A New Framework for Power Estimation of Embedded Systems

A proposed modular framework for assessing power consumption of embedded systems early in the design cycle can be extended to any performance metric and uses a high level of abstraction, leading to a faster execution time. Experimental results indicate that the approach is within 20 percent of gate-level estimation and executes three orders of magnitude faster.

Claudio Talarico
Jerzy W. Rozenblit
University of Arizona

Vinod Malhotra
University of Hawaii at Manoa

Albert Stritter
Infineon Technologies AG

The overall goal of system design is to minimize development time and costs, subject to various performance and functionality constraints. To cope with the rapidly growing complexity of embedded systems, designers must work at higher levels of abstraction.[1]

Depending on the abstraction layer—the level of detail used to describe the system—designers can address different concerns. The key is to model the system at each abstraction layer with as little detail as possible and then collect performance metrics that help the development team make sound engineering decisions.

Among the many metrics used to characterize the quality of an embedded system-on-chip (SoC) design, power consumption has emerged as one of the most important. This is largely due to the proliferation of mobile battery-powered computing devices, the increasing speed and density of CMOS (complementary metal-oxide semiconductor) VLSI (very large-scale integration) circuits, and continuous shrinking of the transistor feature size of deep-submicron technologies.[2]

Designers can estimate power consumption at four different abstraction levels:

- *Circuit-level* approaches simulate the circuit at the transistor or switch level and monitor the supply current.[3]
- *Logic-level* techniques simulate a design at the logic-gate level and calculate power by considering the switching activity and node capacitance. Logic-level approaches execute orders of magnitude faster than circuit-level approaches but at the expense of accuracy.[4]
- *Register-transfer-level* approaches[5] model the power consumption of more abstract components such as muxes, adders, multipliers, and registers. They have satisfactory accuracy (5-10 percent of gate-level power estimates), but their computational time, while orders of magnitude smaller than with logic-level approaches, is too slow when applied to large designs.
- *System-level* approaches[6] estimate power consumption based on simple high-level descriptions of the system's behavior and its intended application, using an abstract notion of capacitance and switching.

Different estimation techniques are best suited to different parts of a design or different stages in the design flow.

We have developed a technique that derives power figures from the execution of high-level models rather than gate- or transistor-level precharacterizations. This technique makes it possible to assess embedded SoC designs much earlier in the design cycle, contributing to sounder decisions

0018-9162/05/$20.00 © 2005 IEEE

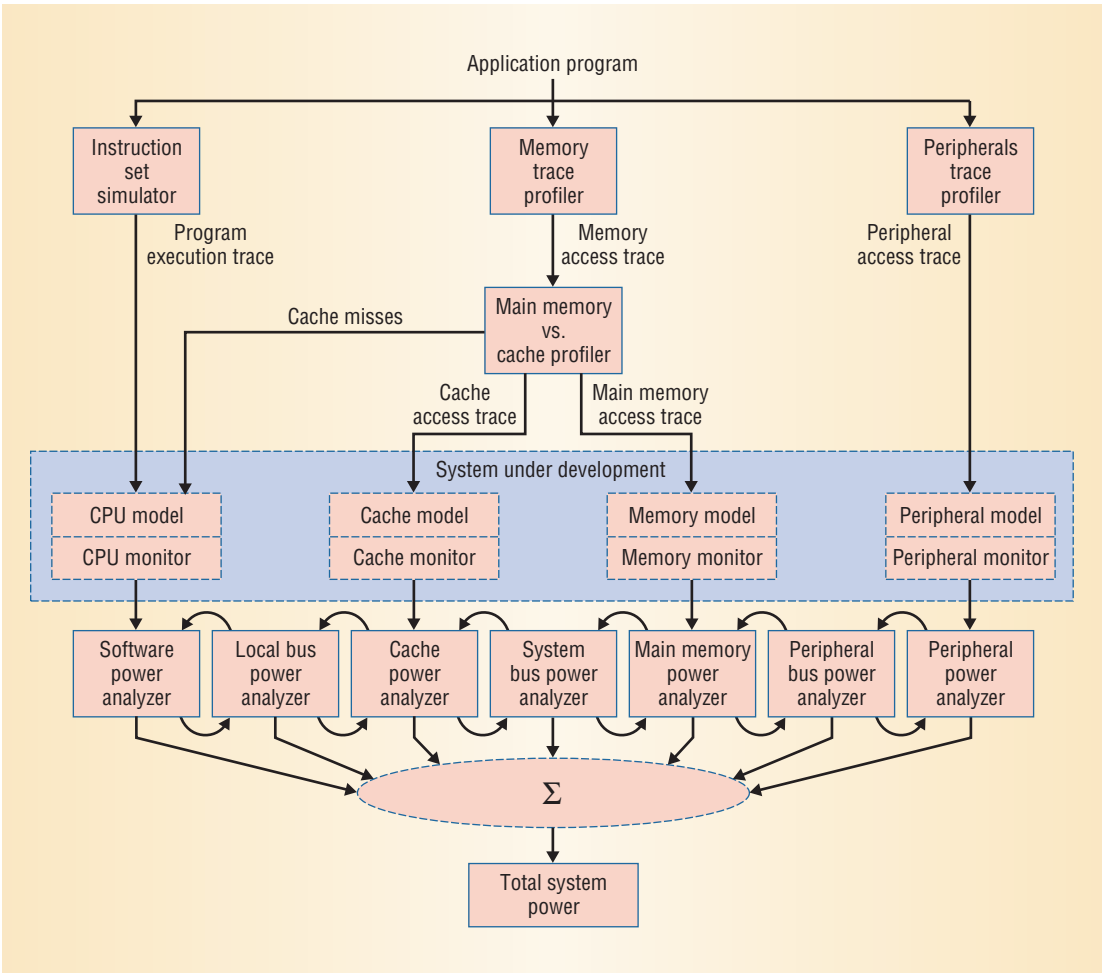

Figure 1. Power estimation framework. The framework acts as a generic wrapper around the system components, each of which has an associated simulation model and monitor. The monitor observes the model's execution and probes the data needed to characterize the component's behavior. Power analyzers then compute the performance indices of interest.

throughout the entire development process and leading to a faster execution time.

To validate our methodology, we applied it to a peripheral core—a baud rate generator—and compared the results with those obtained using a gate-level approach.

POWER CONSUMPTION MODELS

Researchers have developed several techniques for estimating software power consumption for microprocessor and digital signal processor cores, mainly at the instruction level.[2,7,8] Given a program execution trace, this approach computes the energy that each executed instruction consumes. Energy consumption depends on the specific instruction being executed as well as on previously executed instructions and on the data on which the instruction operates. This process can be accelerated by deriving a trace file of reduced size that generates equal power dissipation.[9]

Other researchers have explored software power optimization techniques.[10] In addition, a proposed mathematical model of a generic 32-bit processor, obtained through functional decomposition, classifies instructions based on the functional units exercised.[11] This model estimates the static power consumption of the single instructions executed, but it does not consider the dynamic power information associated with the actual applied input data.

Another technique estimates power consumption of peripheral cores.[12] Finally, a number of proposed system-level models for cache, memory, and bus power consumption consist mainly of closed-form equations that express power consumption as a function of usage/traffic and component parameters.[13-15]

All of these system-level techniques use gate- or transistor-level precharacterizations, which require detailed knowledge of the components' internal structure, to develop energy consumption models. However, such information may not be available early in the design process, or IP providers may not want to disclose it. In addition, a given application's power consumption provides little information about the power consumption of other applications for the same system. Consequently, characterization-based power models are highly accurate only if evaluated in the same context as that used for characterization.

POWER ESTIMATION FRAMEWORK

Figure 1 illustrates our proposed framework for estimating the power consumption of a generic embedded system. The framework functions as a

generic wrapper around the system components, each of which has an associated simulation model and monitor. The monitor observes the model's execution and probes the data needed to characterize the component's behavior. Various power analyzers then compute the performance indices of interest.

Our framework generalizes and extends the schema developed by Tony Givargis, Frank Vahid, and Jörg Henkel.[12] The key difference is that our framework does not rely on gate-level simulation to characterize each core's per-instruction power consumption. In our view, a core's behavior can be seen as the execution of a sequence of instructions, in which the term "instruction" is synonymous with "action" and is not necessarily atomic.

The framework's distinguishing feature is its modularity, which helps isolate the various system components from one another and to abstract their implementation details. This makes it possible to assess designs early in the process, when the impact of decisions is critical to avoid expensive and time-consuming iterations. The modeling concepts' generality also extends our framework beyond power consumption for use in evaluating other performance metrics.

System power consumption

Our framework consists of four steps that lead to an estimate of overall system power consumption:

- *translating* each core's functionality to a set of primitive instructions,
- *simulating* the application program,
- *mapping* the instructions requested by the application program into abstract functional units, and
- *computing* aggregate power consumption of the entire system.

The first step consists of breaking each core's functionality into a set of instructions. A component's functionality represents all possible behaviors it can assume, with behavior meaning the set of actions that the component performs during execution of an application. The goal is to devise a high-level executable model of the core that can output power consumption data during system simulation.

This first step hides the complexity of the core's internal implementation behind the simple interface offered by the instruction set. There is a tradeoff in selecting the right set of instructions: Having many fine-grained instructions can lead to greater accuracy, but it requires a longer simulation time than having fewer coarse-grained instructions.[12] The framework associates with each core's instruction set only the information needed to describe the performance metric of interest—in this case, power consumption.

The second step involves simulating the application program and extracting a trace file for the core. A trace is the sequence of instructions/data items a core executes during its simulation. The aim is to estimate the core's switching activity.

The third step consists of mapping the instructions requested by the various tasks performed by the core into abstract functional units that are used to estimate complexity—that is, gate count. Given switching activity and complexity, the framework can compute the core's power per instruction.

The fourth step involves connecting all the core models to compute the power consumption of the entire system.

The framework's modularity makes it possible to assess designs early in the process, when the impact of decisions is critical to avoid expensive and time-consuming iterations.

Power analyzer modules

Each of the power analyzer modules shown in Figure 1 embodies the analytical expressions needed to compute the power consumed by the various types of cores: processor, cache, main memory, bus, and peripherals.

The input of the power estimation flow is the application program, which feeds into the target CPU's instruction set simulator (ISS) to produce a program trace. A software power analyzer then postprocesses the program trace to estimate the power the processor consumes during software execution.

The application program also feeds into a memory trace profiler, which records all memory access traces and then calculates the number of cache demand misses for both data and instructions. The software power analyzer uses this information to account for additional power consumption due to cache-miss stalls. The main memory power analyzer and cache power analyzer also use this data to compute the power consumption of the main memory and cache accesses.

Depending on whether peripherals are accessed through memory-mapped or dedicated I/O, it is possible to extract a peripheral access trace from either the memory access traces or program traces. Any access to or from main memory, caches, and peripherals translates eventually into information traffic over the communication buses. Specific bus

The framework models the peripheral in terms of a set of instructions and a set of power modes.

power analyzers compute the power that each bus in the system consumes.

Component power consumption

Because gate-level representation of most cores may not be available early in the design process, our framework computes power dissipation analytically, combining the technology parameters obtainable from data sheets with the data gathered by executing the core's high-level model.

The framework uses ad hoc correction methods to evaluate the power consumed by nonlinear components. A typical example is the interaction between cache and processor. In this case, it is necessary to first evaluate processor power consumption by assuming the ideal case in which all instructions and data can be retrieved from the cache and then account for the energy penalty caused by the processor stalling due to read or write misses in the data cache and fetch misses in the instruction cache.

Processor. Our framework relies on an ISS to estimate the power the CPU consumes to execute the application software. The ISS maintains detailed statistics of the processor's internal activity—such as fetches, stalls, instruction execution frequency, and internal register accesses—that the software power analyzer can postprocess to compute power consumption. This technique is an extension of earlier instruction-based approaches.[2,12] The idea behind such approaches is that "by measuring the current drawn by the processor as it repeatedly executes certain instructions, it is possible to obtain most of the information that is needed to evaluate the power cost of a program for that processor."[2]

Cache. To estimate cache energy consumption, we adapted analytical models developed by Milind Kamble and Kanad Ghose.[13] Accurate estimation requires that the cache simulator maintains activity statistics for several metrics including number of hits and misses, number of tag comparisons, word-line activity, and bit-line activity.

The major components of energy consumption are in the bit lines, word lines, output lines, and input lines: $E_{cache} = E_{bitline} + E_{wordline} + E_{output} + E_{input}$. The energy dissipated in other cache components such as comparators, registers, data steering logic, control logic, and sense amplifiers is relatively small and can be neglected.

Main memory. To compute the energy that main memory consumes, our framework uses the analytical models described by Kiyoo Itoh.[14] The main sources of power dissipation are the memory cell array, row decoder, column decoder, and periphery circuits.

Bus. In deep-submicron technologies, bus power is a significant part of total power. Execution time and bus power are inversely related: A smaller bus width implies less wire capacitance and hence less power, but it requires more bus transfers and hence a longer execution time.

Every memory and peripheral access implies a data transfer over a communication bus. The total number of cache accesses N_{acc} measures traffic on the CPU-cache bus, the number of cache misses N_{miss} measures traffic on the cache-main memory bus, and the number of peripheral references N_{per} measures traffic on the peripheral bus—the bus between main memory and the peripheral devices. Given this traffic and assuming that on average at most half of the bits will toggle, our framework can then compute bus switching activity. It uses this value and bus capacitance to compute power consumption.

Peripherals. For a processor, the term "instruction" generally means an atomic action for programming the desired behavior. However, for a peripheral, an instruction is an action that, together with all other instruction set actions, describes the peripheral's functionality.[12] Our framework models the peripheral in terms of a set of instructions and a set of power modes. Power modes take into account that certain instructions can significantly change power consumption.

The framework follows a four-step procedure to obtain peripheral power consumption. First, it profiles the application program for requests to and from the peripheral. The number and frequency of peripheral accesses is a measure of its switching activity. Second, it decomposes the various types of tasks requested into instructions and maps them into abstract functional units for use in estimating complexity. Given switching activity and complexity, the framework then creates a power-per-instruction lookup table. Third, the framework executes the peripheral model to generate the corresponding trace. Finally, given the instructions trace, it uses the power-per-instruction lookup table to compute power consumption.

EXAMPLE SIMULATION

To validate our system-level approach, we used SystemC to model a baud generator unit that clocks the universal asynchronous receiver/transmitter inside the Infineon XC161CJ microcontroller. Embedded systems are inherently heterogeneous—they consist of an intricate intermix of both hard-

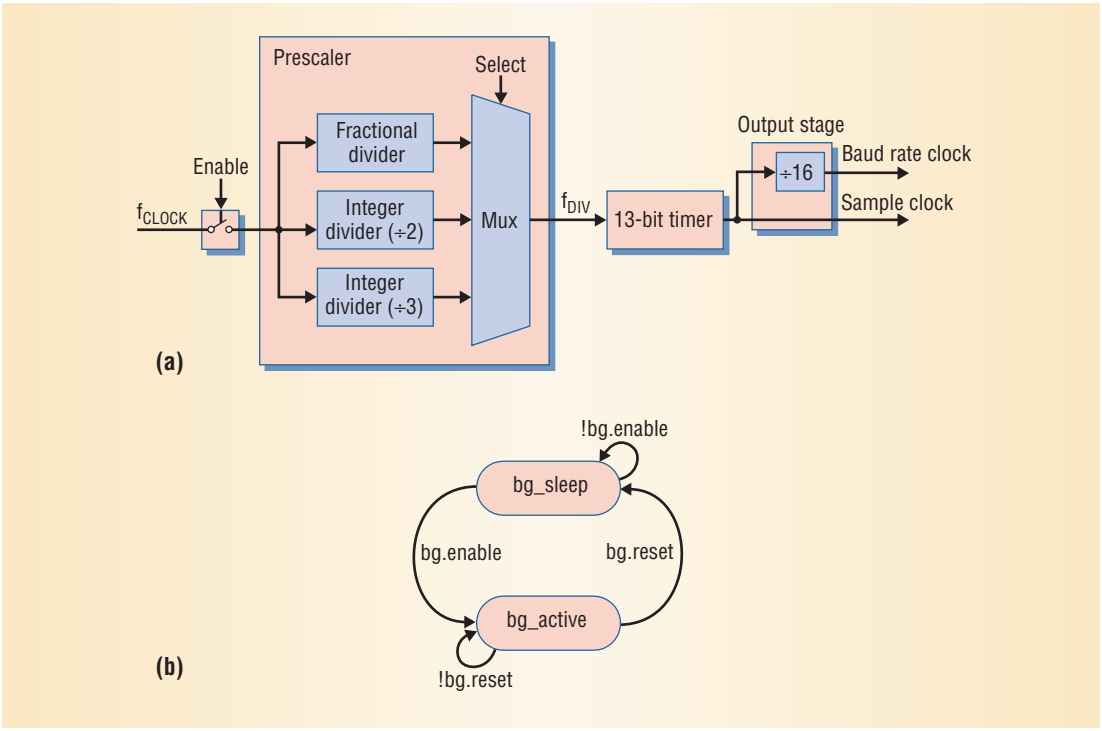

Figure 2. Baud generator architecture and power model. (a) The baud generator consists of a prescaler containing a selectable fractional divider and two fixed-integer dividers, a 13-bit timer, and an output stage providing the baud rate. (b) Each state represents a power mode, and the transition from state to state depends on the instructions from the application program.

ware and software components. Using the same high-level language to describe both hardware and software makes the modeling task easier.

As Figure 2a shows, the baud generator consists of three functional units: a prescaler containing a selectable fractional divider and two fixed-integer dividers, a 13-bit timer, and an output stage providing the baud rate.

Modeling power consumption requires only a small amount of detail. As Figure 2b illustrates, our framework uses a finite state machine to describe power behavior. Each state represents a power mode, and the transition from state to state depends on the instructions from the application program.

The total energy that the baud generator consumes during execution of the application program is given by

$$E_{bg} = T_{clock} \times \sum_{j=1}^{N} \left(n_{cyc,j} \times p_{inst,j} \right)$$

$$= \sum_{j=1}^{N} \left(T_{clock} \times n_{cyc,j} \times p_{inst,j} \right)$$

where T_{clock} is the clock-cycle period, $p_{instr,j}$ is the power dissipated during execution of instruction j, and $n_{cyc,j}$ is the number of clock cycles taken to execute instruction j. The power per instruction can be computed as

$$p_{instr,j} = \frac{1}{2} \times V_{dd}^2 \times \sum_{k=1}^{NF} C_k \times \alpha_{k,j}$$

where V_{dd} is the power supply voltage, NF is the number of functional units composing the baud generator, C_k is the total capacitance of functional unit k, and $\alpha_{k,j}$ is the switching activity occurring within the functional unit k to execute instruction j.

Experimental setup

To test our approach, we implemented a system-level model of the baud generator. The model represents the peripheral module of the power estimation framework shown in Figure 1. We used the C++ language—the availability of SystemC makes this an ideal match for a unified HW/SW framework.

The baud generator model includes only the power behavior. To achieve good accuracy, we used a state machine to express each instruction's dependency on the previous ones, with the instructions triggering transitions from one state to the other.

To compute the power per cycle of each core's instruction, we employed Infineon's 0.25-µm CMOS technology. We estimated the average capacitance of combinational cells and sequential cells separately, averaging the intrinsic capacitance of cells in the target technology, and stored the resulting values in a lookup table to facilitate access during model execution.

Figure 3a shows implementation details of our framework's peripherals module.

The *design explorer* analyzes the functional units forming the various components, estimates their complexity (total capacitance) based on the target technology, and evaluates each unit's power consumption.

The *application profiler* parses the application program and extracts the instructions that affect the peripherals and distributes them accordingly.

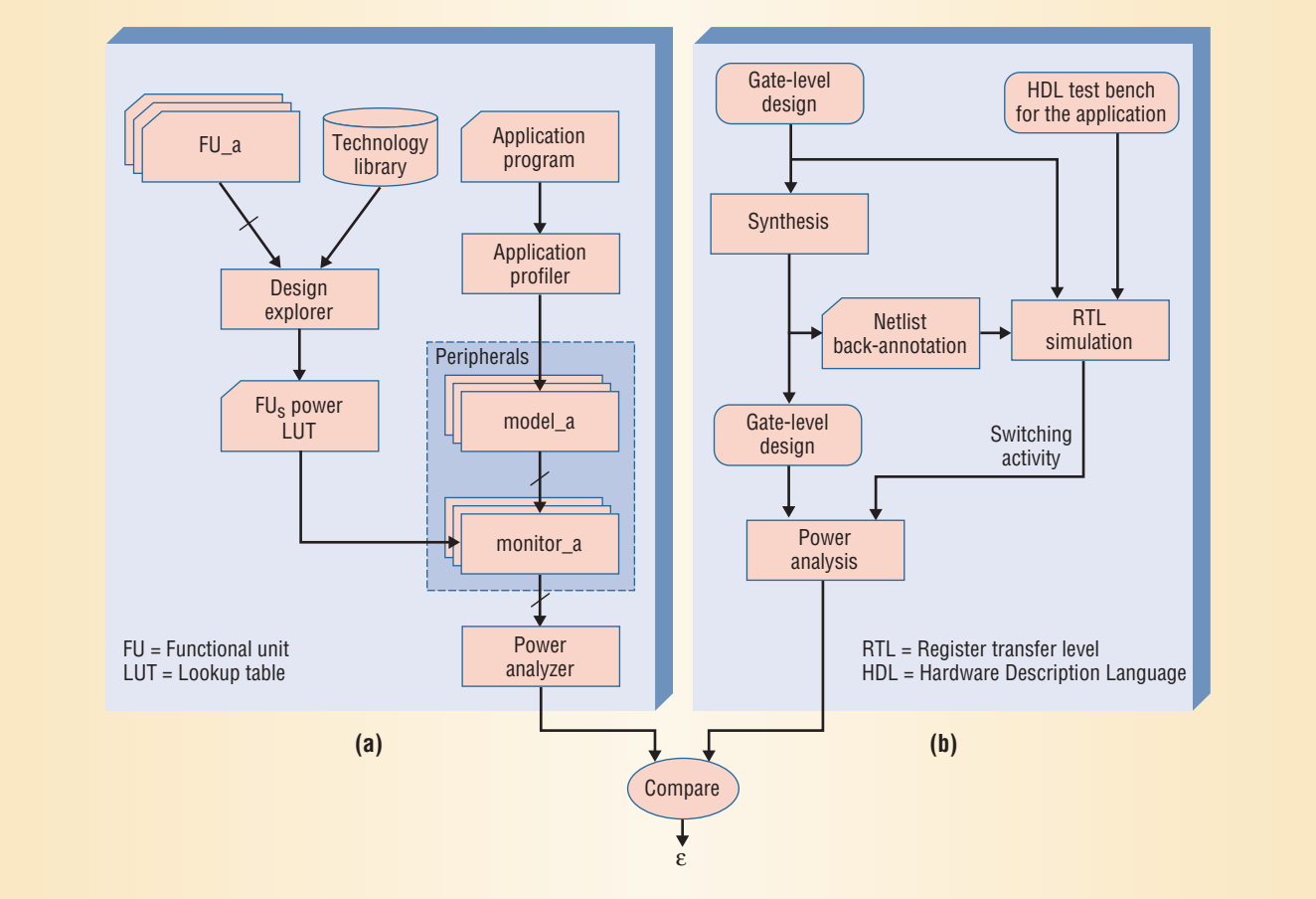

Figure 3. Two approaches to measuring power consumption in a baud generator. (a) System-level approach. (b) Gate-level approach.

Different models and monitors are associated with different peripherals or various refinements of the same peripheral. The monitors observe the associated models' execution and characterize their power behavior. The *power analyzer* collects the information that the monitors capture and computes power consumption.

A distinguishing feature of our system-level approach is that it does not require gate-level synthesis and simulation. However, to validate its effectiveness and efficiency, we compared our results against those obtained using gate-level power estimation, as shown in Figure 3b.

The experiments consisted of running 20 randomly generated application programs for 2,000 clock cycles. To perform a comparative analysis, we used a VHSIC Hardware Description Language model of the baud generator implemented at the register transfer level and then used Synopsys tools to synthesize it down to the gate level. To perform the gate-level power estimation, we used Synopsys power-estimation tools and a set of VHDL test benches generated by replicating the application programs.

Experimental results

Figure 4 summarizes the power per cycle dissipated by each instruction using both approaches. The average error is 9.71 percent, and the standard deviation is 9.36 percent. System-level estimation is consistently lower than gate-level estimation because the former always considers a lower level of detail than the latter. Power consumption underestimation represents a serious problem in scenarios that focus on worst-case design analysis rather than design tradeoffs.

Profiling energy consumption is particularly useful to gain insight into system hot spots. Figure 5a shows the energy the system consumes while executing three of the benchmarked application programs. Figure 5b shows a scatter plot of the power that all 20 benchmarks dissipated over 2,000 cycles. The average error is less than 20 percent, and the standard deviation is 8.26 percent.

Each point on the scatter plot depicts a benchmark's average power. The abscissa represents the average power obtained using our system-level approach, while the ordinate represents the average power obtained using the gate-level technique. If there was no difference between the two methods, all dots would lie on the solid line. Compared to gate-level power estimation, our approach achieves a speedup of 1,343—three orders of magnitude faster.

The primary goal of our approach is to make power-related system-level design decisions as early as possible in the design cycle. Therefore, 20 percent accuracy can be considered satisfactory

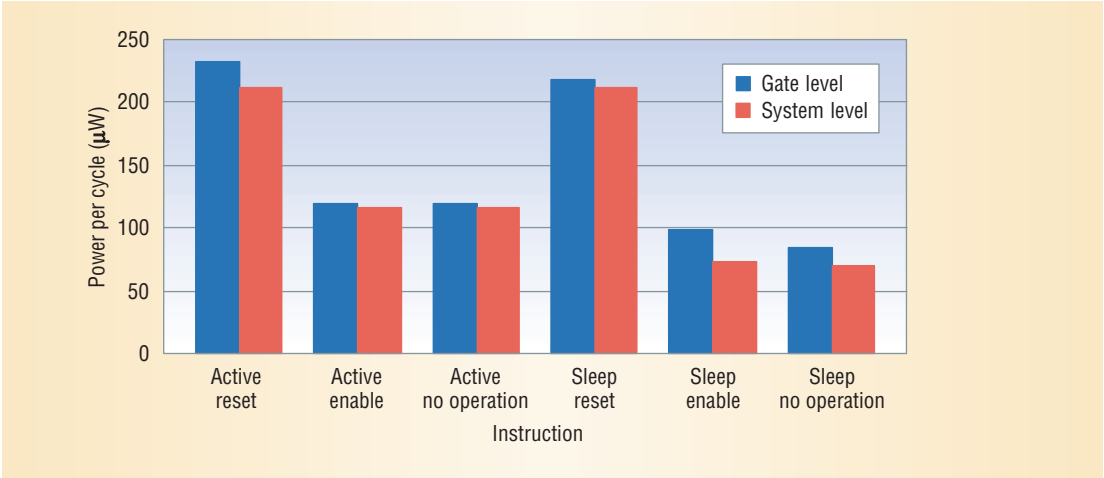

Figure 4. Power per cycle dissipated by each instruction. System-level estimation is consistently lower than gate-level estimation.

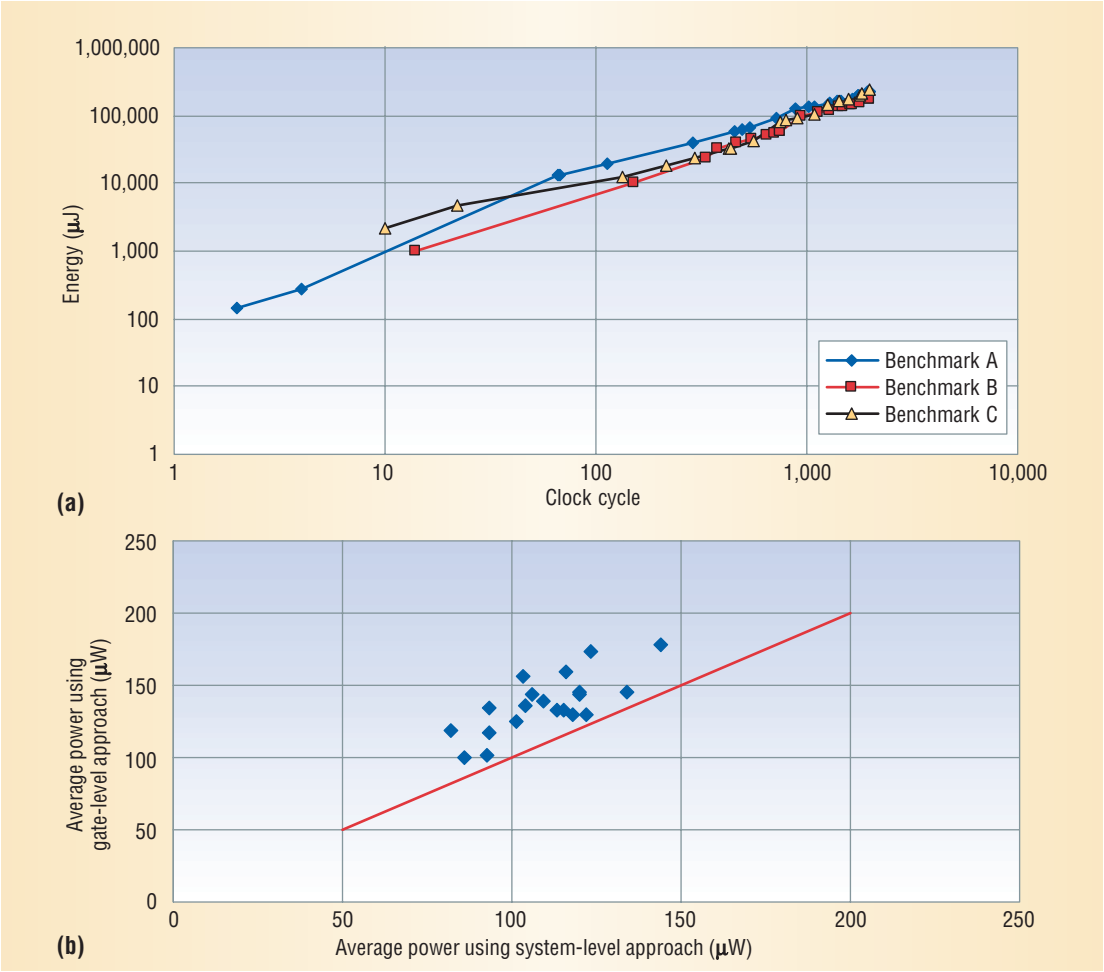

Figure 5. Energy consumption profile. (a) Plot of energy per clock cycles elapsed for three benchmarked applications. (b) Scatter plot of average power for 20 benchmarks over 2,000 cycles.

and, in fact, may be the only viable alternative when gate-level or transistor-level precharacterization is impossible. Indeed, at this level, the key is to provide fidelity—a high percentage of correctly predicted comparisons between design implementations—rather than very high estimation accuracy.

Future work will include validation on larger-scale designs and iterative refinements of the models based on earlier results. We also plan to extend the framework to as many performance indices as possible including response time, throughput, chip area, software size, and production costs. ■

References

1. A. Sangiovanni-Vincentelli and G. Martin, "Platform-Based Design and Software Design Methodology for Embedded Systems," *IEEE Design & Test of Computers*, Nov./Dec. 2001, pp. 23-33.

2. V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step toward Software Power Minimization," *IEEE Trans. VLSI Systems*, Dec. 1994, pp. 437-445.
3. S.M. Kang, "Accurate Simulation of Power Dissipation in VLSI Circuits," *IEEE J. Solid-State Circuits*, Oct. 1986, pp. 889-891.
4. T.H. Krodel, "PowerPlay—Fast, Dynamic Power Evaluation Based on Logic Simulation," *Proc. 1991 IEEE Int'l Conf. Computer Design: VLSI in Computers & Processors* (ICCD 91), IEEE CS Press, 1991, pp. 96-100.
5. S. Ravi, A. Raghunathan, and S. Chakradar, "Efficient RTL Power Estimation for Large Designs," *Proc. 16th Int'l Conf. VLSI Design* (VLSI 03), IEEE CS Press, 2003, pp. 431-439.
6. M. Nemani and F.N. Najm, "High-Level Area and Power Estimation for VLSI Circuits," *IEEE Trans. CAD of Integrated Circuits and Systems*, June 1999, pp. 697-713.
7. R.Y. Chen, M.J. Irwin, and R.S. Bajwa, "Architecture-Level Power Estimation and Design Experiments," *ACM Trans. Design Automation of Electronic Systems*, Jan. 2001, pp. 50-66.
8. A.C.S. Beck et al., "CACO-PS: A General-Purpose Cycle-Accurate Configurable Power Simulator," *Proc. 16th Symp. Integrated Circuits and Systems Design* (SBCCI 03), IEEE CS Press, 2003, pp. 349-354.
9. C-T. Hsieh et al., "Profile-Driven Program Synthesis for Evaluation of System Power Dissipation," *Proc. 34th Design Automation Conf.*, IEEE CS Press, 1997, pp. 576-581.
10. V. Dalal and C.P. Ravikumar, "Software Power Optimizations in an Embedded System," *Proc. 14th Int'l Conf. VLSI Design* (VLSI 01), IEEE CS Press, 2001, pp. 254-259.
11. C. Brandolese et al., "Static Power Modeling of 32-Bit Microprocessors," *IEEE Trans. CAD Integrated Circuits and Systems*, Nov. 2002, pp. 1306-1316.
12. T. Givargis, F. Vahid, and J. Henkel, "Instruction-Based System-Level Power Evaluation of System-on-a-Chip Peripherals Cores," *IEEE Trans. VLSI Systems*, Dec. 2002, pp. 856-863.
13. M.B. Kamble and K. Ghose, "Energy-Efficiency of VLSI Caches: A Comparative Study," *Proc. 10th Int'l Conf. VLSI Design: VLSI in Multimedia Applications* (VLSI 97), IEEE CS Press, 1997, pp. 261-267.
14. K. Itoh, "Trends in Low-Voltage Embedded-RAM Technology," *Proc. 23rd Int'l Conf. Microelectronics* (MIEL 02), IEEE Press, 2002, pp. 497-501.
15. W. Fornaciari, D. Sciuto, and C. Silvano, "Power Estimation for Architectural Exploration of HW/SW Communication on System-Level Buses," *Proc. 7th Int'l Workshop Hardware/Software Co-Design* (CODES 99), IEEE CS Press, 1999, pp. 152-156.

Claudio Talarico *is a research assistant professor in the Electrical and Computer Engineering Department at the University of Arizona. His research interests include design methodologies for integrated circuits and systems with emphasis on system-level design, embedded systems, HW/SW codesign, low-power design, system specification languages, and early design assessment, analysis, and refinement of complex SoCs. Talarico received a PhD in electrical engineering from the University of Hawaii at Manoa. He is a member of the IEEE Computer Society. Contact him at claudio@ece.arizona.edu.*

Jerzy W. Rozenblit *is a professor and heads the Electrical and Computer Engineering Department at the University of Arizona. His research interests are in the areas of complex systems design and simulation modeling. Rozenblit received a PhD in computer science from Wayne State University. He is a senior member of the IEEE Computer Society and the ACM. Contact him at jr@ece.arizona.edu.*

Vinod Malhotra *is an associate professor in the Department of Electrical Engineering at the University of Hawaii at Manoa. His research interests include wet and dry processes for passivation of GaAs and InP surfaces and surface-sensitive devices, such as HBTs, MSM photodetectors, and VCSELs. Malhotra received a PhD in electrical engineering from Colorado State University. He is a member of the American Vacuum Society, the Electrochemical Society, and the IEEE. Contact him at vinod@spectra.eng.hawaii.edu.*

Albert Stritter *is vice president of design automation at Infineon Technologies AG in Munich, Germany. His research focuses on all aspects of electronic design automation and technology integration. Stritter received an MS in electrical engineering from the Università degli Studi di Genova. He is a member of the Virtual Socket Interface Alliance and EDA (Electronic Design Automation) Zentrum, Hannover, Germany. Contact him at albert.stritter@infineon.com.*