Instruction-Based System-Level Power Evaluation of System-on-a-Chip Peripheral Cores

Tony Givargis, *Associate Member, IEEE,* Frank Vahid, and Jörg Henkel, *Senior Member, IEEE*

***Abstract*—Various core-based power evaluation approaches for microprocessors, caches, memories and buses have been proposed in the past. We propose a new power evaluation technique that is targeted toward peripheral cores. Our approach is the first to combine for peripherals both gate-level-obtained power data with a system-level simulation model written in an object-oriented language. Our approach decomposes peripheral functionality into so-called instructions. The approach can be applied with three increasingly fast methods: system simulation, trace simulation or trace analysis. We show that our models are sufficiently accurate in order to make power-related system-level design decisions but at a computation time that is orders of magnitude faster than a gate-level simulation.**

***Index Terms*—Low-power design, power estimation, system-level simulation, system-on-a-chip design.**

I. Introduction

As mobile computing devices have become more popular, minimizing average power and total energy consumption has become an important design goal. Furthermore, short product life cycles and increasing product complexity have led to core-based design paradigms. As a consequence, there is a strong demand for core-based power evaluation and optimization tools.

A core is a pre-designed processing-level component, such as a microprocessor, memory, or peripheral component like a direct-memory access controller, bus interface, or compression/decompression engine. A core may reside on a single chip with tens of other cores, forming a system-on-a-chip, or SOC. Cores typically have numerous parameters to increase the number of applications in which the core can be used. Example parameters include bit-widths and buffer sizes. An important but hard SOC design task is thus to configure the numerous and interdependent parameters of the SOC cores, such that the configuration is tuned to the executing software under power/performance constraints. Fast and accurate evaluation and optimization tools are needed to perform such tuning.

Manuscript received September 1, 2000; revised January 6, 2001, February 26, 2002, and July 4, 2002. This work was supported by the National Science Foundation (CCR-9811164), (CCR-9876006), a Design Automation Conference Graduate Scholarship, and NEC.

T. Givargis is with the Department of Information & Computer Science, Center for Embedded Computer Systems, UCI, University of California, Irvine 92697 USA (e-mail: givargis@ics.uci.edu)

F. Vahid is with the Department of Computer Science & Engineering, Center for Embedded Computer Systems, UCI, University of California, Riverside 92521 USA (e-mail: vahid@cs.ucr.edu)

J. Henkel is with the C&C Research Laboratories, NEC USA, Princeton, NJ 08540 USA (e-mail: henkel@ccrl.nj.nec.com)

Digital Object Identifier 10.1109/TVLSI.2002.808443

A core's power consumption may vary greatly depending on the application driving the SOC, and on the configuration of the core itself. Thus, the average-power tables provided in core data-sheets, even when extended to account for a subset of common configurations, may yield inaccurate power numbers for a particular application and configuration.

Therefore, researchers have proposed techniques for fast system-level power evaluation of various cores, including microprocessor, cache, memory and bus cores. In our efforts to develop a system-level power evaluation environment for parameterized SOC, we found however that no techniques existed to evaluate peripheral cores as fast and accurately as a combined gate-level/system-level model (i.e., executable specification) could provide. Our work uses such an approach and applies it to peripheral cores, namely those single-purpose processing cores that typically surround a microprocessor core.

II. Previous Work

Previous power evaluation work has been done at various abstraction levels, trading off accuracy for speed at higher levels. Logic-level approaches simulate a gate-level design and measure switching activity of design nodes [1], [2], executing orders of magnitude faster than circuit-level approaches [3], [4], but still requiring days to evaluate even one configuration. RTL (register-transfer level) approaches simulate an RTL design, consisting of coarser components like adders and multipliers, and compute power using power models of those components, known as macro-models. These approaches may use table-lookup techniques or analytical models. Early work was done in [5], using table-lookups, where each component was modeled via an N-variable characterization (input density, output density, switching-probability, etc.) of its power consumption [6], [7]. An N-dimensional lookup table is used to lookup the power consumption of an RTL component during simulation. Similarly, analytical models have been devised that compute power consumption of an RTL component given the actual input patterns or some form of input pattern characterization [8], [9]. Lookup tables and the coefficients of the analytical models are often derived from the gate-level circuit structure or lower level power evaluation and simulation. In [10], RTL power evaluation demonstrates accuracy of within 5%, but RTL approaches may still be too slow for extensive system-level exploration, especially considering that just synthesizing an RTL design may take hours, which must be repeated for each configuration.

At an even higher abstraction level, behavioral-level approaches estimate power of a behavioral HDL description before a synthesized design is obtained. Switching is estimated

using entropy from circuit input to circuit output by quadratic or exponential degradation [11], [12]. Such approaches, while fast, will not be nearly as accurate for cores as approaches that take advantage of the fact that cores can be presynthesized.

In [13], a system-level power estimation approach is introduced for mapping a workload to a set of resources, where the workload would be a measure of the computational activity and the resources represent the hardware. Each resource is modeled via a power-state machine that captures the power behavior of that component. A power management unit will translate a resource event and the simulation events into power-state machine transitions while accumulating power consumption.

Work has been done to evaluate power consumption of microprocessor cores. Instruction-level power modeling is proposed by [14]. Given a program execution trace, energy is computed as the sum of the energy consumed by each instruction that is executed, circuit state energy consumed when a particular instruction is followed by another, and energy consumed by other effects such as stalls and cache misses. This approach is sped up in [15] by deriving a shorter program trace that results in equal power dissipation. In [16], a mathematical generic power model for 32-bit microprocessors is proposed. The approach classifies the instruction set into classes like branches, etc. Other researchers have focused on fast system-level models for cache, memory and bus power consumption [17], [18], consisting mostly of equations that compute power consumption as a function of usage/traffic and core parameters. In [19], a cycle-accurate power simulation tool for a system with a microprocessor and memories is introduced, with accuracy within 5% of board measurements. A trace-based approach deploying a mix of analytical models for memories and instruction-set simulation is introduced in [20].

III. Power-Evaluation for Peripheral Cores

While previous system-level evaluation approaches for SOC components focused on microprocessors, cache, memory and buses, we now describe an approach for peripherals.

A. Overview

We have found that peripherals can be viewed as executing a sequence of "instructions." Classically, an instruction represents an atomic action available to a microprocessor programmer. We use "instruction" more generally as an action that, collectively with other actions, describes the range of possible behaviors of a core. We have extended the instruction-level power modeling approach that was previously used for microprocessor cores, for use with peripheral cores. In developing the approach, we noted that cores typically already come with system-level functional models, written in an object-oriented language like C++, or Java, and that in fact the VSIA requires such models in its standard [21].

We informally define the power evaluation problem as follows. Given a parameterized core [e.g., a Universal Asynchronous Receiver/Transmitter (UART)] of an SOC, we are to devise a high-level executable model, say in C++, of that core that can output energy consumption during a system-level simulation. This model must be sensitive to changes in the various parameters of that particular core. Our approach can be applied to each peripheral core in an SOC to obtain the total system energy consumption.

Our approach is broken into seven steps. Of these, the core provider performs the first five steps while the core user performs the last two steps. It is important to note that tasks performed by the core provider are done once while tasks performed by the core user are iterated until desired system power/performance constraints are met.

B. Core Provider Steps

The core provider steps are done for each target technology, and are only performed one time for each. The resulting data is used in any core-based design using the particular core. These steps may take days to complete, forming part of the months required to develop the core. Keep in mind that the core provider performs all the following steps manually. Furthermore, since each core may exhibit different power consumption characteristics, the core provider must manually fine-tune each step, in order to achieve desired accuracies.

Step 1: Selecting Peripheral Instructions: The core provider must first break the core's functionality into a set of *instructions*. Given an RTL model of a core, one first determines the system-level instructions of that core. These instructions must have the property that they collectively cover the entire functionality of the core. As with the instructions of an instruction-set processor, each instruction operates on some input data and produces some output data. For example, for a UART, one might select the following instructions: *Reset*, *Enable_tx*, *Enable_rx*, *Send*, and *Receive*.

In general, there is a tradeoff in choosing the right instructions for power evaluation: having many fine-grained instructions may lead to greater accuracy but longer simulation times than having fewer, coarse-grained instructions. Notice also that instruction creation is currently a manual process, requiring good knowledge of the core's behavior as well as the core's power consumption characteristics.

Step 2: Instruction Data Dependency Modeling: For each instruction, the core provider must determine how dependent the instruction's power consumption is on the instruction's input data. We thus define an instruction's power-dependency characteristic as one of: *dependent* directly on its input data, dependent on a *statistical* characterization of its input data (e.g., the density of 1's in a vector of bits), or *independent* of its input data. Such determination can be based on factory data-sheets, a core designer's knowledge, experimental results or statistical analysis. For example, for a UART example, we ran experiments that provided different data to each instruction, and we determined that the power-dependency characteristic for all instructions was independent. For example, the *Send* instruction consumes approximately a constant amount of energy regardless of the data being sent; likewise for the *Receive* instruction.

Step 3: Core Power-Mode Modeling: Very unlike microprocessors, certain instructions executed on a peripheral core can drastically change the power consumption of succeeding instructions. In particular, certain instructions change the mode of the peripheral core. This concept of mode is very different from that of measuring interinstruction power dependencies (e.g., a

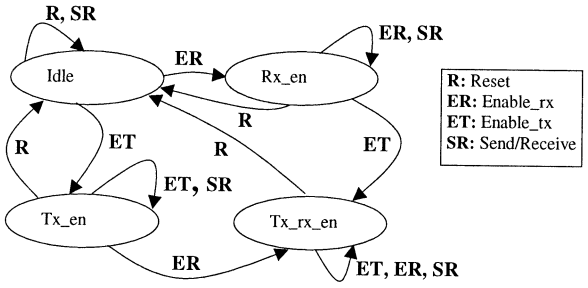

Fig. 1. UART's power-mode transition function.

load following a store may consume more power than a load following an add). To account for this, the core provider must determine the set of modes of a core, referred to as *power-modes* that cause the core to consume significantly more or less power per each execution of its instructions. In our UART example, we found four power modes: *Idle*, *Tx_enabled*, *Rx_enabled*, and *Tx_rx_enabled*. Given these modes, we define a power-mode transition function of a core, that gives the next power-mode given the current power-mode and the most recently executed instruction of that core. For a UART example, the power-mode transition function is shown in Fig. 1.

Step 4: Gate-Level Power Evaluation: Here, we use gate-level simulation to obtain per-instruction energy consumption data for the lookup-tables. Given an RTL model of a core, its instructions, parameters, and modes, we follow the procedure outlined in Algorithm 1.

Algorithm 1:

```
for p ∈ Parameter-Space do
 for m ∈ Power-Mode-Space do
  for i ∈ Instruction-Space do
   if i = independent then
    create test-bench simulating i with
data ← random, mode ← m, parameter ← p
   elseif i = statistical then
    for
s ∈ Statistical-Characterization-Space do
    create test-bench simulating i with
data ← s, mode ← m, parameter ← p
   elseif i = dependent then
    for d ∈ Data-Space do
    create test-bench simulating i with
data ← d, mode ← m, parameter ← p
```

This procedure gives a systematic way of creating a set of test-bench models that, when simulated at gate-level, capture the energy consumption of a particular instruction, in a particular mode with a particular parameter setting. Note that the procedure has different actions for the different power-dependency characteristics introduced in step 2. We then simulate each test-bench with the core's gate-level model, and we analyze the energy consumption of the corresponding instruction, mode and parameter value. We tabulate these energy results into our lookup tables. The following table gives the lookup energy values (μJ) for the UART example. The rows correspond to instructions while columns correspond to the UART's buffer size

TABLE I

Buffer size (byte) →	2	4	8	16	2	4	8	16
	Idle				**Tx_enabled**			
Reset	11	13	14	14	1	13	14	14
Enable_tx	27	32	31	31	23	23	22	24
Enable_rx	17	18	19	18	19	20	19	19
Send	17	19	19	20	133	135	157	209
Receive	14	15	17	18	14	16	18	18
	Rx_enabled				**Tx_rx_enabled**			
Reset	13	14	15	15	13	13	14	14
Enable_tx	22	22	21	21	21	22	21	21
Enable_rx	18	19	18	18	19	19	19	19
Send	19	19	21	23	133	135	157	209
Receive	73	83	93	111	74	81	93	107

parameter values. The entries are repeated for each one of the four modes as shown in Table I.

Step 5: System-Level Modeling: Here, we develop a system-level model of each core that enables rapid power evaluation when executed (i.e., an executable specification). Given an RTL model of a core, its instructions, and its modes, we implement a functional model of the core in terms of its instructions. If using method-calling objects [22], the interface to the object representing the core would have the instructions as methods and the instruction's input/output data as parameters. To each object-oriented model, we add two data objects, called *core_energy* (initialized to zero) and *power_mode* (initialized to *Idle*.) We then augment the implementation of each method of the core's system level model with the code in Algorithm 2.

Algorithm 2:

```
// i: the current instruction, m: current
mode, p: current parameter, d: instruc-
tion's data
behavior();
power_mode ← power_mode_table
[power_mode,i];
if i = independent then
 core_energy ← core_energy + energy_table
[p,power_mode]
elseif i = statistical then
 core_energy ← core_energy + energy_table
[p,power_mode,stats(d)]
elseif i = dependent then
 core_energy ← core_energy + energy_table
[p,power_mode,d]
```

C. Core User Steps

Step 1: Connecting the System-Level Core Model: During this step, the core user selects components from the core library and connects them according to the system-level model organization.

Step 2: System-Level Power Evaluation: Here, the core user simulates the complete SOC. This can take on the order of seconds or minutes. Thus, hundreds or thousands of configurations

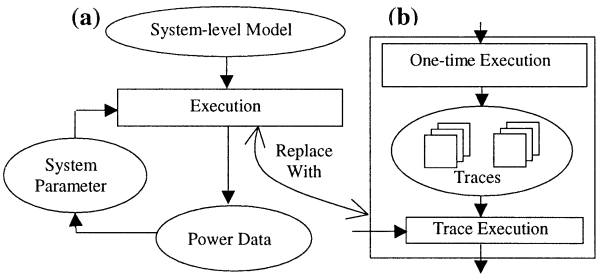

Fig. 2. (a) Functional model, (b) trace-driven model.

can be evaluated. The top-level simulation model will be designed to output the value of the *core_energy* variable, for each core of the system, at the end of each simulation. Likewise, the sum of all *core_energy* values represents the system-level estimate of the SOC energy consumption for a given configuration of its parameters. An estimate of the SOC average power consumption is obtained by dividing the total SOC energy consumption by the execution time.

D. Trace-Driven Evaluation Approach

Processing instruction traces via trace simulators, instead of using a full functional simulator, can speed up the system-level power evaluation approach described above. Our trace-driven approach is inspired by similar work applied to caches and processors. Cache simulators intended for power or performance evaluation typically work off of address reference traces. Processor simulators intended for power evaluation also have recently been developed using instruction traces.

We define a trace, with respect to a core, to be a sequence of instructions with accompanying data items that are executed by that core during its functional simulation. We extend the above simulation-based approach by converting the functional models of the cores to nonfunctional, or partially functional, models. These models operate on a trace. We refer to such nonfunctional models as *trace simulators*.

Fig. 2 shows the functional-simulation-based approach as well as the trace-simulator-based approach. Using trace simulators, a core user simulates a system once to obtain the trace files for each core. These trace files are subsequently processed using trace simulators to obtain power and explore various core parameter effects. Trace-driven simulators are significantly faster than full functional simulators.

We now describe how to construct these trace-driven simulation models for general peripheral cores. First, given a system level functional model of a core, the core developer augments the implementation of each method in that model with code that will append to a trace-file a unique id for that instruction and the corresponding instruction data. When executed, such a model will output a set of traces, one per each core in the system, that is subsequently used by trace simulators as described next.

Given an RTL model of a core, its instructions, and its modes, we implement a nonfunctional model of that core in terms of its instructions. If using method-calling objects, the interface to the object representing the core would have the instructions as methods and the instruction's input/output data as parameters to the corresponding methods. To each object-oriented model, we add two data items: *core_energy* (initialized to zero) and *power_mode* (initialized to Idle). The implementation of each function consists of the code fragment presented in Algorithm 2, but excluding the functional implementation (i.e., the call to *behavior* routine).

Each core's object-oriented model is then designed to read the corresponding trace file and execute the instructions of it accordingly. The top-level simulation model will be designed to output the value of the *core_energy* variable, for each core of the system, at the end of each simulation. Likewise, the sum of all *core_energy* values represents the system-level estimate of the SOC energy consumption for a given configuration of its parameters. An estimate of the SOC average power consumption is obtained by dividing the total SOC energy consumption by the execution time.

E. Trace Analysis

We can further speed up the power evaluation time for cores by reducing the size of the trace files, therefore reducing the processing time required to evaluate power consumption. Our technique is similar in idea to those in [15] intended for microprocessors, but simpler (since microprocessor instruction traces are more complex). Here, we will outline similar approaches for speeding up our trace-driven power evaluation approach for peripheral cores. They are to be compared with the *full trace* approach of the previous section, in which the traces store each instruction along with its complete input data.

In the *reduced trace via characterized-data* approach, rather than storing complete parameter data, we store a statistical characterization of that data. For example, we can store the data-density, defined as the ratio between the number of bits that are set to the total number of bits. Density has been shown to be a good predictor of power in many components, and our own experiments support this.

In the *reduced trace via instructions only* approach, we store the instruction only, without any parameter data. We can take this approach if we determine that power consumption is mostly independent of an instruction's data. Note that we can apply the above trace reductions to the entire trace file, i.e., all instructions, or to selected instructions. Thus, Fig. 3 shows code that can use a different method for each type of instruction.

In the *reduced trace via instruction-frequency* approach, we combine a sequence of instructions that are identical or have identical power consumption into a single instruction augmented with a frequency value. We could further annotate each instruction with a statistical characterization of the data accompanying the combined instructions. The instruction-frequency approach is an area of future work, and could be extended in the direction of [15].

IV. Experiments

We have performed numerous experiments to verify the accuracy, and simulation speedup obtainable by using the power modeling approach presented in this work. We outline our experimental setup and results next.

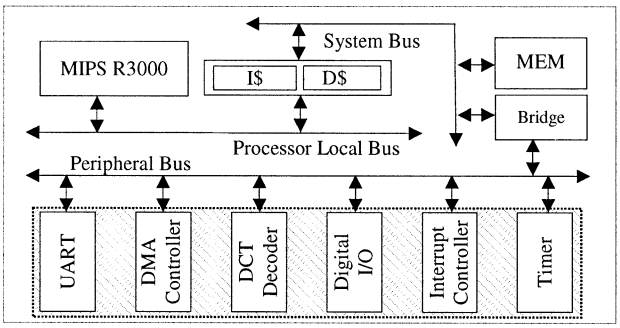

Fig. 3. Target SOC architecture.

TABLE II

Peripheral	Configurations	Description
UART	4	Buffering (2, 4, 8, 16 bytes)
DMA Controller	4	Block Transfer Size (4, 16, 64, 128 bytes)
DCT Decoder	2	Pixel Resolution (10, 12 bits)
Digital I/O	2	mode (bi-directional, uni-directional)
Timer	2	Resolution (8-bit, 16-bit)

TABLE III

Peripheral	Ideal Power-Modes	Description
UART	4	Idle, Tx_enabled, Rx_enabled, Tx_rx_enabled
DMA Controller	4	Idle, Unprimed, Primed, Busy
DCT Decoder	4	Idle, PixelPush, PixelPop, Busy
Digital I/O	2	Idle, Enabled
Timer	2	Idle, Enabled

A. Target SOC Architecture

The target SOC architecture used in our experiments is shown in Fig. 3. Here, a MIPS R3000 processor is connected to instruction and data caches via the processor local bus. In turn, the caches are connected to on-chip main memory via the system bus. The MIPS, caches, memory, and associated buses collectively constitute the processor subsystem of the SOC. The processor subsystem is connected to the peripheral bus via a bridge component. Numerous peripherals reside on the peripheral bus. These are: UART, Direct Memory Access (DMA) controller, Discrete Cosine Transform (DCT) decoder, external peripheral I/O controller (called Digital I/O to distinguish it from the use of the term "peripheral" referring to SOC peripherals), and timer. In our experiments, we focus on power estimation for peripheral components only, shown as shaded in Fig. 3. Each SOC peripheral is parameterized. The number of configurations and a brief description of parameters are given in Table II.

B. Experimental Setup

We have implemented a system-level model of the above SOC in C++. Our implementation is augmented with the power-model approach described in this paper. The energy lookup tables have been obtained using the technique mentioned earlier in this work. A VHDL model of each of the peripheral is used to obtain the energy lookup tables. The VHDL models have been captured at RTL and synthesized down to gate using Synopsys synthesis tools. Gate-level power estimation is performed using Synopsys power estimation tools. To ensure relative accuracy, the same compiler directive (medium optimization effort) and library binding (LSI_10K) has been used for all synthesis and gate-level estimation runs. Each peripheral has been modeled using the following ideal[1] number of power-modes (see Table III).

For our experiments, five different benchmarks have been used: a digital camera application processing a black/white image of planet earth (*digcam-earth*), a digital camera application processing a color image of a car (*digcam-car*), a pseudo-application designed to utilize all peripherals with random instruction and random data (*utilize-random*), a pseudo-application designed to utilize each peripheral with 3000 instruction calls and data set to zero (*utilize-fixed0*), and a pseudo-application designed to utilize each peripheral with 3000 instruction calls and data set to one (*utilize-fixed1*). The following table gives utilization and data characteristics of each of the five benchmarks. These examples were chosen to provide a range of inputs and determine if our estimation method would maintain accuracy under variety of inputs as shown in Table IV.

[1]In some experiments a less than ideal number of power-modes is used to measure estimation accuracy sensitivity.

C. Results

Part 1: Estimation Accuracy: Here, we compared SOC energy consumption (the peripheral subsystem) results obtained using our approach to those obtained using gate-level estimation. Our results are summarized in Fig. 4. The experiments are averaged over all possible configurations of the SOC cores. Furthermore, we used the ideal number of power-modes for this part of the experiment. The percent error for *digcam-earth*, *digcam-car*, *utilize-random*, *utilize-fixed0*, *utilize-fixed1*, was 5.9%, 4.4%, 4.4%, 6.2%, and 10% respectively. The average error was 6.2%. The maximum error of 10% occurred with *utilize-fixed1*, which executed all instructions with data set to one. The minimum error of 4.4% occurred with *digcam-car*, which executed the longest, and *utilize-random*, which executed with random data patterns.

Part 2: Power-Mode Sensitivity: Here, we examined the importance of power-modes. We used the UART peripheral, a fixed configuration, and three different power-mode selections, namely, one power-mode, two power-modes, and four power-modes. We ran all five benchmarks. The energy consumption of the UART core and the percent error relative to gate-level estimation is shown in Table V.

Results show that the error increases as the number of power-modes is reduced from the ideal. Thus, a proper selection of power-modes is important for accuracy.

Part 3: Instruction Sensitivity: Here, we examined the importance of proper instruction granularity selection. We used the UART peripheral, a fixed configuration, and four power-modes. Furthermore, we aggregated the *Send* and *Receive* instructions

TABLE IV

Total Core Utilization					Data Pattern				
digcam-earth	digcam-car	utilize-random	utilize-fixed0	utilize-fixed1	digcam-earth	digcam-car	utilize-random	utilize-fixed0	utilize-fixed1
26807 calls	40207 calls	15000 calls	15000 calls	15000 calls	image-dependent	image-dependent	random	all bits 0	all bits 1

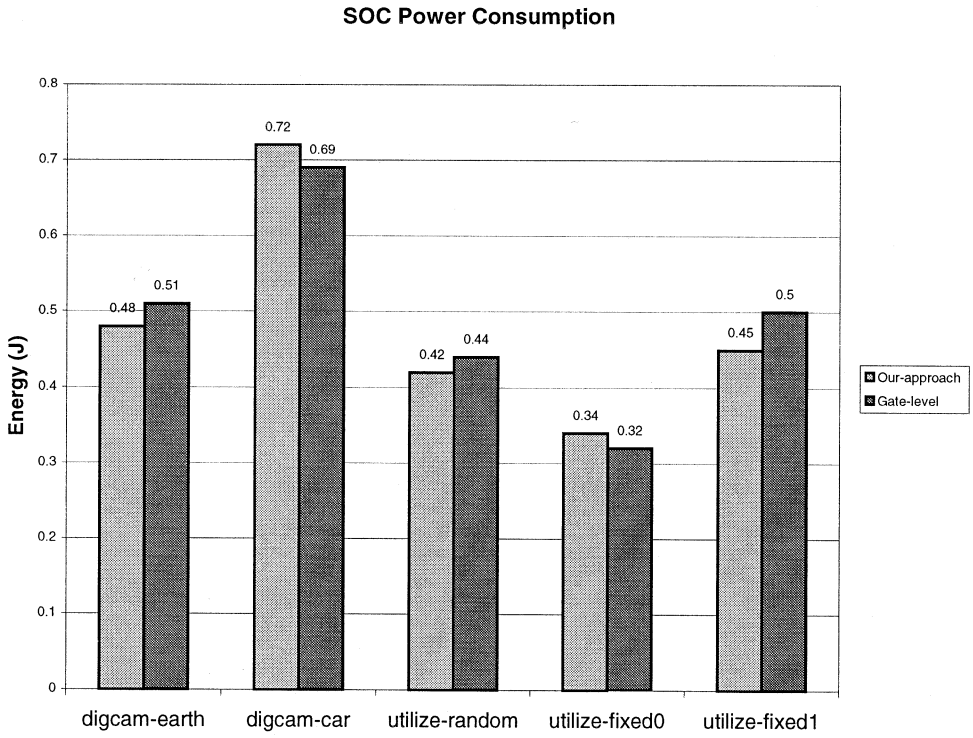

Fig. 4. Power estimation comparison.

TABLE V

Estimation Approach	Energy/error (mJ/%)				
	digcam-earth	digcam-car	utilize-random	utilize-fixed0	utilize-fixed1
Gate-level	64/-	87/-	35/-	30/-	44/-
4-Mode System-level	68/6.3	89/2.3	37/5.7	28/6.7	47/6.8
2-Mode System-level	71/11	94/8.0	40/14	33/10	49/11
1-Mode System-level	75/17	101/16	42/20	35/17	50/14

TABLE VI

Estimation Approach	Energy/error (mJ/%)				
	digcam-earth	digcam-car	utilize-random	utilize-fixed0	utilize-fixed1
Gate-level	64/-	87/-	35/-	30/-	44/-
4 Instructions	68/6.3	89/2.3	37/5.7	28/6.7	47/6.8
3 Instructions	76/19	104/20	41/17	34/13	51/16

into a single *Send_or_receive* instruction. We ran all five benchmarks. The energy consumption of the UART core and the percent error relative to gate-level estimation is shown in Table VI.

Results show that the error increases as the instruction granularity is reduced from the ideal. Thus, a proper selection of instructions is important for accuracy.

Part 4: Trace-File Reduction: Here, we experimented with trace file reduction. We selected two benchmarks, *digcam-earth* and *digcam-car*. For each core, we used a fixed configuration, and the ideal number of power-modes. The results are provided in Table VII.

For each benchmark we measured the sizes of the trace files for the full-trace (*full-trace*), the reduced trace via characterized data (*reduced-trace-c*), and the reduced trace via instructions only (*reduced-trace-i*). Also, we compared the CPU time required to evaluate power consumption, comparing gate-level simulation (*gate-level*), functional system-level simulation (*sys-level*), *full-trace*, *reduced-trace-c*, and *reduced-trace-i*. We note that the file size of *full-trace* was 9 times larger than the file size of *reduced-trace-c* and 64 times larger than the file size of *reduced-trace-i*. Likewise, evaluation time (i.e., CPU cycles required to perform the simulation) using a trace file was on the average 5 times faster than *system-level* simulation. In terms of power, the error using a trace files was on the average less than 1%.

TABLE VII

Trace Size (Kb)			Evaluation Time (sec)					Energy (J)			
full-trace	reduced-trace-c	reduced-trace-i	gate-level	system-level	full-trace	reduced-trace-c	reduced-trace-i	gate-level	full-trace	reduced-trace-c	reduced-trace-i
digcam-earth											
21	2.4	0.33	274K	32	17	2.0	0.3	0.51	0.48	0.49	0.53
digcam-car											
32	3.6	0.5	290K	48	26	2.9	0.4	0.69	0.72	0.75	0.76

V. Conclusion

We have introduced an instruction-based system-level technique for fast and accurate power evaluation of peripheral cores. The technique can be used in conjunction with those previously developed for microprocessors, caches, memories and buses, to achieve power evaluation of systems-on-a-chip, and complements evolving system-level modeling standards. We showed the importance of the power-mode concept, and showed that data-sheet lookup based approaches can be inaccurate. In addition, similar to microprocessor and cache trace-simulator approaches, we have shown a method for using peripheral instruction traces and trace simulators to further speedup power evaluation. Further work may focus on automating the characterization phase of our technique.

References

[1] R. Tjarnstorm, "Power dissipation estimate by switch level simulation," in *Proc. Symp. Circuits and Systems*, 1989.

[2] T. H. Krodel, "Powerplay – Fast dynamic power evaluation based on logic simulation," in *Proc. Int. Conf. Computer Aided Design*, 1991.

[3] S. M. Kang, "Accurate simulation of power dissipation in VLSI circuits," *IEEE J. Solid-State Circuits*, vol. SSC-22, Oct. 1986.

[4] G. Y. Yacoub and W. H. Ku, "An accurate simulation technique for short-circuit power dissipation based on current component isolation," in *Proc. Int. Symp. Circuits and Systems*, 1989.

[5] A. Raghunathan, S. Dey, and N. K. Jha, "Register-transfer level evaluation techniques for switching activity and power consumption," in *Proc. Int. Conf. Computer Aided Design*, 1996.

[6] S. Gupta and F. Najm, "Power macromodeling for high level power evaluation," in *Proc. Design Automation Conf.*, 1997.

[7] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, and G. De Micheli, "Lookup table power macro-models for behavioral library components," in *Proc. Design and Test in Europe*, 1998.

[8] P. Landman and J. Rabaey, "Architectural power analysis: The dual bit type method," *IEEE Trans. VLSI Syst.*, June 1995.

[9] H. Mehta, R. Owens, and M. J. Irwin, "Energy characterization based on clustering," in *Proc. Design Automation Conference*, 1996.

[10] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, evaluation, and optimization," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1061–1079, Nov. 1998.

[11] D. Marculescu, R. Marculescu, and M. Pedram, "Information theoretic measures for power analysis," in *Proc. Int. Conf. Computer Aided Design*, 1996.

[12] M. Nemani and F. Najm, "Toward a high level power evaluation capability," in *Proc. Int. Conf. Computer Aided Design*, 1996.

[13] L. Benini, R. Hodgson, and P. Siegel, "System-level power estimation and optimization," in *Proc. Int. Symp. Low Power Electronics and Design*, 1998.

[14] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 437–445, Dec. 1994.

[15] C. T. Hsieh, M. Pedram, H. Mehta, and F. Rastgar, "Profile driven program synthesis for evaluation of system power dissipation," in *Proc. Design Automation Conf.*, 1997.

[16] C. Barndolese, W. Fornaciari, F. Salice, and D. Sciuto, "Energy evaluation for 32-bit microprocessor," in *Proc. Int. Workshop on Hardware Software Codesign*, 2000.

[17] R. J. Evans and P. D. Franzon, "Energy consumption modeling and optimization for SRAM's," *IEEE J. Solid-State Circuits*, vol. 30, pp. 571–579, May 1995.

[18] T. Givargis, J. Henkel, and F. Vahid, "Interface and cache power exploration for core-based embedded system design," in *Proc. Int. Conf. Computer Aided Design*, 1999.

[19] T. Simunic, L. Benini, and G. De Micheli, "Cycl-accurate evaluation of energy consumption in embedded systems," in *Proc. Design Automation Conf.*, 1999.

[20] Y. Li and J. Henkel, "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems," in *Proc. Design Automation Conf.*, 1998.

[21] (1997) Architecture Document. Virtual SOCket Interface AsSOCiation. [Online]. Available: http://www.vsi.org

[22] F. Vahid and T. Givargis, "Incorporating cores into system-level specification," in *Proc. Int. Symp. System Synthesis*, 1998.

Tony Givargis (S'00–A'01) received the B.S. degree in computer science and the Ph.D. degree from the University of California, Riverside, in 1997 and 2001, respectively, where he was a GAANN (Graduate Assistance in the Area of National Need) Fellow. He received the department's best Thesis Award.

He is currently an Assistant Professor in the Department of Information and Computer Science at the University of California (UC), Irvine. He is also a member of the Center for Embedded Computer Systems at UC Irvine. He is a coauthor of the textbook *Embedded System Design* (New York: Wiley, 2002). His research focuses on platform-based system design, real-time resource management of embedded computing systems, and design space exploration.

Frank Vahid received the B.S. degree in computer engineering from the University of Illinois, Urbana, in 1988 and the M.S. and Ph.D. degrees from the University of California, Irvine, in 1990 and 1994, respectively, where he was an SRC Fellow.

He is currently an Associate Professor in the Department of Computer Science and Engineering at the University of California, Riverside, where he received the Outstanding Teacher of the College of Engineering award in 1997. His is also a faculty member at the Center for Embedded Computer Systems at UC Irvine. He is coauthor of the textbooks *Embedded System Design* (New York: Wiley, 2002). and *Specification and Design of Embedded Systems* (Upper Saddle River, NJ: Prentice Hall, 1994). His current research focuses on architectures and design methods for low-power embedded systems, with an emphasis on tuning system-on-a-chip platforms to their executing programs.

Dr. Vahid received the Best Paper Award from IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS in 2000. He was Program and General Chair for the IEEE/ACM International Symposium on System Synthesis in 1996 and 1997, respectively, and for the IEEE/ACM International Workshop on Hardware/Software Codesign in 1999 and 2000.

Jörg Henkel (M'85–SM'91) is currently a Senior Research Staff Member at the Computers & Communications Research Laboratories, NEC USA, Princeton, NJ, where he is leading several projects in research and development of low power SOC architectures and EDA tools. He is currently the General Co-Chair of the IEEE/ACM International Symposium on Hardware/Software Co-Design (2002) and has served for the same event as a Program Co-Chair in 2001. In addition, he currently serves as a Program Co-Chair for the IEEE International Workshop on Rapid System's Prototyping 2002. He is involved in the following program committees: IEEE/ACM International Symposium on Hardware/Software Co-Design (Codes), IEEE/ACM Design Automation and Test in Europe Conference (DATE), IEEE International Symposium on Low Power Electronics and Design (ISLPED).