A hybrid approach for core-based system-level power modeling

Tony D. Givargis, Frank Vahid
Department of Computer Science and Engineering
University of California, Riverside, CA 92521
{givargis,vahid}@cs.ucr.edu

Jörg Henkel
C&C Research Laboratories, NEC USA
4 Independence Way, Princeton, NJ 08540
henkel@ccrl.nj.nec.com

Abstract

Reducing power consumption has become a key goal for system-on-a-chip (SOC) designs. Fast and accurate power estimation is needed early in the design process, since power reduction methods tend to have greater impact at higher abstraction levels. Unfortunately, current approaches to power estimation, which concentrate on register-transfer-level models or lower, are quite slow. Higher-level approaches, while faster, may suffer from inaccuracy. However, the advent of cores enables a hybrid approach, described in this paper, yielding both fast and accurate estimates from high-level models. In particular, we use power estimation data obtained from the gate-level for a core's representative input stimuli data (instructions), and we propagate this data to a higher (object-oriented) system-level model, which is parameterizable and executable. Depending on the kind of cores, various parameterizable equation or look-up table based techniques are used, resulting in self-analyzing core models. We have applied our technique to several cores of a digital camera SOC and have achieved simulation speedups of over 1000 with accuracies suitable for making reliable power-related system-level design decisions. Although we focus on power estimation, our approach can be used for estimating other metrics as well, such as performance and size.

1 Introduction

As chip capacities continue to grow, enabling systems-on-a-chip, the need for early simulation of system-level models increases. A system-level model describes desired behavior without describing detailed structural or timing information, and such models are often specified in executable languages like C++ or Java, eliminating the need for a separate simulation tool. Such models thus simulate many orders of magnitude faster than lower-level models like register-transfer or gate-level HDL (hardware description language) models, turning otherwise week or month-long simulations into just minutes or hours [13, 3]. Such models are useful for early observation of system behavior and perhaps high-level performance information, and thus designers taking a top-down approach often build and simulate such models before developing lower-level models. However, system-level models have the disadvantage of not providing sufficiently accurate information on detailed design metrics like power consumption and size, so architectural design decisions must often be postponed until later in the design process when lower-levels models are available.

We present an approach that shows that the advent of cores may result in the elimination of this disadvantage, enabling accurate design metric estimation from system-level models. A core is a reusable system-level component, such as a microprocessor, coprocessor or peripheral component, designed to become part of a SOC. Cores may be soft (synthesizable HDL), firm (HDL structure), or hard (technology-specific netlist). By many estimates, SOCs will consist mostly of cores, perhaps 90% [7], with custom synthesized logic comprising the small remaining portion. Because cores are often parameterized and their interconnecting bus structure may be flexible, core-based designs still involve a large design space and hence many architectural design decisions.

The Virtual Socket Interface Alliance (VSIA) is an industry consortium developing core-related standards. Those standards include the development of system-level models for all cores (whether soft, firm or hard). *The situation of system-level models representing already-designed components provides a unique opportunity.* In particular, we can expect the developer of a core's system-level model to have information on the core's power, performance and size metrics (unlike the case when the model was created by a designer following the top-down approach), since a low-level model does exist for cores. Since the core developer wants the core to be re–used, we can expect the developer to spend effort incorporating such metric information into the core's system–level model. This opportunity can be used to overcome the earlier-stated disadvantage of inaccurate estimates from system-level models, and thus can enable extensive design space exploration at the system level.

In this paper, we define an approach for obtaining accurate estimates from system-level core models, applicable to both soft and hard cores. We use an object-oriented system-level model. We focus on *parameterized* cores, where power, performance, and size metrics vary depending on the chosen parameter values. Some examples of parameters include buffer sizes in a UART, block sizes in a DMA, and compression algorithms in a CODEC – because cores are designed to be general, parameters are quite abundant in them. Because a parameter's setting affects other parameters' impact on design metrics (e.g., a smaller cache means less bus traffic, thus increasing the impact of bus parameters on power and performance), the usefulness of typical power tables in a core's datasheet is greatly reduced.

We first discuss related work. We then summarize the system-level modeling approach that we follow. Next, we describe how accurate power estimation can be incorporated into the modeling approach. We then describe the development process of a core model, and also describe the process whereby a designer would use such a core model. Finally, we provide experiments demonstrating the speed and accuracy of the approach, and describe future work and conclusions.

2 Related work

Work on estimating, optimizing and modeling for low power design of digital circuits has been conducted at different levels of abstraction. Here, we focus only on work that deals with *high-level* approaches, i.e., RTL–level or higher, since this is most relevant to our work.

As for the RTL–level approaches, in [14], a power optimization method is introduced that minimizes power at the architectural level (RTL–level) by using a macro model scheme. Hyper–LP [1] belongs to the same group. It is a framework that optimizes power consumption of data–dominated systems through architectural transformations. Recently, a behavioral–level power optimization method has been introduced in [10]. So called *common cases* are exploited in order to synthesize a compact data–path such that power reductions of up to around 92% become possible.

As for high-level modeling, much recent research [8] [12] [18] has emphasized object–oriented models, mainly based on Java, that enable hardware description at a behavioral abstraction level. In addition to providing models for capture and simulation of digital systems, these contributions provide solutions to problems such as: ease of conversion from high-level description to hardware description for synthesis, modeling of reactive systems, and deterministic modeling of digital systems with bounded resource allocation. To the best of our knowledge, these contributions do not provide means for power and area and performance estimation at the system level.

Vahid and Givargis [16] have proposed specification of more abstract components, i.e., SOC cores, that communicate via method-calling or message-passing. Their high-level model and functional communication allows for exploration of SOC busses for low

power [5]. Other work by Givargis et al. [6] has extended this exploration with power and performance results obtained from CPU, interface and cache simulations.

Our work capitalizes on cores to obtain sufficiently accurate estimates using system-level models. The aim was to combine the accuracy of lower–level power estimation approaches with the convenient facility of a parameterizable and fast system–level approach. Therefore, we deployed power data collected from gate-level simulations to estimate the power and performance of a core using object–oriented models. It is important to notice that the gate–level simulations have to be done only *once* for a characteristic set of *instructions*[1] of a core and that this data can be used at the OO–level using a facility of a parameterizable look–up or equation mechanism. As a result, our approach is orders of magnitudes faster than approaches proposed so far, but at a accuracy that is relatively close to low–level obtained power estimations.

3 State–of–the–art in power estimation

Current methods for power estimation/optimization deployed during the design phase of SOC designs can be summarized as follows. (1) Gate-level methods are deployed after high-level decisions like defining a hardware/software partition or an appropriate hardware architecture have already been made, i.e. after RTL synthesis and logic synthesis have been performed. Gate-level power estimation is cycle-accurate and therefore may require prohibitive simulation times when applied to entire SOCs.
(2) RTL-level power estimation tools represent the current state–of–the–art in SOC design for low power. Those tools typically perform an RTL–level simulation that is attached to a technology library of sub–blocks [2] containing toggle data and effective capacitances in order to calculate the consumed energy clock cycle by clock cycle. The accuracy of those methods can be quite good [14] though simulation times are still quite lengthy for entire SOCs. Furthermore, the approach is applicable for hardware parts of a system only, thus not supporting comprehensive system-level design space explorations. (3) Less sophisticated approaches for system-level power estimation use an average power consumption of a core (as available in a data book of a hard core, for example) and sum up the power numbers of all cores of a SOC to obtain a rough estimate. Such average-based approaches have two main drawbacks that may make them very inaccurate. First, they do not consider the particular system application's functionality, and second, they do not consider the impact of one core's parameters on other cores (e.g., the impact of cache size on bus traffic).

As opposed to the existing methods desribed above, our novel hybrid approach uses an executable specification in conjunction with (one-time) obtained low-level power data, and has the following unique features:

1. The granularity of power estimation approach is based on an entire core (i.e. block) and not just a sub–block, thus supporting core-based design techniques.
2. The abstraction level where the complete system simulation takes place is the system-level. The most important advantages here are the speed as well as the simplicity to modify the functionality for the purpose of design space explorations.
3. Our approach is not limited to hardware parts only. Instead, hardware and software are treated in the same way, thus accounting for hardware/software interdependencies. In previous work [15] the importance of software in terms of power consumption has been demonstrated.

4 System-level modeling: an overview

We briefly describe two system-level modeling approaches, namely, method-calling objects and message-passing processes [16]. The major distinction between these two approaches is the way in which cores communicate and synchronize with each other, as described next.

[1] We use a relaxed definition of the term *instruction*: whereas in case of a processor core an instruction might denote operations like *add* or *shift*, for example, an instruction in the context of a UART might denote a writing to or reading from its register.

[2] we use the term *sub–block* for an RTL component like multiplexer, register, adder, shifter, etc. whereas a *block* or *core* is a larger system part that facilitates a comprehensive functionality like an MPEG encoder. *Blocks* are composed of *sub–blocks*. See also [7].

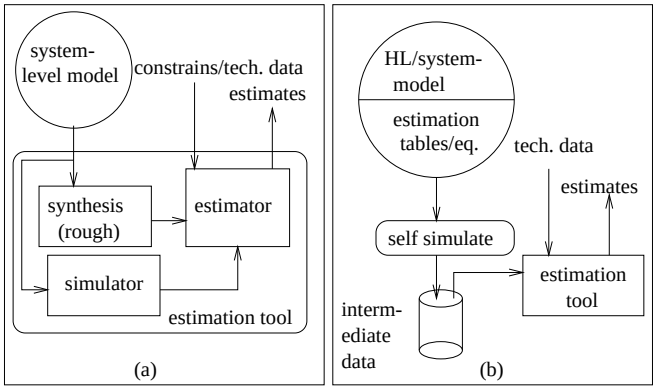

Figure 1: (a) Previous and (b) our system-level estimation approaches.

In method-calling objects, each core is represented as an object, which communicate with each other via method calls. Parameter passing is used to exchange data between cores. The objects may be active objects, meaning each has its own thread of control. Calling a method may be used to transfer control from one core to another, and provides a means for synchronization. The high-level implementation of the core is entirely divided up among the methods of the high-level object. Method-calling objects provides for a very high-level encapsulation of cores with a functional interface allowing for early exploration of system-on-chip busses and encoding schemes. Method-calling objects can be converted to message-passing processes by converting all objects to active ones, and replacing method calls by message passes.

In message-passing processes, each core is represented as a process (perhaps using active objects). Communication is accomplished by passing messages from one object to another. Sending of a message blocks the sender until the reception of the message by the receiving object, hence providing synchronization via message passing. This model is based on Hoare's model of communicating sequential processes. Here, the high-level implementation of the core resides in an infinitely running loop, e.g., the "run" method. A model based on message passing objects can be automatically refined into a lower level implementation [16]. For example, each object's "run" method is converted into a process in a low-level HDL. Likewise, the "sends" and "receives" are converted to bus structures, i.e., ports and signals.

The former approach is more abstract and hence may be easier to work with and may also execute faster. The latter approach involves more communication detail and so may be slower but is closer to a hardware implementation. Both approaches, however, are still extremely fast. Our estimation approach can be incorporated into either of these modeling approaches.

5 Estimation

Previous (non-core-based) system-level estimation approaches have been developed to work with designs that are intended to be fully described, and then synthesized. Such estimation tools require a system-level model of a design as input. This input is subsequently synthesized, in a rough manner, to gather low-level information about the design. In addition, the designer is required to provide constraints and technology specific information, e.g., clock frequency, to the estimation tool. This kind of system-level estimation is depicted in Figure 1(a). Constraints and technology information, combined with simulation of the model and information obtained from synthesis, are then used to provide power, area, and performance estimates.

Core-based design provides us with a unique opportunity to develop estimation tools that can estimate power, area and performance with better accuracy. This is because for pre-designed cores, a low-level model is already available, e.g., RT-synthesizeable HDL for soft cores, netlist for firm cores and layout for hard cores. An estimation tool can use this low-level knowledge to better estimate system metrics. As an example, consider a UART (universal asynchronous reciever/transmitter) core implemented by a core-supplier in synthesizable HDL, having its buffer size as a parameter. By per-

```
class UART {
   unsigned toggle_count = 0;
   public Reset() {
      data_txbuf = 0, data_rxbuf = 0;
      toggle_count += TOGGLE_TABLE[RESET];
   }
   public EnableTx() {
      for(int i=0; i<8; i++)
         s_out = (data_txbuf >> i) & 0x01;
      toggle_count += TOGGLE_TABLE[ENABLETX];
   }
   public WriteTxBuf(unsigned char x) {
      data_txbuf = x;
      toggle_count += TOGGLE_TABLE[WRITEBUF];
   }
   public ReadRxBuf(unsigned & x) {
      // implementation...
      toggle_count += TOGGLE_TABLE[READBUF];
    }
};
```

Figure 2: UART model using method-calling objects

forming gate-level simulation of UARTs with different buffer sizes, one can obtain area and toggle switch information for different parameter settings; we performed such simulations for a particular UART and provided the toggle and size data in Table 1. Note that this is a simple example, and in more complex, multiple-parameter, cores, an equation or even a function may be necessary rather than simply a table. With such data (or equations or functions) available for all cores, the area (hardware effort) of a core-based system for given settings of parameter values can be estimated by summing the area of the individual cores for those settings. Likewise, after simulation of a core-based design at system-level, one can use low-level toggle data to accumulate total toggle counts and estimate power consumption of the design for a given technology.

Figure 1(b) depicts an approach for a system-level model of a core based design. In this approach, the high-level model will contain lookup–tables, equations, or functions, obtained from low–level simulations. Thus, after simulation, intermediate data is collected, e.g., toggle–count, to be combined with technology data in subsequent estimation. We will next outline the application of this approach using method-calling objects.

Buffer-Size(byte)	Area(transistors)	Toggle-count
2	4552	203
4	7360	232
8	1576	238
16	22600	249

Table 1: Data obtained from gate-level simulation of the UART core. Toggle-count is the average transistor state-transitions when sending a random byte of data.

In method-calling objects, each method is augmented with a section of code that accumulates data during simulation. An example of this is given in Figure 2. Here, we assume that TOGGLE_TABLE is obtained from gate-level simulation. Hence, each time the core is "reset" or "enabled", the appropriate toggle count is added to the total toggle-count. At the end of the simulation, toggle-count is used to estimate the power usage of the UART core.

6 Model developement

In this section we outline steps necessary to create a self simulating and analyzing model of a core as illustrated in Figure 3(a). In brief, these steps involve design and capture of a core in some hardware description language (HDL), simulation at gate-level, and design of the object oriented model using data acquired from the gate-level, or lower-levels for hard-cores, simulations.

A core developer will start by designing and capturing the functionality of the core, say a UART, using some HDL, say VHDL. Then, the core developer will identify a set of instructions describing the functionality of this core. In the case of the UART core, the

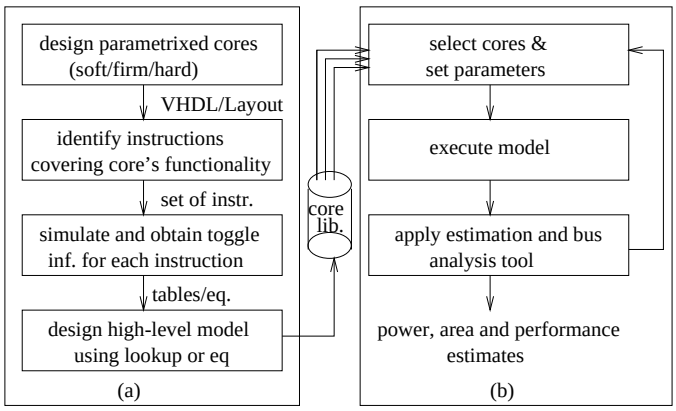

Figure 3: Design (left) and use (right) flow for modeling/using a cores for high-level simulation

following instructions can be identified: "write-reg" (writing a byte to the transmit hold register), "enable-tx" (enabling of the transmitter to serially transmit the content of the transmit hold register), and others such as "reset", etc..

The next step is to acquire gate-level simulation data to be used in high-level estimation. Given an HDL description of a core and a set of instruction, the core developer will create specialized testbench programs for each instruction in order to measure design metrics such as performance or power. For example, given the UART core, we create a testbench that performs a write to the transmit hold registers, i.e., executes the "write-reg" instruction. Then, we synthesize the HDL description of the UART and simulate the UART using the testbench, accumulating toggle switch information. We repeat this process for all other instructions and create a table of toggle switches per instructions. If a core is parameterizable, multiple tables for different parameter values must be obtained, or estimated via equations.

The next step is the design of the high-level model of the core. The core developer can now model the core, using one of the approaches given in the system-level modeling section, in any object oriented language. The object representing the core will provide methods corresponding to the instructions identified in the earlier step.

Figure 4 shows a simplified flow–diagram of the functionality of a generic object–oriented self–analyzing core model: If the core is *Busy*, i.e., another core claimed this core for a particular time in order to accomplish its own task, this core cannot be claimed by any other core. The core model can either be called by another core model (*Core Call*) or it is called by a control object that takes care of the interplay of all cores which together model the whole SOC behavior. In the first case the following actions are performed: The according core is put into the busy state, the energy counter is initialized and the clock counter is set to the number of clock cycles it takes to execute the specified action , *i.e., instruction,* assuming that no delay (e.g. no waiting for other resources like a bus) occurs. Then, it is asked whether the resources that will be used by this core are available or not. If *yes* then those resources are claimed for the time they are needed (and at this time they are unavailable to other cores). If *no* then the execution of this core is delayed costing some energy for the idle state (retrieved from a look–up table). The two leftmost branches of the flowdiagram belong to the case the OO core model is not called by another core model but by an object instead that initializes the according actions of this core after each clock cycle. Therefore, first the clock cycle counter is decremented, then, when it is counted down to zero, the actual action (*instruction*) is executed.[3]

For our UART example, the object will have methods (implementing the *instructions*) "write-reg" and "enable-tx" and a variable called toggle-count, initialized to zero by the constructor, say. When "write-reg" or "enable-tx" are called, toggle-count will be

[3] Assigning the actual action to only one clock cycle is a strong simplification and might possibly cause conflicts. This is the price we have to pay for using the high abstraction level that does not allow for a cycle accurate execution.

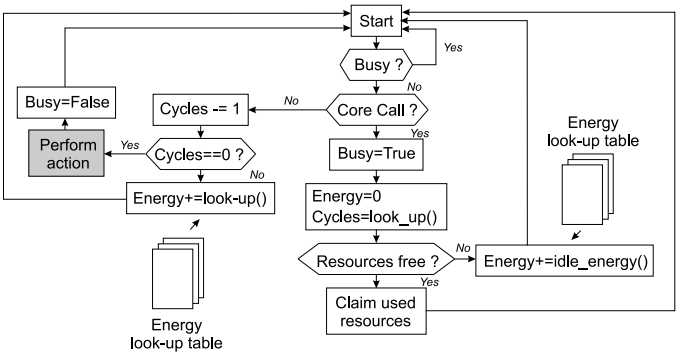

Figure 4: Simplified flow–diagram of the functionality of a generic object–oriented self–analyzing core model

incremented by looking up the toggle switch amount for these instruction in the tables.

7 Model use

In the previous section, we described the steps a core-designer could use to build a core's system-level model in a manner supporting estimation. In this section, we outline steps necessary for a core *user* to use such a core model to build an SOC and perform estimations, summarized in Figure 3(b). In brief, these steps involve creating a high-level model of the system, simulating it to obtain simulation data, and analyzing the results to obtain power and other estimates.

The user of cores will select a set of objects representing the cores of the system under design. These cores will be integrated together using one of the methods described in the system-level modeling section. Then, this model of the system will be simulated. At the end of the simulation, each core in the model will output power and performance data, say toggle count. Multiple simulation runs can be carried out to obtain power and performance data for different core parameter values and configurations.

The data outputted by the cores will be subsequently used in an analysis tool. At the minimum, the analysis tool will apply proper technology specific metrics, e.g., capacitance, clock frequency, etc., to the data obtained from simulation in order to produce power and performance estimates. For example, the analysis tool will multiply the square of the supply voltage and the average cell capacitance with the toggle switch data to obtain power estimates. The analysis tool may examine other simulation data, such as i/o frequency to and from different cores, to also perform estimates of power consumed by on-chip busses [5].

8 Experiments

To illustrate the concepts presented in this paper, we modeled a Digital-Camera using method calling objects as well as using RT-synthesizable VHDL. Figure 5 illustrates the four main cores, CCD-PP, MIPS, CODEC and UART that capture, process and output digitized pictures. We will next detail the Digital-Camera's functions, the object-oriented model of it, and the simulation, analysis and estimation of power and performance results.

The heart of the Digital-Camera is a simplified MIPS processor and on-chip cache bounded together via a high-speed bus and surrounded by a number of peripheral devices, i.e., cores, communicating over a peripheral bus, designed for layout on a single chip. The application running on the MIPS issues a "capture" command to the CCD-PP (charged coupled device - pre-processor) core which in turn uses a CCD to capture and process a single frame and stores it into internal memory. The application then clocks out the image, one pixel at a time, and stores it into memory to be encoded/decoded by the CODEC, displayed and serially transmitted via a UART core. The UART core can be synthesized to have an internal buffer size of 2, 4, 8, or 16. The CCD-PP can be synthesized for use with 8x8, 16x16 and 32x32 CCDs, the CODEC can be synthesized with one of four different compression algorithms.

The high-level Digital-Camera model is composed of four C++ objects representing the simplified MIPS, CCD-PP, UART and CODEC, totalling about 600 lines of code. Figure 6 depicts the

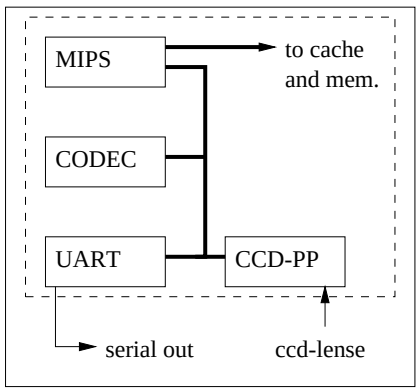

Figure 5: Block diagram of the Digital-Camera example

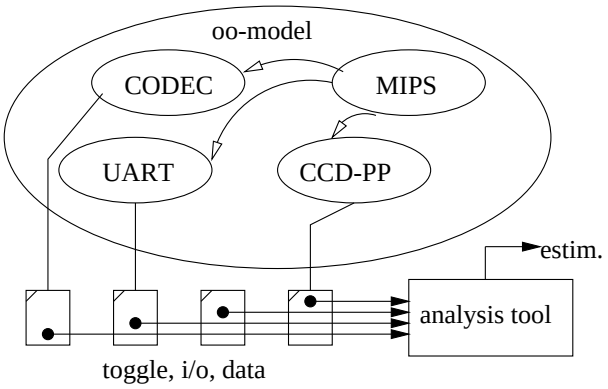

Figure 6: Object-oriented model of the Digital-Camera

objects, and their relationship to each other. Each one of these objects provides member functions, a.k.a., methods, that functionally describe the core it represents. The CCD-PP, UART and CODEC can be instantiated with different parameter values. This allows for estimates for different core parameters configurations. We defined three instructions that abstractly model the CCD-PP, namely, "reset", "capture" and "read pixel". The corresponding methods, when invoked, add to a member variable (initialized to zero on start of the simulation) some number of toggle switches obtained from gate-level simulation. Similarly, the UART is broken into four instructions, "reset", "enable", "disable" and "write buffer". These toggle switch counters are outputted by each object at the end of the simulation, along with i/o and timing information from the MIPS object to be used in a subsequent power/performance and bus analysis tool as described next.

The analysis tool reads the toggle switch data from simulation and computes power using technology specific metrics such as wire/gate capacitance [2] [17], and supply voltage. In addition, i/o (frequency of method calls) is used to explore peripheral bus configurations such as width and encoding for low power and acceptable performance [5].

The low-level Digital-Camera model, used to validate the accuracy of our system-level approach, consists of Synopsys-synthesizable RT models of the four cores, totalling about 4500 lines of code, written as part of UCR's Dalton project. To obtain actual power values to compare with, we synthesized the models to gates and then used the Synopsys power estimation tool to get gate-level accurate power and performance results.

We simulated the low-level and the high-level models of the Digital-Camera for 8 different sets of core parameters. Each simulation was long enough to capture, process and serially transmit a single frame, i.e., digital image. Table 2 summarizes our results. The number in the UART column is the buffer size (in this case, the same values for all sets), that under CCD-PP is the image size, that under the CODEC is the compression algorithm. The next column gives the peripheral data bus width. The next two columns give the

UART	CCD-PP	CODEC	BUS	System-level time	Gate-level time	System-level power	Gate-level power	Power error
2	16x16	1	32	0.07	85	0.02137	0.02560	16%
2	16x16	4	32	0.06	58	0.01511	0.01776	13%
2	8x8	1	32	0.01	20	0.00912	0.00692	31%
2	8x8	4	32	0.01	4	0.00567	0.00455	25%
2	16x16	1	8	0.09	147	0.01953	0.02198	11%
2	16x16	4	8	0.06	90	0.01189	0.01349	12%
2	8x8	1	8	0.01	30	0.00684	0.00553	24%
2	8x8	4	8	0.01	7	0.00443	0.00342	30%

Table 2: Time and power comparison between system-level and gate-level time (minutes) and power (micro-Joules).

CPU-time it took to run the simulations for the system and gate-level models. On the average, the system-level model simulated thousands of times faster than the gate-level model. The next two columns give estimated system-level and actual gate-level power. Estimated power was between 11% and 31% accurate. More importantly, the magnitude relations among the estimated power values match the magnitude relations among the gate-level values, meaning that architectural decisions can be made correctly from the estimated values. For example, the last row represents the lowest power solution, as indicated by both the estimated and actual power numbers, corresponding to a small CCD and a simple compression scheme.

9 Future work

One limitation of our approach is that simulation must be performed for every configuration of core parameters that we wish to consider. While such simulation is many orders of magnitude faster than gate-level simulations, it still requires several seconds and hence prohibits exploration of hundreds or even millions of configurations. Thus, further techniques should be developed that allows one to simulate the system just once, and then rapidly explore different core parameter values (as has already been done for on-chip bus exploration [5]). Work must also be performed to perform estimation for microprocessors and memory cores, which are highly-specialized components requiring specialized estimation techniques. Another avenue of future work may investigate the impact of particular *sequences* of core instructions, whereas in this paper we treated each instruction independently.

10 Conclusion

Power estimation from high-level models early in the system design process previously suffered from much inaccuracy, while very-accurate estimation from lower-level models (e.g., RTL–level or gate–level) suffer from unacceptably long computing times. However, the advent of cores means that accurate low-level power information can now be incorporated into high-level models. We defined such a hybrid approach and conducted experiments, with an emphasis on parameterized cores, resulting in extremely fast system simulations on the order of milliseconds, as well as sufficiently accurate estimations. The approach is applicable to other design metrics also, such as size and performance. The result is that designers of core-based systems using this approach can make power–related architectural design and parameter selection decisions early in the design process, where impact is large, under the guidance of fast and accurate estimations.

11 Acknowledgement

A Design Automation Conference Graduate Scholarship and a NSF grant supported this research. We are grateful for their support.

References

[1] A.Chandrakasan, M.Potkonjak, J.Rabaey, R.Brodersen, *Hyper-LP: A System for Power Minimization using Arhcitectural Transformations*, IEEE Proc. of Int'l Conf. on Computer–Aided Design (ICCAD92), pp. 300–303, 1992.

[2] J.Chern, J.Huang, L.Arledge, P.Li, P.Yang, *Multilevel Metal Capacitance Models for CAD Design synthesis Systems*, IEEE Electron Device Letters, vol. 13, no. 1, pp.32-34, January 1992.

[3] A.Evans et al., *Functional Verification of Large ASICs*, Design Automation Conference, 1998.

[4] W.Fornaciari, D.Sciuto, C.Silvano, *Power Estimation for Architectural Explorations of HW/SW Communication on System–Level Buses*, To be published at HW/SW Codesign Workshop, Rome, May 1999.

[5] T.Givargis, F.Vahid, *Interface Exploration for Reduced Power in Core-Based Systems*, International Symposium on System Synthesis, December 1998.

[6] T.Givargis, J.Henkel, F.Vahid, *Interface and Cache Power Exploration for Core-Based Embedded System Design*, Submitted to International Conference on Computer Aided Design, November 1999.

[7] R.Gupta and Y. Zorian, *Introducing Core-Based System Design*, IEEE Design and Test, Vol. 14, No. 4, Oct-Dec 1997, pp. 15-25.

[8] T.Kuhn, W.Rosenstiel, U.Kebschull, *Object Oriented Hardware Modeling and Simulation Based on Java*, International Workshop on IP Based Synthesis and System Design, Grenoble, France, 1998.

[9] S.Kumar, J.Aylor, B.Johnson, W.Wulf, *Object-Oriented Techniques in Hardware Design*, IEEE Computer, vol. 27, pp. 64-70, June 1994.

[10] G.Lakshminarayana, A.Raghunathan, K.S.Khouri, N.K.Jha, *Common Case Computation: A High-Level Power-Optimizing Technique*, IEEE Proc. of Design Automation Conference (DAC99), June 1999.

[11] F.Mallet, F.Boeri, and J.F.Duboc, *Hardware Architecture Modeling Using an Object-Oriented Method*, Proceedings of the 24th EUROMICRO Conference, August 1998.

[12] C.Passerone, R.Passerone, C.Sansoe, J.Martin, A.Sangiovanni-Vincentelli, R.McGeer, *Modeling Reactive Systems in Java*, Proceedings of the Sixth International Workshop on Hardware/Software Codesign, March 1998.

[13] B.Payne, *Rapid Silicon Prototyping: Paradigm for Custom System-on-a-Chip Design*, http://www.vlsi.com/velocity, 1998.

[14] A.Raghunathan, S.Dey, N.K.Jha, *Glitch analysis, and reduction in register–transfer–level power optimization*, IEEE Proc. of Design Automation Conference (DAC96), pp.331–336, 1996.

[15] V.Tiwari, *Logic and system design for low power consumption*, PhD thesis, Princeton University, Nov. 1996.

[16] F.Vahid, T.Givargis, *Incorporating Cores into System-Level Specification*, International Symposium on System Synthesis, December 1998.

[17] N.H.E.Weste, K.Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, 1998.

[18] J.S.Young, J.MacDonald, M.Shilman, A.Tabbara, P.Hilfinger, A.R.Newton *Design and Specification of Embedded Systems in Java Using Successive, Formal Refinement*, Proceedings of the Design and Automation Conference, June 1998.

[19] *National Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 1997.