SEA: Fast power estimation for micro-architectures

Praveen Kalla[†]
Dept. of Computer Sci. and Engg.
University of Notre Dame
NotreDame, IN - 46556, USA.
e-mail: nkalla@cse.nd.edu

Jörg Henkel
NEC CCRL
Princeton, NJ-08540, USA
e-mail: henkel@nec-lab.com

Xiaobo Sharon Hu[†]
Dept. of Computer Sci. and Engg.
University of Notre Dame
NotreDame, IN - 46556, USA
e-mail: shu@cse.nd.edu

Abstract— Various approaches for micro-architectural power/ energy estimation have been introduced, mainly driven by the need to obtain fast power/energy estimates during early phases of complex SOC designs. In contrast to previous approaches we study power/energy estimation for highly optimized synthesizable description of microprocessor cores. Under this real-world design scenario, we found, unlike related previous research, that power can hardly be estimated closer than around 15% using an instruction level model. However, we can estimate the *energy* as close as 5%. Our research has resulted in the SEA framework that estimates energy/power consumed by a software program, taking specific micro-architectural features of the underlying programmable hardware core into consideration. With this high accuracy in energy estimation we achieve around 5 orders of magnitude faster estimations compared to state-of-the art high-level (RTL) commercial energy/power estimation tool suites. Thus, our framework is capable of reliably estimating the energy/power consumption of future complex SOCs.

I. Introduction

IP-based design methodologies combined with the paradigm of platforms for specific application areas have enabled designers to design new multi-million gate designs in shorter times and at an overall smaller man-month count compared to traditional design method that do not extensively re-use existing IP. Examples of design platforms stem from domains like multimedia processing, wireless communications, real-time control, etc. The task of a designer has changed to integrating and estimating various scenarios of a future complex SOC by means of re-using existing IP and design platforms.

A high-level power estimation method should preferably have the following features:

1) The model should be able to estimate on a per-cycle basis to allow sufficient accuracy.
2) The model should be independent of access and switching activities as they can hardly be predicted from an instruction-level abstraction point of view. This holds especially for highly optimized processor designs that are coded in a mixed behavioral-RTL and structural-RTL fashion.

In this paper we provide an approach according to these constraints and under the assumption that the processor design **might not allow for functional clock gating**, i.e., clock gating that allows RTL blocks at almost any size to be gated, which is true for many real-world processor designs which rigorously mix functional and structural RTL. Block-based power estimation approaches might not be applicable in such cases.

Our model is based on the observations made by studying the optimized synthesizable RTL code of the MicroSparcIIep [25]. Our approach estimates *energy* within an accuracy of 5% and per-cycle *power* within 15% or less on an average.

A review of existing models is given in Sec. II. Our power model is introduced in Sec. III and IV. The SEA framework is discussed in Sec. V. Experiments and results are shown in Sec. VI with our conclusions in Sec. VII.

II. Review of Existing Power Models

A large number of approaches have been proposed for power estimation in recent years. We classify them as follows:

Module-based approaches view the power consumption of a core as a sum of the power consumption of the structural modules present in the core. Modeling the power behavior of these individual units provides the energy consumed every cycle by the core. [2, 4] present such power estimators which estimate the *activity factors*, area, capacitance, etc. for modules present in the core. [7] highlights the differences between these estimators and the difficulty in estimating accurate activity factors. [23] present SimplePower which is another such estimator based on the SimpleScalar tool suite [3]. Another estimator for SH3 is presented in [21]. [10] attempts a slightly broader classification of the core as *datapath*, *control logic*, etc. Such models require an in-depth knowledge of the architecture and the implementation, which, however, are not often provided by the IP providers. Further, modules in highly optimized IP cores can have complex operation inter-dependencies which make module partitioning difficult.

Instruction-based approaches abstract away the low-level details needed by the module-based models above. The energy consumption of the core is captured by assigning power/energy values to each instruction of the instruction set. The first of such approaches has been presented in [22]. The power behavior of each instruction is captured by executing the instruction in a loop and measuring the current drawn by the chip. Using a measurement-based approach helps account for (physical) packaging issues, but it is difficult to back-annotate the system-level loading to the activities on the pins of the chip. Furthermore, the process of measuring the average current *accurately* is a relatively complex and error-prone process [12]. More importantly, a SoC designer generally needs to obtain energy/power data *before* the chip is taped out.

In [8], each instruction is *propagated* along a gate-level net list for accurate energy estimation which is relatively time-consuming. Many models are proposed by studying the various aspects of instructions that might effect the core, such as data, operands, etc. [18] attempts to capture data related effects through *activity indices*. Another data-related study is presented in [5]. *Energy-sensitive* factors are studied in [6] and a regression based analysis is presented in [11]. Another model is presented in [20] which is based on classifying the execution cycles into different types. Most of these models

[†]This work is supported in part by NSF under grant numbers MIP-9701416 and CCR02-08992.

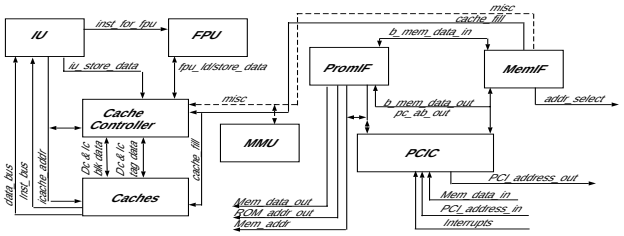

Fig. 1. Block diagram of the MicroSparcIIep synthesizable RTL core

are either based on the observations made for physical chips rather than IP cores (see the paragraph above), or still require the knowledge of the design and implementation.

Function-based approaches provide a yet higher abstraction for power estimation. Works like [1, 16, 17] abstract the processor as a set of *functions or stages*. Behavior of various physical modules is captured by these abstract quantities. A technique for capturing these activities through *performance counters* is presented in [9]. Some processors might not offer such functionality and in some cases, it might not be possible to monitor certain power events accurately. A yet higher level of abstraction is capturing the power behavior for *library routines* [14]. The primary difficulty with such an approach is to capture the statistical run-time behavior of these libraries accurately (the cache misses, etc.). At the most abstract level, the processor can be treated as a black box. [19] presents a cycle-level estimator where an ARM processor is assigned two states, active and nop-waiting. The power values for both states are taken from datasheets. Another such model is presented in [15]. Such a coarse treatment may not be applicable to a IP core and might lead to imprecise estimates.

As a summary, some previous approaches have either made simplified assumptions and do not model the micro-architecture in sufficient detail; or, the models are very detailed but the assumptions of the micro-architecture are rather un-realistic since a highly-optimized micro-architecture *cannot* be modeled as a set of RTL blocks that are either active or non-active. In fact, RTL blocks might dissipate energy during a certain time frame even though a functional simulation does not suggest so. Hence, the idealistic assumption of module-based gated-clock designs is not in compliance with optimized micro-architectures that actually unveil a mixture of structural and behavioral RTL.

The approach we take is to incorporate as many details as possible to extract from an optimized synthesizable RTL description. This inhibits not only the decomposition into blocks, but it may lead to a partly unpredictable energy/power behavior. Overall, however, we can *guarantee* an upper and lower bound for the estimated *power* that is typically in the range of 15% and around 5% accuracy for *energy* estimates.

III. Deriving and Refining an Instruction-Based Energy/Power Model

To derive the power/energy models for IP cores, we use a publicly available, synthesizable model of the MicroSparcIIep core [25] as an example. Employing a commercial core allows us to study many architectural features that typically do not appear in simple processor models constructed for research purposes only. The block diagram of a MicroSparcIIep is shown in Fig. 1 [13]. It is a RISC architecture that integrates a SPARC processor with a floating-point unit (FPU), memory management unit (MMU), separate instruction and data caches, and a PCI bus controller (PCIC) onto a single device.

The synthesizable RTL of the core is highly optimized such that behavioral and structural RTL are inter-mingled extensively. Further, functional clock-gating had not been implemented. Both these factors prevented the adoption of a block-based power model as used by some estimation techniques. Gate-level power estimation is a very time consuming process and is definitely not recommendable for estimating software energy. However, for our initial investigation, we synthesized the core and conducted gate-level simulation in order to verify some conclusions presented in similar previous work.

Based on extensive detailed experiments, we observe that the power/energy variations due to architectural characteristics are quite complex. For example, we note that modules that are not directly involved in the execution of an instruction can still contribute significantly in terms of power variation. The data in Tab. I clearly show this point. Here, the first column lists pairs of instructions, and the second column summarizes the corresponding power differences of various modules. The acronyms used are Core (all modules except caches), IU (integer unit), Ex (execution unit), Rf (register File), CC (cache controller), MMU (memory management unit), Memif (memory interface unit). We use *x.y* to denote that y is a sub-module of x. From Tab. I, one can readily see that units such as MMU and Memif account for a large portion of the total power difference (e.g., more than 30% for Add v.s. And), even though the execution of both instructions is not supposed to involve the two units. Such behaviors are also observed in the FPU (especially the floating point register file) during the execution of integer instructions, which is due to the partial decoding of all instructions in the FPU. Instructions are later discarded if they are not FP-instructions.

The intricate dependencies of instruction executions on functional units make it impossible to model the power consumption of the core by means of a simple module decomposition. Moreover, even the most complex model would not accurately estimate the power consumption since many effects are simply due to the structural RTL coding style of the core. To overcome these difficulties, we adopt an instruction level model, capture the various effects through simulation and store them in a multidimensional database that is accordingly accessed by our estimation framework for estimating the power/energy consumption of a C program.

Let us first review a basic instruction-based power model presented in [22] by Tiwari, et al. We then consider the complications of applying it to the MicroSparc core. Through this exercise, we identify the model's weakness and propose a modified model for an efficient power estimation tool. The power model from [22] is given in (1).

$$E_{prog} = \sum_i (B_i N_i) + \sum_{i,j} (O_{i,j} N_{i,j}) + \sum_k E_k \qquad (1)$$

TABLE I
Power variations between instruction sequences for different modules in the core.

Instructions	Module: Power Difference [mW]		
Add vs And	Core: 60 CC: 14 Memif :10	IU: 18 MMU :12	IU.Ex: 10 IU.Rf: 8
Add vs Sub	Core: 10 MMU: 2	IU: 1	IU.Ex: 1.5
And vs (Add.And.Sub)	Core: 40 CC: 2.3 MMU: 2	IU: 25 Memif: 2	IU.Ex: 13 IU.Rf: 8

where, E_{prog} is the total energy, B_i is the base energy cost for instruction i, N_i is the number of occurrences of instruction i, $O_{i,j}$ is the circuit state overhead for each pair of consecutive instructions (i, j), $N_{i,j}$, the number of times the pair occurs, and E_k, the energy contribution of other effects such as pipeline stalls and cache misses.

Through detailed simulation, we have made the following observations with respect to the instruction-based model in (1). (These are further elaborated in Sec. IV)

1) Irrespective of the source of the pipeline stall (memory-access, register-dependency, etc.), various modules in the core show quite similar stall power behavior, which is a direct consequence of the lack of a block-based clock gating. Therefore, differentiating between *execution cycles* and *stall cycles* would be beneficial.
2) Data variations contribute significantly to the power variation within a single instruction. Thus, a single average power number may not be sufficient to capture the power behavior.
3) Inter-instruction effects are difficult to model due to the large variations within the instruction execution context. Nonetheless, such effects are overshadowed by the effect of data variations.

Considering the above observations, we propose a refined cycle-based, instruction-level energy model. It assumes that the energy consumed in a certain cycle is induced by the instruction that resides in the execution unit at the (clock cycle) time of interest. The execution of an instruction i, can be broken down to two parts: n_{a_i} *active* cycles and n_{s_i} *stall* cycles. The stall cycles are assigned the same stall power, no matter which instruction causes the stall. Multi-cycle instructions such as *"multiply"* will have multiple *active* cycles. The average power consumed by instruction i in a cycle, P_{avg_i}, reflects the average power consumed by the micro-architecture over all cycles when that instruction is residing in the *execution stage* of the pipeline. Our model is shown in Eqn. 2.

$$E_{prog} = ((\sum_{i=1:n} P_{avg_i} n_{a_i}) + (\sum_{i=1:n} n_{s_i}) * P_{stall}) \cdot T \quad (2)$$

Here, E_{prog} is the total energy consumed by a software program, *n* is the number of instructions of the instruction trace, P_{avg_i} is the average power consumed by instruction i, P_{stall} is the average stall cycle power, and T is the period of the clock cycle. Our power database (see Section IV) not only includes the P_{avg_i} for each instruction, but also the lower and upper bounds so as to provide various options to the designer.

Use of the model in (2) requires various energy model data. Since measurement-based techniques as in [22] cannot be employed for IP cores, we employed simulation to obtain the energy model data. Each instruction is exposed to various test cases and a power value is assigned to it. In order to reduce the number of test cases, the instruction set is partitioned into *classes*, and each class is assigned a power value. We classified the instruction set into memory based and non-memory based as suggested by our initial gate-level simulations. Then, we further classified the instructions by whether or not special-purpose hardware is invoked by the respective instructions and whether integer or floating point register files were affected.

In contrast to the existing instruction-level models that employ simulation for obtaining power/energy data, our model captures the details that can actually be observed at the instruction level and abstracts away architectural features whose power/energy implications are not visible at the instruction level. Such an approach helps to retain both efficiency and accuracy of an instruction-level model when dealing with a highly optimized model of an IP core. We would like to emphasize the absence of *functional clock-gating* in many real-world, highly optimized processor cores due to which certain architectural details required by previous modeling techniques are not exposed to the power model. Our discussions in the next section will further justify the model that we proposed.

IV. Building the Architectural Power/Energy Database

The model proposed in (2) requires a power/energy database for extracting various power data required by the model (e.g., P_{avg_i} and P_{stall}). The consideration of architectural characteristics in conjunction with the optimized design representation of the MicroSPARCIIep processor core is key for constructing such a database and hence designing an accurate energy/power estimation tool. This section discusses the prominent issues to be resolved for applying the model.

A. Stall energy estimation

Pipeline stalls due to factors such as cache misses and data dependencies are unavoidable in modern microprocessors. Using measurements to capture stall energy as in (1) cannot be easily done. Detailed knowledge of the architecture and the functionalities of different modules is needed to develop good test cases for capturing these effects. This knowledge is unfortunately, not readily available for many IP cores.

In [2], idle energy of modules of a CPU is estimated as 10% of the corresponding active energy. However, the highly optimized synthesizable RTL of the MicroSparcIIep led to a different conclusion. The data presented in Tab. II show that idle power can contribute to 50% (or even higher) of the average power consumption of the respective modules.

Such observations suggest that stalls have to be modeled accurately and have to be treated in a way similar to instructions. This is especially true in control-dominated and reactive applications that tend to have a higher stall rate. The stall energy/power of MicroSparcIIep, however, has been observed to be nearly constant, independent of the kind of stalls. Therefore, we decompose the total execution cycles of an instruction into two parts, $active\_cycles$ and $stall\_cycles$ and account for these parts separately in terms of energy/power consumption. Separating the stall cycles from active cycles decouples various power/energy related effects and thus facilitates a precise and reliable power/energy estimation.

B. Variations through data dependencies

The energy/power consumed in various hardware units of the processor core depends on, among others, the data that are processed by a certain instruction. Fig. 2 shows an excerpt of

TABLE II
STALL MODE VS. ACTIVE MODE POWER CONSUMPTION

Module	P_{stall} [mW]	P_{active} [mW]	Module	P_{stall} [mW]	P_{active} [mW]
IU.Q	23.524	50	IU.D	17.805	35
IU.Pc	11.924	20	IU.Ex	8.535	30
IU.Hc	204 uW	1.75 W	IU	405.32	586.5
CC	20.296	22.5	PromIF	5.47	10.5

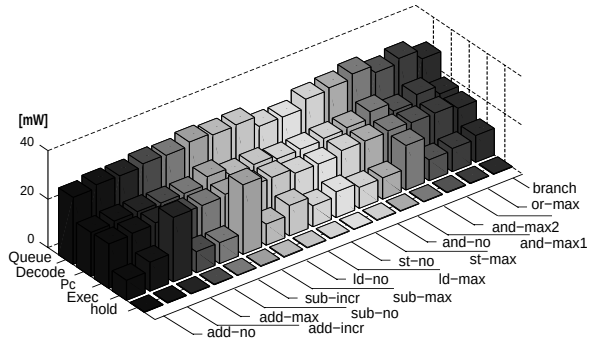

Fig. 2. Power variation within the Integer Unit IU

measurements (obtained through RT-level estimation): the z-axis shows the power consumption (in mW), the y-axis shows the break-down to the most prominent sub-units (RTL modules) of the IU while the x-axis shows diverse instructions with altered data. (*add-no*: an `add` instruction being executed with no alteration of the operands; *add-incr*: the *add* instruction incrementing an operand by 1; and *add-max*: the maximum observed power consumption of the `add` instruction by inducing maximum switching activity of the register bits holding the operands (e.g., a 0-1-0 to 1-0-1 switch of a 3-bit wide register)). The data show that some sub-blocks vary significantly in power consumption whereas others remain more or less constant when data is being altered. Overall, the variations are quite significant and cannot be ignored.

The variations in the power consumption within an instruction are important for certain investigations. For example, the maximum power values are useful for estimating the peak power, which is a key parameter in studying battery utilization efficiency. To facilitate such estimations, we maintain the maximum, the minimum, and the average power for each instruction in the power database. These data can then be used in (2) to estimate the maximum, minimum and average energy of the whole core.

C. Inter-instruction effects (IIEs)

The energy/power consumed by an instruction may vary depending on the *context* in which an instruction is executed, i.e., the instructions immediately before and after the instruction. The model in (1) captures these effects by assigning a single energy value to each pair of instructions and adds such energy values. However, our detailed simulations show that the inter-instruction effects (IIEs) are quite complex and the simple additive model in (1) is not appropriate.

Tab. III presents the power values for different instruction sequences. The first two columns are for instruction sequences containing only a single instruction, the next two columns for sequences containing two instructions, and so forth. Careful analysis of the data will show that using the formula in (1) would produce wrong results. Consider, as an example, a sequence of `and-or-sll`. The total energy for this sequence

TABLE III
POWER VARIATIONS OF INSTRUCTION SEQUENCES

Instr Seq	P_{avg} [W]	Instr Seq.	P_{avg} [W]	Instr Seq.	P_{avg} [W]
Add	1.08	Add.And	0.94	Add.and.sub	1.08
Sll	1.03	Add.Sll	0.93	And.or.sll	1.02
And	1.02	And.Or	0.93	Add.ld.and	1.09
Or	1.03	Or.Sll	0.95	Add.and.sub.or	1.09

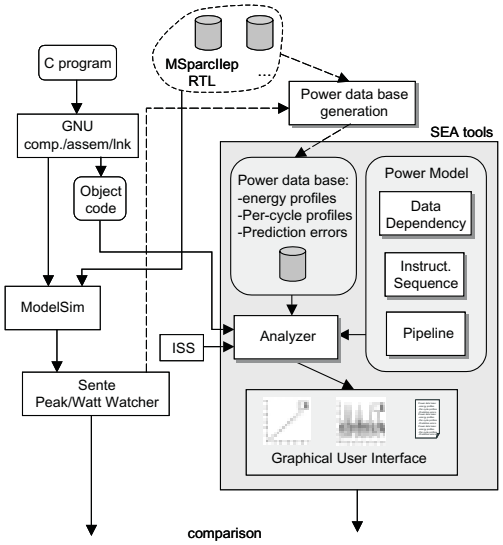

Fig. 3. Our SEA tool flow (gray underlayed) and third party tools including ModelSim and Sente

is $1.02 \times 3 \cdot T = 3.06 \cdot T$. However, if using the model in (1), the energy is computed as

$$(1.02 + 1.03 + 1.03 - 3 * 0.09) \cdot T = 2.81 \cdot T \qquad (3)$$

(The first three terms are the energy for the three instructions individually while the fourth term accounts for the IIEs calculated using the data in the 2nd and 4th columns.) Clearly, the accumulated IIEs do not correctly model the energy consumed by the instruction sequence. The data in Tab. III also show that P_{avg} has *decreased* for 2-instruction sequences and *increased or remained constant* for 3- and 4-instruction sequences. This indicates that IIEs between a pair of instructions may not necessarily be constant. Therefore, it is difficult to capture the IIE between a pair of instructions by a single value.

Our experiments also reveal that the power variations induced by the IIEs are quite small compared to the variations due to changes in data. Tab. IV illustrates the power variations due to *data* and *context*. The power variations under different data for an instruction have been shown in Cols. 2-4. The power variations due to combining that instruction with other instructions are presented in Cols. 5-7. One can readily see that the power variations due to IIEs can be ignored.

V. THE SEA POWER ANALYSIS FRAMEWORK

Our SEA framework developed on the basis of our proposed cycle-based, instruction-level power/energy model, has been depicted in Fig. 3. SEA performs

1) Accurate estimation of average as well as minimum and maximum power consumption (on a per-cycle basis or through the whole application),

TABLE IV
DATA VS IIES

	Single Instr. Seq.			Two-instr Seq.(Instr.XYZ)		
Instr.	Min [W]	Max [W]	Varn. [%]	Min [W]	Max [W]	Varn [%]
Add	0.85	1.10	29.4	0.93	1.09	17.20
Or	0.72	1.08	50.0	0.93	0.95	2.15
Sll	0.72	1.03	43.1	0.93	0.95	2.15
Ld	0.72	0.96	33.3	1.06	1.09	2.83

2) Accurate estimation of the energy consumed (minimum, maximum, actual), considering the effects of input stimuli, data dependencies, instruction dependencies and architectural characteristics, and
3) Various statistical analyses through a graphical user interface.

The input to SEA (Fig. 3) is a binary of an application program that, for example, has been written in C. The "Analyzer" is further fed with instruction traces that contain timing information. For that purpose, either an Instruction Set Simulator (ISS) or a HDL simulator can be used. The Analyzer accesses the power models capturing instruction sequences, data dependencies and pipeline effects. These models are input from the database. The database is directly generated (manually supported in a one-time effort; indicated by the dashed arrows) from the synthesizable RTL core in conjunction with a energy/power estimation tool (we used Sente). A graphical user interface allows to represent various power/energy related graphical representations along with statistics. The reference flow is Sente's Peak/Watt Watcher that is fed by the ModelSim simulator with stimuli generated by the execution of an application written in C.

VI. Experiments and Results

To verify our power/energy estimation tool suite, we have constructed the power database for the MicroSparcIIep core. We then applied our SEA framework to diverse applications that are written in C. SEA can operate in various modes and thereby aid the system designer in different design stages. The modes are described in the following.

Power/Energy Min/Max Mode: In this mode, SEA computes the minimum and maximum energy consumption during every cycle. It calculates the minimum and maximum bounds, assuming that the same program might run on different data. Therefore, the system designer gets a reliable energy estimate even when the data the application will run on is not available yet. A typical plot for this mode can be seen in Fig. 4. It shows the energy bounds *"min"* and *"max"*, the predicted energy *"pred"* and *"actual"* which is the comparison to a commercial tool (Sente [24]). As can be seen the actual energy values lie always within the min/max range. Moreover, even the predicted graph is very close to the *"actual"* derived from Sente. As we will discuss later, our tool is several orders of magnitude faster than the Sente tool.

Fig. 5 shows a plot of the power consumption with respect to clock cycles. This mode is useful when the designer wants

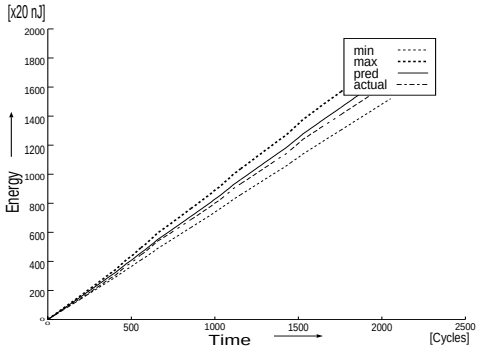

Fig. 4. Energy estimation for *"key3"* using SEA

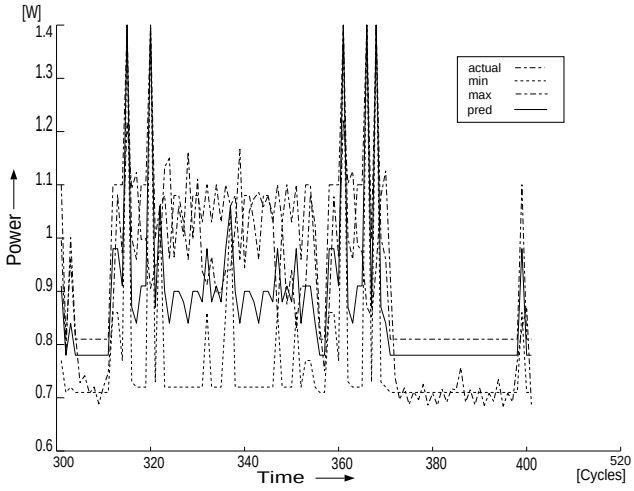

Fig. 5. Power estimation for *"key3"* in a certain time window using SEA

to identify power-critical windows in the execution of an application as it shows the power consumption cycle by cycle. It can also be used to optimize the software program in order to adapt to the battery characteristics or even to detect power-related problem areas in the hardware design. Here too, SEA is compared to Sente and the results are close but the speed of SEA is orders of magnitude faster.

Batch Mode: In this mode the SEA framework gives the energy estimation of a program without a graphical interface. This enables SEA to operate in batch mode as a tool that is being invoked by other tools.

Discussion of Results: We compare the results obtained by SEA with those obtained by the Sente tool suite [24] as the latter is a commercial tool providing the highest level of abstraction (RT level) for estimating power consumption. The comparisons are performed under following assumptions: Sente and SEA were in the mode where per-cycle energy/power consumption is estimated. Both tools assume that there is an instruction trace already available. Since instruction traces can be generated by various tools (by an ISS or by *Modelsim*, for example) at various speeds, we have not included the time for trace generation in our results, as we wanted to compare the pure efficiency of the power/energy estimation.

The results are summarized in Tab. V. The most interesting results are shown in the last three columns of the table where the computation times are compared absolutely and in terms of relative difference. Simulating a whole processor core with the stimuli data for an application turned out to be very computation intensive for the Sente tool: around 2k-3k simulated cycles (col. 8) of an application running on the synthesizable RTL of the MicroSparcIIep took between 7hrs and 15hrs for the Sente tool. Our SEA framework estimated the power using exactly the same traces in less than a second resulting in speed-ups of more than five orders of magnitude. However, we need to put this performance improvement in the context of the accuracy we achieved: The energy data estimated for various applications using our SEA framework are shown in columns 2, 3 and 4 (minimum, maximum and average). The column named *"Actual"* is the reference energy consumption achieved by Sente. The column *"PE"* shows the error in prediction. It shows that our SEA framework is in all cases within a 5% accuracy compared to Sente. However the per-cycle power accuracy, *"PCE"*, is within 12% to 17%. This is the

TABLE V
FINAL RESULTS INCLUDING ENERGY/POWER DATA AND COMPUTATION TIME

Application Program	Energy Data [uJ]				PE [%]	PCE [%]	Simulated Cycles	Sim Time		
	Min.	Max.	Avg.	Actual		Avg.		Sente	SEA	Speed-up
"Bubble Sort"	40.69	51.87	46.53	47.67	2.40	13.15	2588	15.09 H	0.10 s	$5.4x10^5$
"Heap Sort"	33.10	41.04	37.47	37.59	0.31	12.27	2188	12.76 H	0.09 s	$5.1x10^5$
"Insertion Sort"	17.74	21.47	19.87	19.08	4.15	16.63	1188	06.93 H	0.04 s	$6.2x10^5$
"Key3"	30.37	36.36	33.94	32.64	3.96	10.70	2100	12.25 H	0.09 s	$4.9x10^5$
"3d-image"	35.18	42.14	39.29	37.25	5.48	11.66	2400	14.00 H	0.10 s	$5.0x10^5$

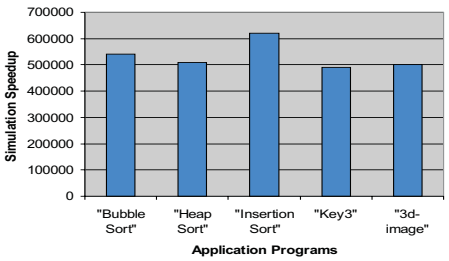

Fig. 6. Speed-up (i.e., "times faster" of our SEA framework compared to the Sente tool).

toll we have to pay for our fast database oriented estimation model. Fig. 6 summarizes the results in terms of how many times faster our SEA framework estimates energy/power compared to the Sente tool.

The comparison of our SEA framework to the Sente tool has to be set in relation, though: the aim of Sente is to estimate power of *any* RTL design whereas SEA needs a separate database for every new processor core. On the other hand, SEA's computation time is independent of the (RTL) complexity of the processor since we estimate power/energy consumption at instruction level. The higher abstraction level is eventually the reason for the high speed-up. The level of accuracy we achieve, especially for average power/energy consumption, makes SEA a reliable tool for a system designer.

VII. CONCLUSIONS

In this paper, we have introduced a cycle-based, instruction-level power/energy model and the SEA framework for fast micro-architectural energy/power estimation. As opposed to previous work in the field our estimation is for a highly optimized synthesizable RTL core which prohibits the usage of module-based estimation approaches. Instead, our models employ the notion of minimum/maximum energy/power consumption and estimate depending on how much information and certainty is available within a certain time window. Various databases have been generated to accomplish this task. As a result, our technique estimates around 5 orders of magnitude faster compared to the Sente tool suite with an accuracy that is within 5% for energy estimates and within 15% for power estimates. Though the experiments shown here are all based on the MicroSparcIIep processor core, the general techniques we used are applicable to other cores as well.

REFERENCES

[1] C. Brandolese, W. Fornaciari, F. Salice and D. Scuito, "Energy estimation for 32-bit microprocessors", *CODES* 2000, pp. 24-33.

[2] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A framework for architectural- power analysis and optimization", *ISCA* 2000, pp. 83-94.

[3] D.C. Burger, T.M. Austin and S. Bennett, "Evaluating future microprocessors:The SimpleScalar Tool Set", *TR:1308, University of Wisconsin-Madison*, July 1996.

[4] G. Cai and C.H. Lim, "Architectural-level power/performance optimization and dynamic power estimation", *Cool Chips Tutorial colocated with MICRO32*, Nov 1999.

[5] C. Chakrabarti and D. Gaitonde, "Instruction level power model of microcontrollers", *ISCAS* 1999, vol.1, pp. 76-79.

[6] N. Chang, K. Kim and H.G. Lee, "Cycle-accurate energy consumption measurement and analysis: Case study of the ARM7TDMI", *ISLPED* 2000, pp.185-190.

[7] S. Ghiyasi and D. Grunwald, "A comparison of two architectural power models", *PACS, ASPLOS-IX*, Nov 2000.

[8] C-T. Hsieh and M. Pedram, "Microprocessor power analysis by labeled simulation", *IEEE Trans. on Computer Aided Design*, Vol. 17. No. 11, Nov. 1998, pp. 1080-1089.

[9] R. Joseph and M. Martonosi, "Run-time power estimation in high-performance microprocessors", *ISLPED* 2001. pp. 135-140.

[10] P. E. Landman and J. M. Rabaey, "Activity-sensitive architectural power analysis". *IEEE Trans. on Computer-Aided Design of ICAS*, pp. 571-587, 1996.

[11] S. Lee, A. Ermedahl and S.L. Min, "An accurate instruction-level energy consumption model for embedded RISC processors", *LCTES* 2001, pp. 1-10.

[12] M. Levy, "Processors measure up to the power challenge", *Article in EDN*, July 22, 1999, www.e-insite.net/ednmag.

[13] "MicroSPARC-IIep Users manual", Sun Microsystems, April 1997.

[14] G. Qu, N. Kawabe, K. Usami and M. Potkonjak, "Function-Level power estimation methodology for microprocessors", *DAC* 2000, pp. 810-813.

[15] J. T. Russell and M. F. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors", *ICCD* 1998, pp. 328-333.

[16] A. Sama, M. Balakrishnan and JFM Theeuwen, "Speeding up power estimation of embedded software", *ISLPED* 2000, pp.191-196.

[17] M. Sami, D. Sciuto, C.Silvano and V. Zaccaria, "Instruction-level power estimation for embedded VLIW cores", *CODES* 2000, pp.34-38.

[18] D. Sarta, D. Trifone and G. Ascia, "A data dependent approach to instruction level power estimation", *VOLTA* 1999, pp.182-190.

[19] T. Simunic, L. Benini and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems", *DAC* 1999, pp.867-872.

[20] A. Sinha, and A.P. Chandrakasan, "JouleTrack - A web based tool for software energy profiling", *DAC* 2001, pp. 220-225.

[21] P. Stanley-Marbell and M.S. Hsiao,"Fast, flexible, cycle-accurate energy estimation", *ISLPED* 2001, pp. 141-146.

[22] V. Tiwari, S. Malik, A. Wolfe and M.T. Lee, "Instruction level power analysis and optimization of software", *Journal of VLSI signal processing*, 1996, pp. 1-18.

[23] W. Ye, N. Vijayakrishnan, M. Kandemir and M.J. Irwin, "The design and use of SimplePower: A cycle accurate energy estimation tool", *DAC* 2000, pp340-345.

[24] www.sequencedesign.com

[25] http://www.sun.com/microelectronics/communitysource