability to assess each design quickly, facilitating the efficient iteration over numerous candidate designs. To be truly useful, this evaluation should include the major design criteria, including performance and power consumption.

Our research aims to provide a fast, accurate, and practical power estimation tool that can work with a variety of performance estimation frameworks. Toward this end, we present a methodology aimed at a system designer assembling common well-characterized components (such as processors, caches, buses, SRAM, DRAM, peripherals, etc.) into a system, either as discrete components or IP cores. We define a software architecture for power modeling that can easily plug into existing execution-driven simulation infrastructures, including, but not limited to, SystemC. Each component's power consumption is characterized using short instruction sequences to find the values of the parameters in its power model. Since these are extremely short sequences, they can be run either on hardware (such as by done by Tiwari et al. [1994] for microprocessors), or on RTL or low-level power models (as suggested by Givargis et al. [2000b]). As a case study, we apply this methodology to a real nontrivial embedded system based on the Intel XScale microprocessor. We build parameterized SystemC-based power models of a variety of system components and integrate them with a SystemC-based performance modeling framework. These include a detailed power model of the Intel XScale microprocessor [Intel 2004] (including previously unobserved effects), L1 caches, the XScale's Wireless MMX (WMMX) SIMD coprocessor [Paver et al. 2004], SDRAM memory, and system buses. To allow this approach to be used elsewhere, we provide details of the software architecture used, including the interfaces and data structures required for implementation.

Finally, we validate our models and methodology by comparing the simulation-based power estimates obtained against physical measurements on specially-instrumented hardware, while running applications on a real operating system (Windows CE, in this case). We find that our approach allows both high-speed simulation and accurate power estimation. Integrating our power models into an existing simulation infrastructure (simulating at 1 MIPS[1]) caused no appreciable slowdown in simulation speed, yet provided power estimates that were accurate to within 5% of physical measurements. In addition, simulation-based power modeling also exposes fine-grained power behavior that cannot be revealed by physical measurements.

[1]It must be noted that there is a difference between MIPS and MHz in simulations, since an instruction can take more than one clock cycle, espe-cially when there is a memory stall. We report MIPS because it is the *lower* of the two and it is more pertinent to instruction-based simulation.

To the best of our knowledge, this is the first system description language (SDL)-based approach to validate the accuracy of the power estimates obtained against physical measurements from a real system. We are also the first to report power-enabled simulation speeds over 1 million instructions per second (MIPS). We simulate realistic lengths of time (a complete Windows CE OS boot and application run over billions of clock cycles) and use large, realistic benchmarks. The power estimates obtained are accurate within 5% of actual physical measurements from hardware. The power modeling and characterization techniques we use here were applied to a SystemC-based simulation infrastructure but are applicable to any execution-driven simulation framework.

The contributions of the presented work include:

— Realistic validation of a system-level execution-driven power modeling approach against physical hardware.
— Detailed power models of various components, including an accurate instruction-level power model of the XScale processor.
— A scalable, efficient and validated methodology for incorporating fast, accurate power-modeling capabilities into system description languages, such as SystemC. Major steps including characterization, modeling, validation, and software architecture are all discussed in detail.

The rest of this article is structured as follows. Section 2 provides an overview of various approaches to power modeling and a discussion of related work in literature. Section 3 contains an overview of SystemC and transaction-level modeling. Section 4 provides an overview of the proposed methodology, including characterization and modeling methods and software architecture. Section 5 provides detailed power models for each component studied: the XScale microprocessor, the WMMX SIMD coprocessor, address and data buses, caches, and SDRAM. Section 6 describes the experimental setup used for validation of the system-level power models against hardware and Section 7 describes the results obtained. Finally, conclusions, as well as directions for future work, are discussed in Section 8.

2. RELATED WORK

As power is a growing concern for a wide variety of systems, a range of tools have been built that address the spectrum of issues, design stages, and levels of abstraction involved. HDL/RTL-based tools allow cycle-accurate performance simulation, and accurate power modeling, but simulate many orders of magnitude too slowly to permit the modeling of realistic system-level workloads.[2] Layout-level tools are even slower and are not suitable for any system-level design space exploration. Microarchitectural simulators, such as SimpleScalar [Austin et al. 2002], allow cycle-accurate microprocessor performance modeling. Microarchitectural power-modeling tools [Brooks et al. 2000; Chen et al. 2001;

[2]At typical HDL simulation speeds of 100 Hz, modeling the billion or so cycles required to boot Windows CE on a 403 MHz XScale would take approximately 4 months.

Contreras et al. 2004; Ye et al. 2000] can often work in conjunction with such tools. These can run 10–100× faster than RTL and maintain reasonable accuracy. However, they require detailed microarchitectural information about each component being modeled, which may not always be available to a designer, especially in the case of proprietary third-party IP cores. For microprocessors, in particular, instruction-level power modeling can be used to speed things up [Sinha and Chandrakasan 2001; Tiwari et al. 1994; 1996] and this can often be coupled with a fast instruction-based timing simulator or traces from a simulation run. This approach has been successful in modeling embedded microprocessors, since they can be constructed without knowledge of the processor microarchi-tecture. However, this still leaves the system designer with *ad-hoc* models for nonmicroprocessor components. Software developers often also use functional simulators—fast instruction-level simulators that discard all timing information to achieve very high simulation speed.

A recent advance in system design has been the use of higher-level languages and tools for expressing hardware constructs. These include SystemVerilog [Rich 2004], SystemC [Grotker et al. 2002], SpecC [Fujita and Ra 2001], and Handel-C [Loo et al. 2002], among others [Habibi and Tahar 2003]. Such system description languages (SDLs) can characterize designs at a variety of levels of abstraction, enabling extremely compact system descriptions early in the design flow, and fast simulation without compromising accuracy [Jayadevappa et al. 2004]. In particular, SystemC, a C++-based library that provides a variety of hardware-oriented constructs and an event-based simulation kernel, has gained wide acceptance, especially as a tool for early design space exploration. It is now supported by a variety of EDA tools and IP vendors and has rapidly gained acceptance as a standard modeling platform (approved as IEEE standard 1666 in December 2005) that enables the development and exchange of very fast system-level models and for the development of system-level software. It is not uncommon to reach performance simulation speeds up to 500 KHz with these tools [Rissa et al. 2005]; however, such tools are aimed at performance modeling of systems and provide no constructs for expressing component power consumption or methodologies on how to address power modeling.

System-level power modeling has been explored relatively recently in literature. Initial attempts at system power modeling were aimed at developing simple power models of various components so that insights into overall system behavior could be obtained. Examples of this include Simunic et al.'s work [1999] with simple analytical models of embedded system components and Benini et al.'s [1998] study of simple statemachine-based power models for certain components of embedded systems. Further work in the field has focused both on using detailed power models and on overall methodologies.

Bergamaschi et al.'s [2003] *SEAS* attempts early block diagram-level analysis of systems in order to allow key design decisions to be made earlier. Core performance is estimated using latency-based timing approximations, and actual functionality is abstracted away in order to achieve higher throughput. Power is modeled using state machine/spread sheet-based analysis. Designers can also use SEAS to aid floorplanning. We differ from this approach, since we model

power, performance, and *functionality*, yielding a simulation tool that enables designers to easily obtain power information for real applications. However, designers must rely on other tools for floorplanning decisions.

Power models parameters are typically obtained from characterizations of low-level (such as HDL) or hardware implementations of the component being studied. Givargis et al. [2000a; 2000b] present characterization and modeling techniques for HDL system components. Lajolo et al. [2002; 2000], on the other hand, suggest runtime calls to low-level simulations as an alternative to precharacterization. Component characterization is sometimes assumed to be an already performed step and is often omitted from work on power-modeling methodologies. Since validation against real hardware is a key contribution of the work we present, we address stimulus-based characterization of physical hardware and provide a details of the characterization methodology used.

Key research on system-level power modeling methodology includes *Platune* [Givargis and Vahid 2002], which is targeted at tuning SoC design parameters by running small synthetic kernels on a number of different system configurations. This allows rapid exploration of large design spaces, albeit using small programs and parameterized abstractions of the components used. This is a useful and complementary approach to ours, since we describe a tool that can run large, extremely realistic workloads on a detailed system model to provide accurate analysis of both performance and power. The appropriate tool for a given task will depend on the usage scenario.

A fundamental problem in any power-performance cosimulation is the manner in which information about component behavior and state is conveyed from the performance models to the power models. One approach is trace analysis, in which a performance-only simulation writes out information to a file, which is then fed to a power analysis program. Talarico et al. [2005] suggest such an approach, in which traces from SystemC performance simulation runs are postprocessed to estimate component activity for modeling. They demonstrate its application to a simple baud-rate generator, run simulations over a few thousand clock cycles, and verify against RTL-based estimates. However, such trace-based approaches have limitations in terms of performance, since system calls and disk I/O are expensive operations.

An alternative to trace analysis is execution-driven power modeling, in which power and performance modeling are tightly integrated, and power analysis can take place efficiently at runtime. Bona et al. [2004] and Caldari et al. [2003] focus on execution-driven power modeling of certain on-chip buses for use with SystemC. They run performance simulation for an entire system to accurately drive a bus power model. Power consumption for other components is not modeled. Bona et al. [2004] run small benchmarks to validate the bus power model against HDL, while Caldari et al. [2003] do not present any validation data. Beltrame et al. [2004] describe a plug-in to the StepNP simulation platform for enabling power estimation for multi-processor systems-on-a-chip. However, they do not quantify modeling accuracy and speed, or perform validation. In addition to power-modeling large workloads and validating against physical hardware, we also discuss a more scalable software architecture for

power-performance cosimulation. Compared to previously proposed architectures, we allow a higher degree of scalability by providing support for power modeling in hierarchical systems: where subcomponents and subsubcomponents may have their own power models that feed information to the power model for a high-level component.

Bansal et al. [2005] use an execution-driven power-modeling framework and compare the computational effort of simulating each component with its share of the power consumption. They suggest that the computational effort involved in power simulation can be reduced by using different power models for each component at different points of time. They report simulation speeds 10x faster than an RTL simulation (which is presumably in the 1–10 KHz range). We differ from their approach in that we use computationally lightweight power models at high levels of abstraction throughout, thus achieving very high simulation speeds (over 1 MIPS), while maintaining a high degree of accuracy.

These studies represent varying approaches to the problem and provide insights into the many issues involved. However, most of studies reported in literature do not validate their models against hardware — of the studies listed above, only Benini et al. [1998] and Simunic et al. [1999] have studied real hardware; the remainder either do not discuss validation at all or compare their results to HDL. While HDL-based methodologies provide useful insights, the low speed of HDL simulation has proved to be a significant obstacle to realistic validation with large workloads. Successful application to a real-world system is crucial both as proof-of-concept and for validation of the underlying models and methodology. In addition, *none* of the approaches listed above report runtimes corresponding to full-system simulation speeds over 100 KHz and most do not quantify execution speed.

In this paper, we describe a SystemC-based methodology for fast and accurate power-performance cosimulation and integrate it with detailed power models of the XScale microprocessor, caches, busses, and DRAM. Based on this, we present a case study in which we compare physical measurements from a hardware platform running realistic applications on top of a Windows CE operating system as a real-world validation of the techniques applied.

3. SYSTEMC AND TRANSACTION-LEVEL MODELING

With increasing system complexity, a variety of tools and languages have evolved to specifically address high-level system performance modeling. These include SpecC [Fujita and Ra 2001] and SystemC [Grotker et al. 2002], among others [Habibi and Tahar 2003; Loo et al. 2002; Rich 2004]. In this section, we provide a brief overview of SystemC as a system-level performance-modeling tool.

SystemC has been emerging as a standard open-source system modeling language and methodology and was recently approved as an IEEE standard. SystemC implementations provide a C++ based class library for description of hardware modules and communication channels, and a fast event-based (rather than clock-driven) simulation kernel. The ability to compactly describe a system makes it a popular tool for exploration in early design stages, as a simulator

for software development and performance analysis, and as a behavioral verification tool.

SystemC enables several levels of abstraction, including:

— *Functional Specification*, which has no timing information.
— *Register Transfer Level implementation*, which is both bit and cycle accurate, and
— *Transaction-Level Modeling* (TLM) [Cai and Gajski 2003; Grotker et al. 2002] represents a range of abstraction levels that have both functional and timing information at a high level of abstraction.

For this paper, we define *transaction-level models* as those which interact through well-defined high-level inter module interactions (such as a memory bus read request) that may span multiple clock cycles. Performance modeling is based on accurate clock cycle counts at the initiation and completion of each transaction. Events occurring during a transaction need to be modeled only to the extent that they affect correctness at this level of granularity. This is often also referred to as *cycle-count accurate modeling*. This level of abstraction is leveraged in the described work to achieve compact system descriptions, high simulation speed, and accurate performance modeling.

4. METHODOLOGY

We divide the methodology into three sections: *parameter extraction*, in which the components are characterized, *performance modeling*, in which a SystemC-based performance simulation infrastructure is set up, and *power modeling*, where the performance modeling framework is augmented with power modeling capabilities.

4.1 Stimulus-Based Parameter Extraction

In this section, we describe how system components can be characterized so that the high-level power models reflect accurate information. We use short assembly programs (*stimuli*) to characterize various components. A stimulus sets up the system into a predefined state and runs a large number of instances of a short instruction sequence in a loop. For example, the energy cost of a microprocessor instruction can be calculated by running a number of instances of the instruction in a loop while measuring average power. To study more complex effects, a sequence of several instruc- tions can be replicated several times and looped. The loop should be short enough to fit in the instruction cache (unless out-of-cache effects are being studied) and long enough for the error because of the branch at the end of the loop to have negligible impact on measurements [Tiwari et al. 1996]. Similarly, stimuli running repeated cache misses or memory accesses can be used to easily measure the energy cost of each bus transaction type. Stimuli for each component are based on its ISA or external interface, not on its detailed microarchitecture, and so are fairly straightforward to create.

Using the method described, we ran various stimuli on hardware to obtain the parameters for the power models. However, it must be stressed that this

approach is not limited to postsilicon characterization, but can be used with any lower-level tool (including RTL and microarchitectural descriptions) that can map an energy value to each stimulus. A wide variety of RTL and microarchitectural power modeling tools exist and stimuli can be run on these instead of hardware to extract power model parameters (this approach is taken, for example, by Givargis et al. [2000a; 2001] for peripheral cores). It must be noted that such tools are not completely accurate and their inaccuracies will be reflected in the final power estimates when they are used for characterization instead of hardware. We characterize directly with hardware to quantify how much *additional* inaccuracy is introduced by our methodology and we find this to be well within 5%.

4.2 Performance Modeling

The platform we model is based on the *XScale* [Intel 2004], a family of Intel microprocessors that implement the ARM ISA, use deep pipelines and microarchitectural optimizations for high performance, and feature a WMMX (Wireless MMX) SIMD coprocessor. We use Intel's *Xsim*, a C-based cycle-count accurate performance simulator for the XScale family. It models all XScale instructions, the L1 caches, and the WMMX coprocessor. The fetch and retire times of each instruction are computed by tracking dependencies and resource constraints instead of detailed pipeline modeling. Xsim has been validated to be cycle-accurate at instruction execution and accurate within 2% of hardware on memory accesses. We modified Xsim to enable its reuse as a modular SystemC component.

We use transaction-level SystemC models of SDRAM, SRAM and other system modules. We create a transaction-level bus model to reflect the off-chip system bus. The various memories (SDRAM, SRAM, and Flash) are bound to identical address ranges on the simulated platform and on actual hardware.

A complete SystemC-based platform simulation consistently reached execution speeds between 1 and 1.2 MIPS, allowing us to complete a Windows CE boot and application run in under 20 min. No appreciable slowdown was observed (at a measurement accuracy of 2%) when power modeling capabilities were added.[3] This is to be expected, since the computational overhead of the kind of high-level power modeling performed in this case is typically very small (a member function invocation and a counter update each time a component power model is invoked) compared to the computation involved in decoding and executing a single processor instruction (which involves multiple nested switch statements, bit manipulation, a variety of function calls, checks for special conditions, register file and cache updates, checks for stalls and hazards, updating pipeline state, managing timing information, and possibly TLB and branch predictor lookups). The overall execution speed is thus determined by

[3]Measured on a 2.8-GHz Pentium 4 HT with 1 GB of 400-MHz DDR RAM. Disabling power modeling at compile time changed the average exe-cution time over 10 runs of a 1.25 billion instruction run from 1068 to 1059 s (0.8%). The variation of individual runs from the mean (because of random latencies, background processes, etc.) was 21 s, in each case (1.87%).

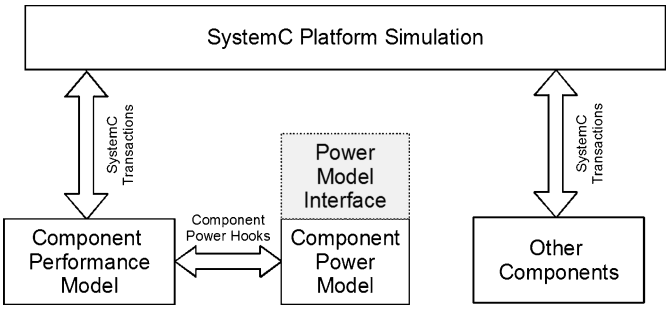


Fig. 1. Proposed software structures for systemC power modeling. The hooks for communication between performance and power models are component-specific, while the power model interface is standardized.

the performance modeling and, power modeling, if done at a sufficiently high level of abstraction, is not a bottleneck.

It must be noted that the high *accuracy* in terms of power consumption is because of the detailed nature of the power models used, including the most detailed instruction-level XScale/ARM power model to date. However, the gains in *speed* result from the high transaction-level abstraction at which both power and performance were modeled.

4.3 Software Architecture for Power Models

At the most fundamental level, the purpose of a component's power model is to monitor component activity in order to track its energy consumption. We separate out activity monitoring, which is highly component-specific, from energy accounting and reporting, which can be standardized across all power models.

While the implementation of this is not complex, we found that the robustness and reuse achieved through this approach considerably simplified both the creation of power models and the top-down data gathering required for power analysis. No changes in the SystemC kernel were required and the power-enabled components fit directly into the existing framework. In addition, the fact that the power model of each component exposes a standard interface to the rest of the system simplifies component-independent power analysis operations (such as sorting all components by average power consumption, finding components with the maximum variation in power, etc.). We outline some of salient details, which will show its general applicability.

4.3.1 *Interfaces.* Each performance model uses a component-specific interface (the component power hooks) to transfer power-specific information to the corresponding power model, which computes the energy involved based on this information. However, rather than having a separate energy accounting and reporting scheme for each component, a generic power model interface provides a standard definition and implementation of these. This is seen in Figure 1.

4.3.1.1 *Component power hooks.* These are a small set of functions exposed by the power model and called by the component (the performance model) when

Table I. Using the Power Model Interface[a]

Get power model of a component	`component.getPM()`
Get total energy consumed by a component	`component.getPM().getEnergy()`
Get read energy consumed by SDRAM	`sdram.getPM().getEnergy(``read")`
Add 32 nJ to SDRAM Read energy	`sdram.getPM()`
(can only be called by a power model)	`.incrementEnergy(``read", 32E-9)`
Find out if the given contributor is a subcomponent	`mem.getPM().isComponent(``sdram1")`
Get the power model of a subcomponent	`mem.getPM().getPM(``sdram1")`

[a]This table illustrates common operations, such as obtaining a reference to a component's power model, obtaining the total energy consumed, getting the energy consumed for particular operations, updating the energy consumed, and manipulating the power models of subcomponents, if any.

it has power-relevant information (such as information about the current transaction). The XScale power model exposes functions, such as *gotInstruction*() and *gotCacheAccess()*, etc. The information needed by the power model is passed as parameters to these functions. For example, the performance model passes the cache access type and number of bytes accessed as parameters to *gotCacheAccess*(), which causes the power model to calculate the incremental energy consumed and update its state accordingly. The component power hooks are tightly coupled to functionality and each component can have a different set of these.

4.3.1.2 *Power model interface.* This is a common interface implemented by all power models. We implement this as a generic power model class, which defines default implementations of all functions and data structures required. It provides a set of public functions to allow system-wide power data gathering and analysis (Table 1). In addition, it also implements a set of protected functions and fields that maintain internal data structures and energy accounting information. Power models extend this class and do not have to duplicate common functionality, thus creating a unified energy accounting structure and freeing power models from having to implement individual energy accounting schemes.

4.3.2 *Internal Data Structure.* The total energy consumed by a component can often be further broken down into various categories. SDRAM energy, for example, comprises read, write, activate, and power-down energy. A single lumped "energy" counter would discard the fine-grained information that a power model can provide.

To address this, we break down each component's energy consumption into various *contributors*. Each contributor in a component is identified by a name and has its own energy counter. The data for each component's contributors is kept in an internal hash table for fast lookup by name. The class that implements the generic power model interface performs all housekeeping and energy-accounting tasks.

In hierarchical systems, subcomponents can be mapped as contributors and their power models are queried for energy values when needed. This hierarchical structuring enables system scalability, since it allows a top-level system

power analysis or trace generation scheme to study the system at various levels of granularity without having to be aware of the details of each modeled component. Thus, low-level modules can be added, subtracted, or substituted without having to rewrite the top-level power data-gathering procedures (which only need to know about top-level modules). This is contrast to schemes where each power model for each component in the system under study must be considered separately, since there is no hierarchical structure associated with the power models [Bona et al. 2004].

All of these are implemented in the generic power model class, which manages these data structures and exposes only simple function calls to the outside world (Table I). Note that in a hierarchical system (such as a memory subsystem), a contributor may itself be a component and have its own power model.

5. POWER MODELS

We now describe the power models used for the XScale microprocessor, its WMMX SIMD coprocessor, off-chip address and data buses, caches, SRAM, and SDRAM.

5.1 The XScale Microprocessor

To model the Xscale processor, we use an instruction/event-based processor power model, partly drawn from earlier studies of microprocessor power consumption [Russell and Jacome 1998; Sinha and Chandrakasan 2001; Tiwari et al. 1996; Varma et al. 2005]. Our stimulus programs characterize the following energy effects:

— Leakage Power and Voltage-Frequency Scaling: The XScale processor provides a large number of voltage/frequency settings. We run a given stimulus at a fixed voltage and vary the frequency, obtaining a linear plot. Static power dissipation, which is largely leakage power, is estimated by extending this curve to obtain power consumption at zero frequency [Sinha and Chandrakasan 2001]. Power is then given by:

$$P = P_{static} + P_{dynamic} = \mathrm{VI}_{static} + \frac{1}{2} C_L f V_{dd}^2 \tag{1}$$

where P_{static} and $P_{dynamic}$ are the static and dynamic power consumption, respectively, I_{static} is the static current consumed, C_L is the load capacitance of the system, f is the switching frequency, and V_{dd} is power supply voltage.

— Low-Power States: We also characterize power consumption of the processor in various low-power modes such as *idle, deep idle, standby, sleep,* and *deep sleep* [Intel 2004].

— Instruction Opcode: Based on functionality, we divided the instructions divided into 11 different types (*add, load*, etc.), similar to Sinha and Chandrakasan [2001]. Each energy cost was measured using straightforward stimuli running the same instruction repeatedly with zero operands and built into the power model.

— Operand Value: The value of the operands affects the energy consumed to execute an instruction. Energy tends to increase roughly linearly with the operand value and the number of "1"s in the operand [Contreras et al. 2004; Sinevriotis et al. 2000].
— Bypass Paths: A rather interesting pattern of bypass path behavior was observed, with three different cases:

1. The base case is when there are no interinstruction dependencies and all source operands are obtained through register file reads.
2. When all source registers for an instruction are the destination registers for a previous instruction, the source operands are obtained from bypass paths, and 4% less energy than the base case is used.
3. When both the bypass paths and the register file are used to get source operands, 5% more energy than the base case is used.

To the best of our knowledge, we are the first to characterize this effect.

— Sign Update and Conditional Flags: Instructions, which updated or used the conditional flags, consumed more energy than instructions that did not. This increase was under 0.5% and so it has not been made part of the power model.
— Register Switching: When two consecutive instructions use different source or destination registers, an energy overhead is incurred, depending upon the number of registers switched. This can exceed 10% of the total instruction energy and can be expected to be incurred often. To the best of our knowledge, we are the first to characterize this effect.
— Cache Accesses: Caches are modeled as on-chip SRAM. From the instruction-set point of view, the energy cost of a load or store depends on the number of bytes accessed. We characterize and model this.
— Shifts: The ARM instruction set allows the last operand of an instruction to be bit-shifted by an immediate or a register value. This shift causes an additional result latency of one cycle. Incremental energy costs for all shift types are modeled.
— Stalls: Stalls can be divided into instruction stalls, which result from interinstruction data dependencies, event stalls, such as stalls on a double-word load, branch stalls, or the pipeline flush penalty, and memory stalls on accesses to external memory. Energy costs of all stall types were characterized and modeled.

5.2 The WMMX Coprocessor

The XScale processor family has an on-die ISA-mapped Wireless MMX coprocessor [Paver et al. 2004] for fast SIMD processing. We divided the WMMX instructions into 17 types based on functionality, in a manner similar to that for the main processor. Base costs for WMMX instructions were characterized separately and built into the power model.

5.3 Address and Data Buses

An off-chip system bus connects the processor to Flash ROM, SDRAM, and various peripherals. We characterize the power requirements of both the address and data bus by using stimuli to drive specific sequences of values onto them. Bus energy consumption in the nth bus clock cycle can be expressed as:

$$E_n = C_1 \times H(D_n, D_{n-1}) + C_0 \tag{2}$$

where C1 is a constant depending on bus capacitance, C_0 is the energy cost of turning on the bus driver I/O circuits, D is the data on the bus (including control bits), and H represents the Hamming distance between two binary numbers.

For each type of memory transaction (write, burst write, read line, etc.), the exact sequence of low-level operations involved is defined by the memory protocol and the memory controller configuration. For example, for an 8-word read from a particular memory address (a typical cache miss), the exact timings of row and column address strobe assertions as well as the row address, column address, activation time, etc., are known. The SystemC bus power model simply calculates these and assigns an energy consumption to each incoming transaction rather than running a cycle-by-cycle simulation, which would drastically affect simulation speed.

Note that the bus is driven by multiple power sources: the processor drives both address and data buses, while the SDRAM consumes I/O power when it drives data onto the data bus. We account for these appropriately.

5.4 Caches and SRAM

We use an SRAM power model similar to Beltrame et al. [2004], Coumeri and Thomas [1998], and Itoh et al. [1995] to model caches and on-chip memory-mapped SRAM. Energy consumption is modeled as:

$$E = N_{read}E_{read} + N_{write}E_{write} + N_{idle}E_{idle} \tag{3}$$

where N is the number of times each operation is performed and E is the energy cost of that operation. The cache energy consumption is modeled in the XScale instruction-level power model, with each kind of cache access (load or store, byte, half-word, word or double-word) characterized and modeled separately.

5.5 SDRAM

SDRAM power consumption can be divided into core power consumption (for the memory elements) [Micron 2003], and I/O power consumption (for driving data onto the data bus). We characterize these using stimuli. We use the data bus power model to calculate SDRAM I/O power consumption. The main elements of SDRAM core power are:

— Base Power Consumption (P_b): The average power consumed by SDRAM, when not accessed, is the sum of the standby and average refresh power.
— Activation Power (P_{act}): The average power consumed when an SDRAM page is active.

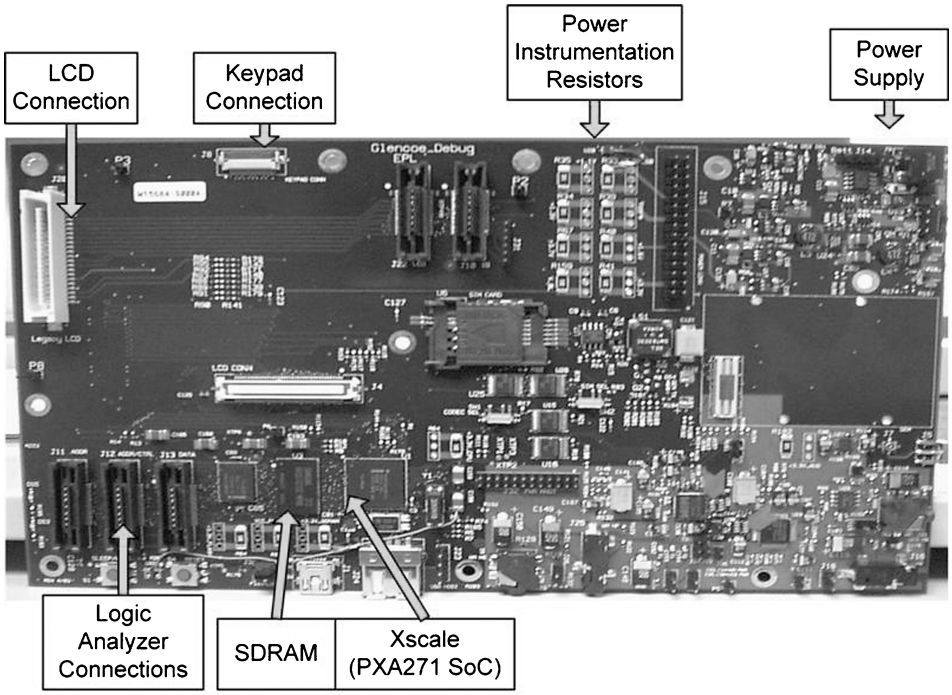


Fig. 2. The reference platform used for physical experiments. The XScale processor, the WMMX unit, and the L1 caches are on the PXA271 SoC. The logic analyzer connections allow bus signals and timing to be observed, while an array of power instrumentation resistors allows the power supply of each component to be separately studied.

— Read or Write Power (P_{rw}): The power consumed during each SDRAM read/write operation. The values of read and write current for SDRAM are equal [Micron 2003].

SDRAM power is modeled in a manner similar to the bus transactions. The low-level operations in each transaction are defined by the bus protocol and memory controller. The SDRAM power simply uses these to calculate the energy cost of each incoming transaction without having to run a cycle-by-cycle simulation. For a given transaction, energy consumption is given by:

$$E_{transaction} = P_b T_b + P_{act} T_{act} + P_{rw} T_{rw} \tag{4}$$

Representing the power model at the transaction level, rather than at the cycle level, lowers the computational overhead of power modeling, and contributes to simulation speedup.

6. EXPERIMENTAL SETUP

For validation, we use a reference platform (Figure 2) featuring an XScale-based PXA271 SoC, which contains an XScale processor, its WMMX coprocessor, L1 instruction and data caches (32 KB each), and other system components [Intel

2004]. The platform also has 64-MB on-board SDRAM, 32-MB synchronous Flash, and a variety of peripherals. The main board is instrumented with 100-mΩ resistors in series with the power supply on each module, which enable power measurements of individual components at a granularity similar to that at which power is modeled.

We simultaneously measure the power consumption over multiple channels using an NI-DaQ data acquisition card, sampling at up to 20 KHz with a postcalibration accuracy of ± 5 μV. The voltage drop across each resistor is of the order of millivolts to tens of millivolts. The instrumentation resistors used are 1% accurate. Postprocessing of acquired data is done using LabView virtual instruments.

The major contributors to power consumption are the XScale-based PXA271 SoC, the SDRAM memory, and the off-chip buses. The three power domains we measure for validation are:

— Core Power: The 1.3-V main power supply to the XScale- based PXA271 SoC. In our configuration, it powers the XScale microprocessor core, L1 caches, WMMX unit, clocks, and on-chip interconnect.
— I/O Power: The 3.3-V supply to the PXA271. It powers the on-chip memory controller and I/O pads, including all off-chip buses. It also provides standby power for on-chip components.
— SDRAM Power: The 3.3-V power supply common to the SDRAM (both SDRAM core and I/O pads).

We compare the predicted and measured power for each domain separately. Processor frequency is varied, while the memory controller runs the off-chip bus at 91 MHz.

7. RESULTS

For validation, we measure average power over long benchmark runs and compare it with the power estimates obtained from simulation. We use Windows CE as the operating system, and run identical benchmarks on the hardware and the simulator. The simulator runs a complete OS boot routine followed by the application. Each benchmark is run in a loop, with average power measured physically on hardware over a period of 1 s and compared with the estimate obtained from the simulator.

To validate our results, we use the following benchmarks:

— Autocorrelation and Huffman decoding benchmarks from the EEMBC benchmark suite.
— The motion estimation kernel from an h.264 video encoder.
— A video filter (vidsp) from an h.264 video decoder. We evaluate three versions of this filter: plain C, hand- opti- mized assembly, and hand-optimized assembly with additional WMMX optimizations.
— FFT (for 10,000 samples), JPEG forward DCT (JFDCT) and matrix multiply (MatMul) benchmarks from SNU-RT benchmark suite from Seoul National University.

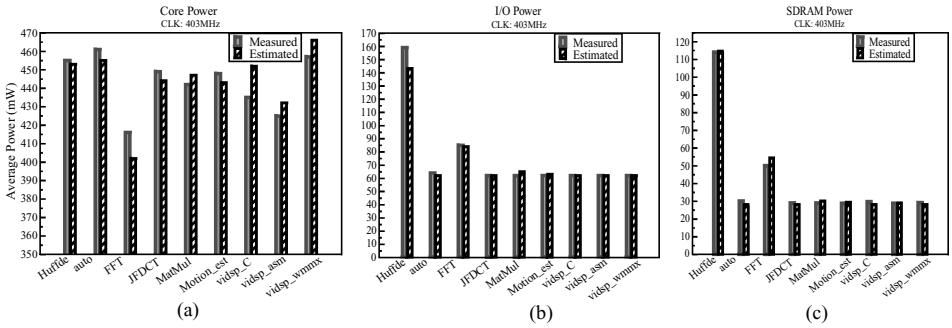


Fig. 3. Power consumed by various power domains at 403 MHz. Note that Huffman decoding and FFT generate the highest bus activity, thus consuming the most SDRAM and I/O power.

Figure 3a shows microprocessor core power consumption. We see excellent agreement between the measured and estimated power, with a worst-case error of 3.9% (for vidsp C). As in earlier studies [Russell and Jacome 1998; Sinha and Chandrakasan 2001], we observe a low variation in processor power at a constant speed.

The power consumed by the I/O power supply is illustrated in Figure 3b. The base power consumption when there is no I/O activity is 62 mW. Activity, such as bus transactions, consume additional power. Large benchmarks with frequent memory accesses, such as Huffman decoding or FFT, stress the memory hierarchy, leading to increased bus power consumption. Of the other benchmarks, only MatMul is cache-bound. However, the large (32 KB) cache size ensures that benchmarks with sufficient cache locality display very sporadic bus traffic, hence, consuming little bus power. For example, the motion estimation benchmark uses an 800-KB data set. However, it performs significant computation on each block before fetching the next one, thus having low average bus power dissipation. Figure 3c shows the power consumed by the on-board SDRAM. The patterns observed are similar to those observed for XScale I/O, since the bulk of bus transactions map to an SDRAM access. The SDRAM standby power is 28 mW, which corresponds closely to the sum of power-down active standby power and average self-refresh power calculated from the component datasheet (31 mW).

It is interesting to note that while physical hardware measurements can only reveal the total power consumed by each component, detailed power modeling can expose a much finer degree of detail. For example, Figure 4 shows the various components of core power while running Huffman decoding and FFT at 403 MHz. Direct physical measurements cannot resolve net power into these components.

We also study power variation with core frequency. Figure 5 shows system power consumption while running Huffman decoding and FFT at various core speeds, with bus frequency kept at 91 MHz. Note that nonlinearities in I/O and SDRAM power are correctly tracked (Figure 5(a)). These nonlinearities arise because Huffman decoding generates a very large amount of memory traffic. At high core speeds, the traffic is so high that the SDRAM clock on the bus almost

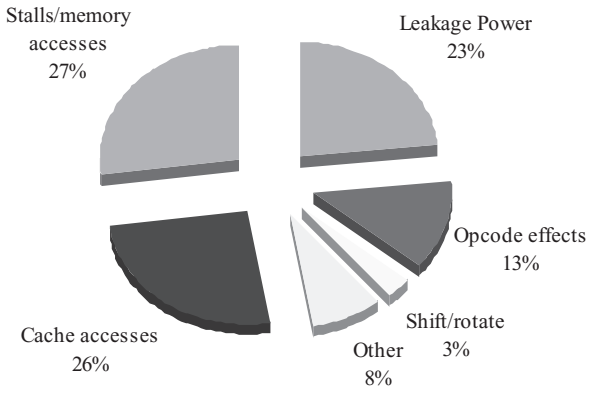
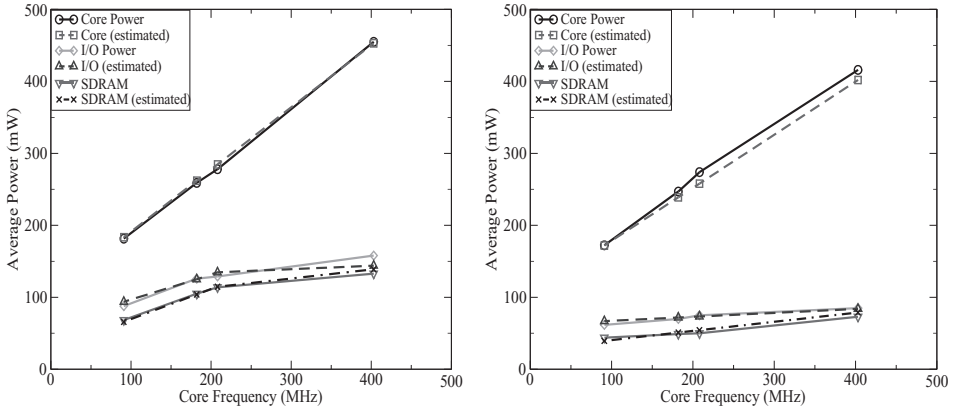


Fig. 4. Contributors to core power consumption while running Huffman decoding at 403 MHz. In contrast to the fine detail visible here, hardware measurements can only measure the total power consumption.



(a) Huffman Coding (b) FFT

Fig. 5. System power consumption at various core speeds. Bus speed is kept at 91 MHz. Note that nonlinearities in I/O and SDRAM power for Huffman decoding are correctly modeled.

always on. As core speed falls, the bus traffic falls linearly. Below a certain point, the time between transactions is sufficient for the SDRAM clock on the bus to be turned off for significant amounts of time, leading to the further lowering of power consumption at 91 MHz. FFT (Figure 5b) does not display such high memory traffic, leading to a more linear plot. For other benchmarks, the bus traffic is low and power consumption is mostly the base power consumption, which does not decrease significantly as core speed is lowered.

For all benchmarks, the power estimates obtained were in excellent agreement with physical measurements. While power consumption for each component varied by over a factor of two over this frequency range, we tracked this accurately and obtained average errors under 5% and worst-case errors under 10% for each component at all speeds.

8. CONCLUSIONS AND FUTURE WORK

Modeling at high levels of abstraction enables component reuse, top-down design, and rapid design space exploration. While languages such as SystemC provide valuable and widely accepted tools for high-speed high-level system performance modeling, there still exists no standard strategy for high-speed system power modeling. We define a simulation-based methodology for extending system performance modeling frameworks to also include power modeling. We demonstrate the use of this methodology with a case study of a real, complex embedded system, comprising the Intel XScale embedded microprocessor, its WMMX SIMD coprocessor, L1 caches, SDRAM, and the on-board address and data buses. We describe detailed power models for each of these components and validate the system power estimates against physical measurements from hardware, demonstrating that such frameworks enable designers to model both power and performance at high speeds without sacrificing accuracy.

The power-enabled system simulator described predicts power accurately across a variety of applications, with the the worst-case difference between estimated and measured power being under 10%, and average error under 5%. Since the power models are implemented at a high level of abstraction, they are extremely lightweight in terms of computation, and adding them to existing performance models appreciably affected simulation speed. The simulation proceeded at speeds in excess of 1 MIPS, enabling us to run complete applications on a real-world operating system. Our current work is focused on modeling peripheral components and extending our methodology to presilicon systems using stimulus-based power characterization of microarchitectural descriptions instead of hardware.

ACKNOWLEDGMENTS

The Xsim simulator was built by Brett Gaines and modified for SystemC by Bhaktha Keshavachar. The reference board used for physical experiments was provided by Jerzy Kolinski and Thao Xiong. Windows CE ports were provided by Sai Prasad, Peter Adamson, Thomas Cronin, and Josh Miller. The video filter for H.264 decoding was created by Bob Reese.

REFERENCES

AUSTIN, T., LARSON, E., AND ERNST, D. 2002. SimpleScalar: An infrastructure for computer system modeling. *IEEE Comput.* 35, 2 (Feb.), 59–67.

BANSAL, N., LAHIRI, K., RAGHUNATHAN, A., AND CHAKRADHAR, S. T. 2005. Power monitors: a framework for system-level power estimation using heterogeneous power models. In *18th International Conference on VLSI Design*, Kolkota, India. 579–585.

BELTRAME, G., PALERMO, G., SCIUTO, D., AND SILVANO, C. 2004. Plug-in of power models in the StepNP exploration platform: Analysis of power/ performance trade-offs. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Washington, D.C.

BENINI, L., HODGSON, R., AND SIEGEL, P. 1998. System-level power estimation and optimization. In *International Symposium on Low-Power Electronics and Design (ISLPED)*.

BERGAMASCHI, R. A., SHIN, Y., DHANWADA, N., BHATTACHARYA, S., DOUGHERTY, W. E., NAIR, I., DARRINGER, J., AND PALIWAL, S. 2003. SEAS: A system for early analysis of SoCs. In *International Symposium on System Synthesis*.

BONA, A., ZACCARIA, V., AND ZAFALON, R. 2004. System level power modeling and simulation of a high-end industrial network-on-chip. In Design *Automation and Test in Europe (DATE)*.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architecture-level power analysis and optimization. In *International Symposium on Computer Architecture (ISCA)*.

CAI, L. AND GAJSKI, D. 2003. Transaction Level Modeling: An overview. In *International Conference of Hardware-Software Codesign and System Synthesis (CODES+ISSS)*.

CALDARI, M., CONTI, M., COPPOLA, M., CRIPPA, P., ORCIONI, S., PIERALISI, L., AND TURCHETTI, C. 2003. System-level power analysis methodology applied to the AMBA AHB bus. In *Design Automation and Test in Europe (DATE)*.

CHEN, R. Y., IRWIN, M. J., AND BAJWA, R. S. 2001. Architecture-level power estimation and design experiments. *ACM Trans. Design Automation of Embedded Sys. 6*, 1 (Jan.), 50–66.

CONTRERAS, G., MARTONOSI, M., PENG, J., JU, R., AND LUEH, G.-Y. 2004. XTREM: A power simulator for the Intel XScale. In *Languages, Compilers, and Tools for Embedded Systems (LCTES)*.

COUMERI, S. L. AND THOMAS, D. E. 1998. Memory modeling for system synthesis. In *International Symposium on Low-Power Electronics and Design (ISLPED)*.

FUJITA, M. AND RA, H. N. 2001. The standard SpecC language. In *International Symposium on System Synthesis*.

GIVARGIS, T. AND VAHID, F. 2002. Platune: A tuning framework for system-on-a-chip platforms. *IEEE Trans. Comput.-Aided Design Integrated Circuits and Syst. 21, 11 (Nov.)*, 1317–1327.

GIVARGIS, T., VAHID, F., AND HENKEL, J. 2000a. Instruction-based system-level power evaluation of SoC peripheral cores. In *International Symposium on System Synthesis*.

GIVARGIS, T. D., VAHID, F., AND HENKEL, J. 2000b. A hybrid approach for core-based system-level power modeling. In *Asia South Pacific Design Automation Conference (ASP-DAC)*.

GIVARGIS, T. D., VAHID, F., AND HENKEL, J. 2001. Trace-driven system-level power evaluation of system-on-a-chip peripheral cores. In *Asia South Pacific Design Automation Conference (ASP-DAC)*.

GROTKER, T., LIAO, S., MARTIN, G., AND SWAN, S. 2002. *System Design With SystemC*. Kluwer Academic Publishers, Boston, MA.

HABIBI, A. AND TAHAR, S. 2003. A survey on system-on-a-chip design languages. In *Proceedings of the 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications*.

Intel. 2004. *Intel XScale Microarchitecture for the PXA255 Processor: User's Manual. Intel.*

ITOH, K., SASAKI, K., AND NAKAGOME, Y. 1995. Trends in low-power RAM circuit technologies. In *Proceedings of the IEEE 83*, 4, 524–543.

JAYADEVAPPA, S., SHANKAR, R., AND MAHGOUB, I. 2004. A comparative study of modeling at different levels of abstraction in system on chip designs: A case study. In *IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI)*.

LAJOLO, M., RAGHUNATHAN, A., DEY, S., AND LAVAGNO, L. 2000. Efficient power estimation techniques for system-on-chip design. See citeNLajolo2002, 253–266.

LAJOLO, M., RAGHUNANDAN, A., DEY, S., AND LAVAGNO, L. 2002. Cosimulation-based power estimation for system-on-chip design. *IEEE Trans. VLSI Syst. 10*, 3 (June), 253–266.

LOO, S. M., WELLS, B. E., FREIJE, N., AND KULICK, J. 2002. Handel-C for rapid prototyping of VLSI coprocessors for real time systems. In *The 34th Southeastern Symposium on System Theory*.

Micron. 2003. TN-46-03 Calculating DDR Memory System Power. Micron.

PAVER, N., ALDRICH, B., AND KHAN, M. 2004. *Programming with Intel Wireless MMX Technology: A Developer's Guide to Mobile Multimedia Applications*. Intel Press, Santa Clara, CA.

RICH, D. I. 2004. The evolution of SystemVerilog. In *Electronics Systems and Software*.

RISSA, T., DONLIN, A., AND LUK, W. 2005. Evaluation of SystemC modelling of reconfigurable embedded systems. In *Design Automation and Test in Europe (DATE)*.

RUSSELL, J. T. AND JACOME, M. F. 1998. Software power estimation and optimization for high-performance 32-bit embedded processors. In *International Conference on Computer design (ICCD)*.

SIMUNIC, T., BENINI, L., AND MICHELI, G. D. 1999. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference (DAC)*.

SINEVRIOTIS, G., LEVENTIS, A., ANASTASIADOU, D., STAVROULOPOULOS, C., PAPADOPOULOS, T., ANTONAKOPOULOS, T., AND STOURAITIS, T. 2000. SOFLOPO: Towards systematic software exploitation for low-power designs. In *International Symposium on Low-Power Electronics and Design (ISLPED)*.

SINHA, A. AND CHANDRAKASAN, A. P. 2001. JouleTrack–a web based tool for software energy profiling. In *Design Automation Conference (DAC)*.

TALARICO, C., ROZENBLIT, J. W., MALHOTRA, V., AND STRITTER, A. 2005. A new framework for power estimation of embedded systems. *IEEE Comput. 38*, 2 (Feb.), 71–78.

Tiwari, V., Malik, S., and Wolfe, A. 1994. Power analysis of embedded software: A first step towards software power minimization. *IEEE Trans. VLSI Syst. 2*, 4, 437–445.

TIWARI, V., MALIK, S., WOLFE, A., AND LEE, M. T.-C. 1996. Instruction-level power analysis and optimization of software. In *IEEE International Conference on VLSI Design*.

VARMA, A., DEBES, E., KOZINTSEV, I., AND JACOB, B. 2005. Instruction-level power dissipation in the Intel XScale embedded microprocessor. In *SPIE's 17th Annual Symposium on Electronic Imaging Science and Technology*.

YE, W., VIJAYKRISHNAN, N., KANDEMIR, M., AND IRWIN, M. J. 2000. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Design Automation Conference (DAC)*.

Received November 2005; revised February 2006; accepted April 2006