Instruction-based System-level Power Evaluation of System-on-a-chip Peripheral Cores

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

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

Abstract

Various system-level core-based power evaluation approaches for core types like microprocessors, caches, main memories, and buses, have been proposed in the past. Approaches for other types of components have been based either on the gate-level, register-transfer level, or behavioral-level. We propose a new technique, suitable for a variety of cores like peripheral cores, that is the first to combine gate-level power data with a system-level simulation model written in C++ or Java. For that purpose, we investigated peripheral cores and decomposed their functionality into so-called instructions. Our technique addresses a core-based system design paradigm. We show that our technique is sufficiently accurate for making power-related system-level design decisions, and that its computation time is orders of magnitude smaller than lower-level simulation approaches.

Keywords

System-on-a-chip, low-power design, intellectual property, caches, cores, estimation, silicon platforms, system parameters.

1. Introduction

As mobile computing devices have gained a major market share in the computing sector and their functionality (i.e., complexity) has increased rapidly, minimizing power consumption has become one of the most important design goals. Furthermore, the short life cycles of consumer products in conjunction with increasing product complexity has led to a core-based design paradigm. 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, DMA controller, UART, bus interface, or CODEC, residing on a system-on-a-chip (SOC) with typically tens of other cores. A core, also known as Intellectual Property or IP, may come in different forms: a soft core comes as a synthesizable model in hardware-description language (HDL), a firm core as a structural model in an HDL, and a hard core as a technology-specific layout. Core providers typically parameterize their cores, whether soft, hard, or firm in form, in order to increase the core's applicability in different applications and to ease its re-usability. Example parameters include bit-widths and buffer-sizes. Soft and firm cores may be parameterized via HDL generics, while hard cores via layout generators. For example, a UART's buffer size may be varied from 1 to 16 bytes, trading off size and power for performance. An important system-on-a-chip design task is thus the configuring of all cores' parameters, such that the configuration is tuned for the application (i.e., the software running on the SOC's microprocessor core) and for the power, size and performance constraints of the SOC.

Since the whole SOC has to be optimized in terms of power and the according parameters are interdependent and large in number, fast and accurate evaluation and optimization tools are needed.

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 power tables provided in core databooks, representing average power data, may yield inaccurate power numbers for a particular application and configuration, even when extended to account for a subset of common configurations.

Therefore, numerous researchers have proposed techniques for rapid, system-level power evaluation of certain types of cores, including microprocessor, cache, memory and bus cores. In our efforts to develop a system-level power evaluation environment for parameterized SOC's, we found however that no techniques existed to evaluate general cores as fast and accurately as a combined gate-level/system-level (i.e., executable specification) could provide. Our work uses this approach and applies it to peripheral cores, i.e., those single-purpose processing cores that typically surround a microprocessor cores, such as DMA controllers, UARTs, bus interfaces and CODECs, to explore this promising technology.

The remainder of the paper is organized as follows. In Section 2, we describe related work. In Section 3, we present our power evaluation technique. In Section 4, we give our experimental results. In section 5, we conclude.

2. Previous work

Previous power evaluation work has been done at various abstraction levels. Circuit-level approaches simulate the circuit at the transistor level while monitoring supply current [1][2]. Logic-level, or gate-level, approaches simulate a gate-level design, and calculate power by considering switching activity of nodes in the design [3][4], executing orders of magnitude faster than circuit-level approaches at the expense of some accuracy. Even so, logic-level approaches are many orders of magnitude too slow to be used in SOC configuration exploration, requiring days to obtain data for even one configuration.

Figure 1: Constructing VHDL models for extraction of gate-level power estimates.

```
Note: k and s chosen to maintain error within range
for each parameter p
  for each power-mode m
    for each instruction i
      if i is independent
        create a test-bench that simulates i(random),
        in mode m, configured as p, k times
      else if( i is dependent on statistical char. )
        for each statistical characterization s
          create a test-bench that simulates i(s),
          in mode m, configured as p, k times
      else if( i is dependent directly on input data)
        for each cluster of input pattern d
          create a test-bench that simulates i(d),
          in mode m, configured as p, k times
```

Figure 2: Augmenting the functional implementation of an instruction for power evaluation.

```
// functional implementation before here
power-mode = NextPowerMode(power-mode, current-inst);
p = this core's current parameter values;
m = power-mode;
i = current-instruction's identification number;
d = data passed into the current-instruction;
if( i is independent of its data ) {
   total-power += PowerLookupTable[p][m][i];
}
else if( i is statistically dependent on its data ) {
  total-power +=
  PowerLookupTable[p][m][i][[GetStats(d)];
}
else if( i is dependent on it data ) {
   total-power += PowerLookupTable[p][m][i][d];
}
```

RTL (register-transfer level) power evaluation operates at an even higher-level of abstraction, modeling power consumption of more abstract circuit components, such as adders and multipliers. Simulation is performed at the RT-level and power is obtained by using these power models, also known as macro-models. The approach taken here can be divided into two categories, macro-modeling using table-lookup techniques and analytical models. Power modeling and evaluation in 0 was among the first to show the feasibility of RTL-based approaches that showed a very good accuracy and a much higher speed than gate-level approaches. Using table-lookups, each component is 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. While RTL power evaluation is shown to be accurate to within 5% of actual power consumption 0, it too suffers from simulation times too slow for extensive system-level exploration. Furthermore, just synthesizing an RTL design for a given configuration can take hours, independent of simulation.

Previous behavioral-level approaches seek to estimate power of a behavioral HDL description before a synthesized design is obtained. An abstract notion of physical capacitance and switching activity is used. Switching is estimated using entropy from circuit input to circuit output by quadratic or exponential degradation [11][12]. While such behavioral approaches can provide fast evaluation of power for custom designs, they will not be nearly as accurate for cores as approaches that take advantage of the fact that cores are pre-designed.

Work has been done to evaluate power consumption of microprocessor cores. One approach, instruction-level power modeling, is proposed by [13]. 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 [14] by deriving a shorter program trace that results in equal power dissipation when compared to the original trace. In [15], a mathematical generic power model for 32-bit microprocessors is proposed. The approach classifies the instruction set into classes like branches etc. The model has been applied to various 32-bit processors. Other researchers have focused on fast system-level models for cache, memory and bus power consumption [16][17][18], consisting mostly of equations that compute power consumption as a function of usage/traffic and core parameters. Further approaches aim at estimating the power consumption of CPU, cache, memory and bus together. In [19], a cycle-accurate power simulation tool is introduced for an embedded system using a StrongARM architecture. The reported results are accurate within 5% compared to measurements conducted on a real existing hardware board. A trace-based approach deploying a mix of analytical models (for instruction cache, data cache, and main memory) and instruction set simulators is introduced in [20]. Their Avalanche tool can either conduct a fast design space exploration or it can be used to determine the optimum system parameters.

3. Power-evaluation for peripheral cores

3.1 Overview

We examined a variety of peripheral cores, and found that they all could be viewed as executing a sequence of what we call "instructions", using this term in a relaxed manner. Typically,

Table 1: UART's power mode transition function.

Current Instruction	Current Mode	Next Mode
Reset	Idle	Idle
	Tx_enabled	Idle
	Rx_enabled	Idle
	Tx_rx_enabled	Idle
Enable_tx	Idle	Tx_enabled
	Tx_enabled	Tx_enabled
	Rx_enabled	Tx_rx_enabled
	Tx_rx_enabled	Tx_rx_enabled
Enable_rx	Idle	Rx_enabled
	Tx_enabled	Tx_rx_enabled
	Rx_enabled	Rx_enabled
	Tx_rx_enabled	Tx_rx_enabled
Send/Receive	Idle	Idle
	Tx_enabled	Tx_enabled
	Rx_enabled	Rx_enabled
	Tx_rx_enabled	Tx_rx_enabled

an instruction represents an atomic action available to the programmer of a microprocessor. But we use "instruction" more generally as an action that collectively with other actions describes the range of possible behaviors of a core. Furthermore, an "instruction" can be better used for power evaluation since – compared to a classical instruction – our notation of instruction can denote a smaller or a larger piece of functionality depending on the power characteristics. Thus, we have extended the instruction-level power modeling approach, previously used for microprocessor cores, to peripheral cores. In developing the approach, we noted that cores typically already come with system-level functional models, written in a language like C, 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 SOC core, say a UART, we are to devise a high-level executable model, say in C++, of that core that can output power dissipation during a system-level simulation. Furthermore, this model must be sensitive to changes in the various parameters of that particular core. Although we define our modeling and evaluation approach for a single parameterized core, the approach can be applied to each peripheral core in an SOC to obtain total power.

Our approach can be broken into a number of steps as follows. The core provider must select a set of appropriate instructions, perform gate-level power analysis to construct power lookup tables for each instruction, and create a system-level core model that utilizes the lookup-tables for power evaluation. The core user connects the system-level core models, executes the whole system (which is possible since the system-level model represents an executable specification), and thus obtains power data after a system execution/simulation. We now describe each step in more detail, using a UART core as an example.

3.2 Peripheral instructions

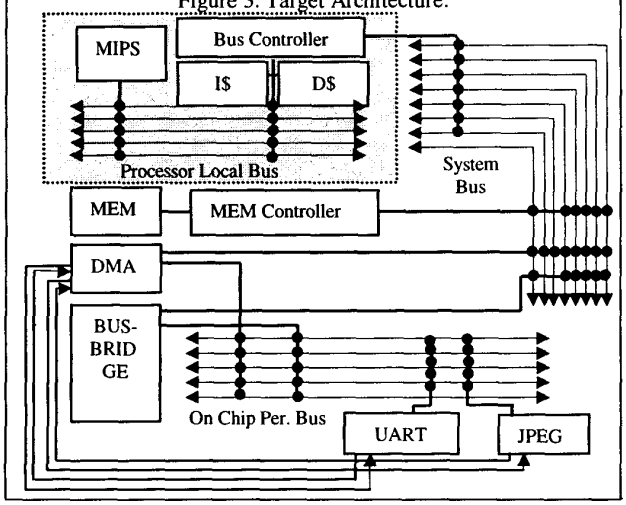
The core provider must first break the core's functionality into a set of instructions. Given an RTL model of a core called *C*, we first determine the system-level instructions $i_1, i_2, i_3, \ldots i_n$, of *C*. These instructions must have the property that they collectively cover the entire functionality of *C* and that no two instructions cover the same function of *C*, i.e., $i_1 \cup i_2 \cup i_3 \cup \ldots \cup i_n$ = *Functionality of C* & $i_j \cap i_k = \varnothing$, *for* $1 < j, k \leq n$. As with the instructions of an instruction-set processor, each instruction i_j operates on some input data and produces some output data. In our UART example, we selected the following instructions: *Reset*, *Enable_tx*, *Enable_rx*, *Send*, and *Receive*.

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 databooks, a core designer's knowledge, experimental results or statistical analysis. For our 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 power regardless of the data being sent; likewise for the *Receive* instruction.

Note that this is just a simple example. In general, there is a trade-off in choosing the right instructions for power evaluation: if the granularity is small, we will have many instructions leading to a longer simulation time. On the other hand, a small granularity tends to produce more accurate results since more subtleties are taken care of. When we have coarse-grained instructions, then the total number of instructions is smaller, leading to faster simulation. But coarser instructions might not be able to take into consideration subtle effects, and as such may lead to less accurate results.

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 inter-instruction power dependencies (e.g., a load following a store may consume more power than a load following an add). To account for a mode, the core provider must determine the set of modes, $m_1, m_2, m_3, \ldots m_k$ in *C*, referred to as *power-modes*, that cause *C* to consume significantly more or less power per each execution of its instructions, $i_1, i_2, i_3, \ldots i_n$. In our UART example, we found four power modes: *Idle*, *Tx_enabled*, *Rx_enabled*, *Tx_rx_enabled*. Given these modes, we define a power-mode transition function, that gives the next power-mode given the current power-mode and the most recently executed instruction of *C*. For the UART example, the power-mode transition function is given in Table 1.

Figure 3: Target Architecture.



3.3 Gate-level power evaluation

The second task consists of using gate-level simulation to obtain per-instruction power data for the lookup-tables. Given an RTL model of a core called *C*, its instructions $i_1, i_2, i_3, \ldots i_n$, and its modes $m_1, m_2, m_3, \ldots m_k$, we follow the procedure outlined in Figure 1. This procedure gives a methodical way of creating a set of testbench models that, when simulated at gate-level, capture the power consumption of a particular instruction, in a particular mode with a particular parameter setting.

We then simulate each testbench using the gate-level model of *C*, and we analyze the average power consumption of the corresponding instruction, mode and parameter value. We tabulate these power results into our lookup tables. Table 2 gives the lookup table for the UART example. The rows correspond to instructions while columns correspond to the UART's buffer size parameter values. The entries are repeated for each one of the 4 modes.

Table 2: Four mode UART power lookup table.

	2 (bytes)	4 (bytes)	8 (bytes)	16 (bytes)
Mode 1: Idle				
Reset	11 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	27 µJ	32 µJ	31 µJ	31 µJ
Enable_rx	17 µJ	18 µJ	19 µJ	18 µJ
Send	17 µJ	19 µJ	19 µJ	20 µJ
Receive	14 µJ	15 µJ	17 µJ	18 µJ
Mode 2: Tx_enabled				
Reset	13 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	23 µJ	23 µJ	22 µJ	24 µJ
Enable_rx	19 µJ	20 µJ	19 µJ	19 µJ
Send	133 µJ	135 µJ	157 µJ	209 µJ
Receive	14 µJ	16 µJ	18 µJ	18 µJ
Mode 3: Rx_enabled				
Reset	13 µJ	14 µJ	15 µJ	15 µJ
Enable_tx	22 µJ	22 µJ	21 µJ	21 µJ
Enable_rx	18 µJ	19 µJ	18 µJ	18 µJ
Send	19 µJ	19 µJ	21 µJ	23 µJ
Receive	73 µJ	83 µJ	93 µJ	111 µJ
Mode 4: Tx_rx_enabled				
Reset	13 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	21 µJ	22 µJ	21 µJ	21 µJ
Enable_rx	19 µJ	19 µJ	19 µJ	19 µJ
Send	133 µJ	135 µJ	157 µJ	209 µJ
Receive	74 µJ	81 µJ	93 µJ	107 µJ

3.4 System-level modeling

The next step is to 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 called *C*, its instructions $i_1, i_2, i_3, \ldots i_n$, and its modes $m_1, m_2, m_3, \ldots m_k$, we implement a functional model of *C* in terms of its instructions.

If using method-calling objects [22], the interface to the object representing *C* would have the instructions $i_1, i_2, i_3, \ldots i_n$, as methods and the instruction's input/output data as parameters to the corresponding methods. To each object-oriented model, we add two data objects, called total-power (initialized to zero) and power-mode (initialized to "*Idle*".) We then augment the implementation of each method of *C*'s system level model with the code outlined in Figure 2.

3.5 System-level power evaluation

The above three steps are performed by the core developer. They may take days to complete, forming part of the months required to develop the core. They must be done for each target technology. But note that those steps have only to be performed one time. Once done, the resulting data can be used in any core-based design that is using the particular core.

The core user does not perform the above steps (unless re-targeting to a new technology, in which case the core provider may supply the necessary testbenches). Rather, the user connects the core models and simulates them. Simulation of a complete SOC, using system-level models, takes on the order of seconds or minutes. Thus, hundreds or thousands of configurations can be evaluated. The top-level simulation model will be designed to output the value of the total-power variable, for each core of the system, at the end of each simulation. The sum of these total-power values represents the system-level estimate of the system's power consumption for a given configuration of its parameters.

Table 3: One mode UART power lookup table.

	2 (bytes)	4 (bytes)	8 (bytes)	16 (bytes)
Reset	13 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	23 µJ	25 µJ	24 µJ	24 µJ
Enable_rx	18 µJ	19 µJ	19 µJ	19 µJ
Send	76 µJ	77 µJ	89 µJ	115 µJ
Receive	44 µJ	49 µJ	55 µJ	64 µJ

Table 4: Two mode UART power lookup table.

	2 (bytes)	4 (bytes)	8 (bytes)	16 (bytes)
Mode 1: Idle				
Reset	11 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	27 µJ	32 µJ	31 µJ	31 µJ
Enable_rx	17 µJ	18 µJ	19 µJ	18 µJ
Send	17 µJ	19 µJ	19 µJ	20 µJ
Receive	14 µJ	15 µJ	17 µJ	18 µJ
Mode 2 : Enabled				
Reset	13 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	23 µJ	25 µJ	24 µJ	24 µJ
Enable_rx	18 µJ	19 µJ	19 µJ	19 µJ
Send	76 µJ	77 µJ	89 µJ	115 µJ
Receive	44 µJ	49 µJ	55 µJ	64 µJ

4. Experiments

4.1 System architecture

To evaluate the accuracy and simulation speed of our approach, we applied it to three cores within an SOC. We will next describe our architecture, its three selected cores, relevant parameters, and application. Then we describe our results.

Our parameterized system-on-a-chip architecture, depicted in Figure 3, is as follows. A MIPS R2000 processor and instruction and data caches communicate over a high-speed processor-local bus. The on-chip memory and direct memory access (DMA) controller cores are connected to the system bus, which in turn is bridged to the processor-local bus via a bus controller. Universal Asynchronous Receiver and Transmitter (UART) and JPEG decoder cores are connected to the peripheral bus, which is bridged to the system bus. Both the UART and JPEG decoder cores are DMA capable. The DMA controller is capable of transferring data between peripheral cores and memory without the intervention of the processor. The processor can run concurrent to the DMA until a cache miss occurs, at which point the processor is blocked waiting for the DMA transfer to complete. The UART, DMA and JPEG decoder cores in our architecture are parameterized. The DMA controller can be instantiated with maximum block transfer size set to one of 4, 16, 64 or 128 bytes. The UART core's transmitter/receiver buffer sizes can each be set to one of 2, 4, 8, or 16 bytes. The JPEG decoder core's pixel resolution can be set to one of 10 or 12 bits. This architecture is used to implement a JPEG image decode accelerator. JPEG images are input serially through the UART, transferred via the DMA to memory, Huffman decoded by the MIPS, transferred from memory to the JPEG-decoder and back to the UART to be outputted to the host device. Most of these operations take place in a pipelined fashion for maximum throughput. We have RTL synthesis models for all three of the parameterized cores.

4.2 System-level functional simulation

We implemented system-level functional simulation models, as described in the previous sections, for all components in the JPEG image decode accelerator. Our gate-level characterization procedure for the three parameterized cores took roughly one week for synthesis, simulation and analysis. After this analysis, we obtained power lookup tables (as noted before, this step is only performed once by a core provider). We have used our model to evaluate power consumption of the three cores when processing a 640x480 pixel image, and have compared the results to gate-level power evaluations done by the Synopsys Power Compiler. Only the three cores are simulated at the gate-level and the remainder of the architecture is simulated at a behavioral VHDL level, for faster simulation, since we are only interested in power estimates for the peripheral cores. For our system-level simulation model, all components (including the MIPS) of the architecture are simulated as an executable binary obtained from the C++ models.

Our results are summarized in Figure 4. For comparison purposes, we have provided power estimates based on a cycle-based databook approach. Here, for each of the three cores, we measure the amount of time that the core was busy computing as well as the amount of time that the core spent idle. We then used idle and average power consumption data from the databooks (for similar technology to that used during our synthesis) to compute the total power consumption of each of the cores. This databook-based approach is similar to that outlined in [23]. Our results show that average system-level power error is 2.7% (min=1%, max=5%) when compared to gate-level power estimation. In contrast, average databook-based power error is 30% (min=14%, max=38%) when compared to gate-level simulation. Our simulation model for processing a 640x480 pixel image ran in just under 14 seconds while the gate-level simulation required 12 hours, i.e., our functional system-level approach ran 3000 times faster than gate-level simulation. In addition, we estimated the cycle-based simulation time to be about 200 times slower than our system-level approach.

We performed a second experiment to see the importance of the concept of power-modes. In this experiment, we implemented two additional UART system-level models, one with only a single power-mode and another with two power-modes. For our single mode model, we measured and recorded

Figure 4: Energy consumption of the JPEG image-decode accelerator.

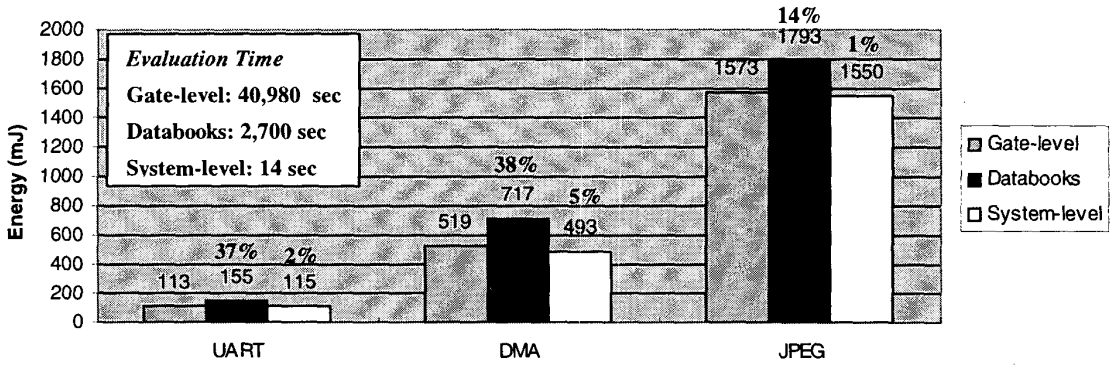


Table 5: UART's power evaluation accuracy for various power modes.

	Power (mJ)		Error
	Gate-level	System-level	Ref. Gate-level
Single-mode	113	86	23%
Two-modes		104	8.6%
Four-modes		115	1.7%

the lookup table shown in Table 3. The lookup table for our two-mode UART model is shown in Table 4.

We simulated the same JPEG image-decode accelerator application as in our first experiment. The power results for the single-mode, two-mode and four-mode UART are summarized in Table 5. Using a single mode only, which is the same as not considering modes, results in an error of 23%. Using two modes helps, but is still off by 8.6% Our four-mode model gave only 1.7% error. Thus, a proper selection of modes is important for accuracy.

In our next experiment, we examined the importance of proper instruction granularity selection, by using a smaller set of instructions for the UART core and comparing our power evaluation technique to our original experiment. The lookup table for this version of the UART is given in Table 6. Notice that we aggregated the *Send* and *Receive* instructions into a single *Send_or_receive* instruction. Such aggregation performed repeatedly will eventually result in a power evaluation model that considers the average power consumption for a core when that core is computing, i.e., a scheme similar to [23].

We ran our JPEG decode accelerator example using this UART model (with fewer instructions) and compared the results to those obtained from our original UART model. The results are given in Table 7. When compared to gate-level, selecting a smaller instruction set resulted in 17% error.

Based on our experiments, we conclude that our system-level, instruction based lookup table power evaluation technique for general cores is very accurate and fast. We further have shown that the selection of appropriate power-modes and instructions for a particular core is important in obtaining high power evaluation accuracy.

5. Conclusions

We have introduced an instruction-based 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 databook lookup approaches can be inaccurate since they are not sensitive to different data. Our future work includes generating peripheral instruction traces, and then using trace simulators to further speedup power estimation, similar to microprocessor and cache trace-simulator approaches.

6. Acknowledgement

This work was supported by the National Science Foundation (grants CCR-9811164 and CCR-9876006) and a Design Automation Conference Graduate Scholarship.

7. References

[1] S.M. Kang. Accurate Simulation of Power Dissipation in VLSI Circuits. IEEE Journal of Solid-State Circuits, vol. CS21, no. 5, pp. 889-891, October 1986.

[2] G.Y Yacoub, W.H. Ku. An Accurate Simulation Technique for Short-Circuit Power Dissipation Based on Current Component Isolation. IEEE International Symposium on Circuits and Systems, pp. 1157-1161, 1989.

[3] R. Tjarnstorm. Power Dissipation Estimate by Switch Level Simulation. IEEE symposium on Circuits and Systems, pp. 881-884, 1989.

[4] T.H. Krodel. PowerPlay - Fast Dynamic Power Evaluation Based on Logic Simulation. IEEE International Conference on Computer Aided Design, pp. 96-100, Oct. 1991.

Table 6: Four mode UART power lookup table using coarse instruction granularity.

	2 (bytes)	4 (bytes)	8 (bytes)	16 (bytes)
Mode 1: Idle				
Reset	11 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	27 µJ	32 µJ	31 µJ	31 µJ
Enable_rx	17 µJ	18 µJ	19 µJ	18 µJ
Send_or_receive	15 µJ	17 µJ	18 µJ	19 µJ
Mode 2: Tx_enabled				
Reset	13 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	23 µJ	23 µJ	22 µJ	24 µJ
Enable_rx	19 µJ	20 µJ	19 µJ	19 µJ
Send_or_receive	74 µJ	76 µJ	88 µJ	114 µJ
Mode 3: Rx_enabled				
Reset	13 µJ	14 µJ	15 µJ	15 µJ
Enable_tx	22 µJ	22 µJ	21 µJ	21 µJ
Enable_rx	18 µJ	19 µJ	18 µJ	18 µJ
Send_or_receive	46 µJ	51 µJ	57 µJ	67 µJ
Mode 4: Tx_rx_enabled				
Reset	13 µJ	13 µJ	14 µJ	14 µJ
Enable_tx	21 µJ	22 µJ	21 µJ	21 µJ
Enable_rx	19 µJ	19 µJ	19 µJ	19 µJ
Send_or_receive	104 µJ	108 µJ	125 µJ	158 µJ

Table 7: UART's power evaluation accuracy with coarser instruction granularity.

	Power (mJ)			Error
	Gate-level	System-level	System-level (coarse granularity)	Ref. Gate-level
UART	113	115	94	17%

[5] A. Raghunathan, S. Dey, N.K. Jha. Register-transfer level evaluation techniques for switching activity and power consumption. International Conference on CAD Aided Design, pp. 158-165, 1996.

[6] S. Gupta, F. Jajm. Power Macromodeling for High Level Power Evaluation. Design Automation Conference, June 1997.

[7] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, G. De Micheli. Lookup Table Power Macro-Models for Behavioral Library Components. Design Automation and Test In Europe, March 1998.

[8] P. Landman, J. Rabaey. Architectural Power Analysis: The Dual Bit Type Method. IEEE Transactions on VLSI Systems, vol. 3, no. 2, June 1995.

[9] H. Mehta, R. Owens, M.J Irwin. Energy Characterization Based on Clustering. Design Automation Conference, June 1996.

[10] E. Macii, M. Pedram. High-Level Power Modeling, Evaluation, and Optimization. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 17, no. 11, November 1998.

[11] D. Marculescu, R. Marculescu, M. Pedram. Information Theoretic Measures for Power Analysis. IEEE Transactions on Computer Aided Design, vol. 15, no. 6, pp. 599-610, 1996.

[12] M. Nemani, F. Najm. Toward a High Level Power Evaluation Capability. IEEE Transactions on Computer Aided Design, vol. 15, no. 6, pp. 588-598, 1996.

[13] V. Tiwari, S. Malik, A. Wolfe. Power Analysis of Embedded Software: A First Step Toward Sofware Power Minimization. IEEE Transactions on VLSI Systems, vol. 2, no. 4, pp. 437-445, 1994.

[14] C.T. Hsieh, M. Pedram, H. Mehta, F. Rastgar. Profile Driven Program Synthesis for Evaluation of System Power Dissipation. Design Automation Conference, June 1997.

[15] C. Barndolese, W. Fornaciari, F. Salice, D. Sciuto. Energy Evaluation for 32-bit Microprocessor. International Workshop on Hardware/Software Co-Design, 2000.

[16] R. J. Evans, P.D. Franzon. Energy Consumption Modeling and Optimization for SRAMs, IEEE Journal of Solid-State Circuits, Vol. 30, No. 5, pp. 571-579, 1995.

[17] T.D. Givargis and F. Vahid. Interface Exploration for Reduced Power in Core-Based Systems, ISSS, 1998, pp. 117-122.

[18] T.D. Givargis, J. Henkel, and F. Vahid. Interface and Cache Power Exploration for Core--Based Embedded System Design. ICCAD 1999.

[19] T. Simunic, L. Benini, G. De Micheli. Cycle-accurate Evaluation of Energy Consumption in Embedded Systems. Design Automation Conference, pp. 876-872, 1999.

[20] Y. Li, J. Henkel. A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems. Design Automation Conference, pp. 188-193, 1998.

[21] Virtual Socket Interface Association, Architecture Document, http://www.vsi.org, 1997.

[22] F. Vahid, T.D. Givargis. Incorporating Cores into System-Level Specification. International Symposium on System Synthesis, November 1998.

[23] T. Simunic, L. Benini, G. De Micheli. Cycle-Accurate Simulation of Energy Consumption in Embedded Systems. Design Automation Conference, June 1999.