Efficient Power Co-Estimation Techniques for System-on-Chip Design

Marcello Lajolo
Politecnico di Torino
lajolo@polito.it

Anand Raghunathan
NEC USA, C&C Research Labs, Princeton, NJ
anand@ccrl.nj.nec.com

Sujit Dey
UC San Diego, La Jolla, CA
dey@ece.ucsd.edu

Luciano Lavagno
Università di Udine
lavagno@uniud.it

Abstract

We present efficient power estimation techniques for HW/SW System-On-Chip (SOC) designs. Our techniques are based on concurrent and synchronized execution of multiple power estimators that analyze different parts of the SOC (we refer to this as *co-estimation*), driven by a system-level simulation master. We motivate the need for power co-estimation, and demonstrate that performing independent power estimation for the various system components can lead to significant errors in the power estimates, especially for control-intensive and reactive embedded systems.

We observe that the computation time for performing power co-estimation is dominated by: (i) the requirement to analyze/simulate some parts of the system at lower levels of abstraction in order to obtain accurate estimates of timing and switching activity information, and (ii) the need to communicate between and synchronize the various simulators. Thus, a naive implementation of power co-estimation may be too inefficient to be used in an iterative design exploration framework. To address this issue, we present several acceleration (speedup) techniques for power co-estimation. The acceleration techniques are energy caching, software power macro-modeling, and statistical sampling. Our speedup techniques reduce the workload of the power estimators for the individual SOC components, as well as their communication/synchronization overhead. Experimental results indicate that the use of the proposed acceleration techniques results in significant (8X to 87X) speedups in SOC power estimation time, with minimal impact on accuracy. We also show the utility of our co-estimation tool to explore system-level power tradeoffs for a TCP/IP Network Interface Card sub-system and an automotive controller.

1 Introduction

The System-on-Chip paradigm has the potential to offer the designer several benefits (including improvements in system cost, size, performance, and power dissipation) if system-level tradeoffs are well explored. Achieving the disparate goals of reduction in design turn-around-time while better exploring system-level tradeoffs requires efficient and accurate analysis tools that guide the designer in the initial stages of the system design process.

A large class of applications which stand to benefit significantly from system-level integration are portable products in consumer electronics and communications, which have a critical need for minimizing power consumption to prolong battery life. Reducing chip packaging and cooling costs, which may be significant for deep sub-micron SOCs, is another motivation to investigate low-power system design methodologies. Power analysis and optimization at the early stages of the design cycle, starting from the system-level, has been shown to yield large power savings, and can lead to fewer and faster design iterations for designs with aggressive power consumption constraints [1, 2, 3].

1.1 Previous work

Previous work on low-power design techniques has mostly focussed on estimating and optimizing power consumption in the individual system-on-chip components (application-specific hardware, embedded software, memory hierarchy, buses, *etc.*) separately. Various power estimation and minimization techniques for hardware at the transistor, logic, architecture, and algorithm levels have been developed in the recent years, and are summarized in [1, 2, 3, 4, 5]. Power analysis techniques have been proposed for embedded software based on instruction-level characterization [6] and simulation of the underlying hardware [7]. Power estimation and optimization techniques for other system-on-chip components, including memory hierarchies and system buses, have been proposed, and are surveyed in [2, 8].

Recently, researchers have started investigating system-level tradeoffs and optimizations whose effects transcend the individual component boundaries. Techniques for synthesis of multiprocessor system architectures and heterogeneous distributed HW/SW architectures for real-time specifications were presented in [9, 10]. These approaches either assume that all tasks are pre-characterized with respect to all possible implementations for delay and power consumption, or assume a significantly simplified power dissipation model. In [11] and [12], separate execution of an instruction set simulator (ISS) based software power estimator, and a gate-level hardware power estimator were used to drive exploration of tradeoffs in an embedded processor with memory hierarchy, and to study HW/SW partitioning tradeoffs. The traces for the ISS and hardware power estimator were obtained from timing-independent system-level behavioral simulation.

The power estimation techniques used to drive the above system-level power optimization techniques may either require an unacceptable amount of pre-characterization work (*e.g.*, measuring power consumption of all tasks on all candidate processors), or, as shown later in this paper, they may significantly compromise the accuracy of the power estimates due to factors including the following:

- The presence of shared system resources (*e.g.* processors running multiple tasks, shared memory hierarchies and buses). The activity in such shared resources (and hence their power consumption) is dependent on the manner in which the interactions with the various system components are interleaved in time.
- Performing separate power estimation for the different system components using input traces obtained from (timing-independent) behavioral simulation can lead to significant errors in the power estimates. In many systems (especially control-flow intensive, event-driven systems), the input traces and hence internal execution traces of various components are inter-dependent and timing-sensitive, requiring that power estimation be performed through simultaneous analysis of timing and power models of the various system components.

Recently, a cycle-accurate energy estimation framework for an embedded processor based system was proposed in [13], which demonstrated that high accuracies can be obtained through the integration of performance and energy models for various system components.

1.2 Paper Overview and Contributions

The aim of our work is to provide a power co-estimation [1] framework for HW/SW System-on-Chip designs that is accurate and efficient enough to drive coarse-level (*e.g.* HW/SW partitioning, component selection) as well as fine-grained (*e.g.*, tuning the parameters of a specific bus) system design tradeoffs. As shown later in the paper, power co-estimation is necessary in order to obtain accurate power estimates. This is especially important for control-intensive and reactive embedded systems where the input data and execution traces of the various components are often highly inter-dependent and timing-sensitive.

The various power estimators plugged in to our co-estimation framework could possibly operate at different levels of abstraction (*e.g.*, RTL/gate-level power simulator for the application-specific hardware parts, ISS-based power estimator for the embedded software, and a behavioral model based power estimator of the SOC integration architecture). This heterogeneity can be exploited to employ more accurate models for the harder-to-model parts of the system, and use more efficient models for the easier-to-model parts (*e.g.*, regular structures). It is important to note that the focus of this work is on how to combine power estimation techniques for individual SOC components in an accurate and efficient manner. Thus, advances in power estimation techniques for hardware and embedded software can be easily incorporated into our SOC power co-estimation framework.

We demonstrate that the computation time in a power-co-estimation framework is often dominated by: (i) the requirement to analyze/simulate some hard-to-model parts of the system at lower levels of abstraction in order to accurately estimate timing and switching activity, and (ii) the need to communicate data and synchronize between the various simulators (*e.g.* hardware simulator and ISS [14]). A naive implementation of power co-estimation may be too inefficient to be used in an iterative design exploration framework. This creates an important need for techniques to improve the computational efficiency of SOC power co-estimation.

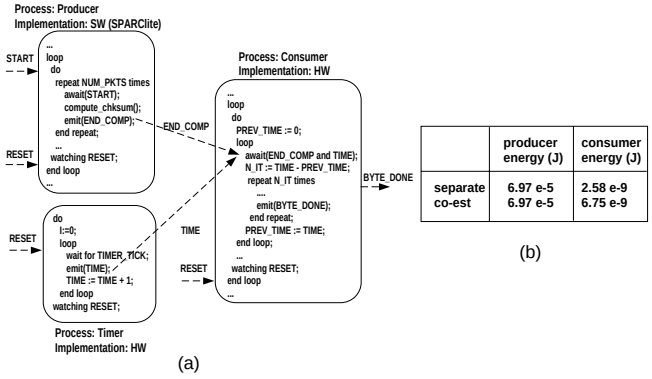
One way to improve the efficiency/accuracy of our SOC power estimation framework is to improve the efficiency/accuracy of the individual component power estimators. However, our focus in this work is in a complementary direction - on using top-level (system-level) information to accurately and efficiently employ the individual component estimators. We believe our work to be the first in that direction. We present acceleration (speedup) techniques - energy caching, software power macro-modeling, and statistical sampling - which help reduce the workload of each of the lower-level simulators, as well as the communication/synchronization overhead.

The rest of the paper is organized as follows. Section 3 presents the basic SOC power co-estimation framework. Section 4 presents our acceleration techniques that significantly improve the efficiency of co-estimation, with minimal impact on accuracy. Experimental results demonstrating some applications of our SOC power estimation framework, and evaluating the efficiency and accuracy of our acceleration techniques, are presented in Section 5.

2 Motivation for co-estimation

This section motivates the need for power co-estimation by illustrating, using a simple example system, that separately performing power estimation for the various system components can lead to significant errors in the power estimates.

[1] **Power co-estimation** refers to system power estimation where the various system components are analyzed simultaneously (*e.g.*, using co-simulation) in order to accurately model their interactions and power inter-dependencies



	producer energy (J)	consumer energy (J)
separate	6.97 e-5	2.58 e-9
co-est	6.97 e-5	6.75 e-9

Figure 1: Illustrating the need for co-estimation: (a) An example system, and (b) energy estimates obtained using separate HW/SW estimation and co-estimation

Consider the example system shown in Figure 1(a), that is described as a set of concurrent processes with event-based communication. The process `producer`, upon being triggered by an event from the environment, performs a computation and sends some data to process `consumer`. Process `timer`, as its name indicates, keeps track of the current time and sends it to process `consumer`. Process `consumer`, upon receiving data from `producer`, uses the difference between the times at which it received the current and previous data sent by `producer`, and executes a computation loop whose iteration count depends on the above mentioned time difference. An important feature to note is that the time interval between two consecutive data packets sent by `producer` impacts the computations performed by `consumer`. Such timing-functionality inter-dependence can occur in control-intensive systems, including telecommunication protocols and networking applications. For example, when multiple packets are waiting for transmission, it is not uncommon to use the time at which each packet arrived to decide in what order packets will be processed for transmission.

We consider an implementation of the example system where the `producer` process is implemented using embedded software running on a SPARClite processor, and the `consumer` process is synthesized into application-specific HW. To illustrate the need for co-estimation, we performed power estimation using two different approaches for the system shown in Figure 1(a).

- In the first experiment, we performed an initial behavioral simulation of the system, during which we captured input traces for the various system components. These traces were then used to drive power estimation for the individual components using a gate-level HW power estimator [15] and a SW power estimator based on an instruction set power model [6].

- In the second experiment, we used our tool (described in Section 3) to perform power co-estimation through concurrent and synchronized execution of the same individual (HW and SW) power estimators that were used in the first experiment.

The results of the two experiments are presented in Figure 1(b). The figure presents the total energy consumed in the `producer` and `consumer` processes in order to process a fixed pre-defined amount of data. The energy estimates obtained using independent execution of the HW and SW power estimators, and co-estimation are shown in rows `separate` and `co-est`, respectively. Note that the energy estimates obtained for process `consumer` through separate estimation and co-estimation differ significantly (the `separate` estimate leads to an under-estimation of about 62%). Upon further analysis, we confirmed that this error could be attributed to be the inter-dependence of the input (and hence internal execution) traces of the `producer` and `consumer` processes, as described earlier. Thus, power co-estimation is nec-

essary in order to accurately explore system-level tradeoffs.

3 Co-simulation based SOC power estimation system

Our System-On-Chip power co-estimation framework is described in Figure 2. While we have implemented them in the context of the POLIS system design environment [16], we believe our techniques are quite general in nature and can be easily adapted to other HW/SW co-design flows as well. In POLIS, the system is described at the behavioral level as a set of concurrent communicating processes. The user is allowed to specify the mapping of each process to hardware or embedded software, set Real-Time Operating System (RTOS) parameters such as scheduling policy and priorities, map the communication events between processes to shared buses or dedicated channels, and specify the parameters for shared buses. A refined description of the various system components (HW netlist, SW for target compilation, RTOS, *etc.*) is automatically generated for simulation with the PTOLEMY [17] simulation platform.

The compilation process that needs to be performed prior to co-estimation is illustrated in Figure 2(a). Starting with a system-level specification and various implementation parameters and constraints, POLIS is used to generate C code for each process that could potentially be mapped to SW and a hardware netlist for each process that could potentially be mapped to HW. A compiler for the target processor is used to compile these C descriptions (together with an RTOS that is automatically generated by POLIS), into object or executable code for the target processor. In addition to the HW and SW descriptions, a behavioral discrete-event [17] model of the entire system is generated, including the software processes, the hardware processes, the RTOS, and a model of the SOC integration architecture (bus). The discrete-event model is used to synchronize the execution of the hardware and software simulators/power estimators The shaded parts of Figure 2(a) are then passed on to the run-time flow for power co-estimation.

The run-time flow for power co-estimation in our tool is shown in Figure 2(b). An enhanced Instruction Set Simulator [2] for the target embedded processor is used to simulate the SW parts of the system, while a hardware power simulator is used to estimate power consumption in the HW parts of the system. The HW power simulator could be either a register-transfer level [2, 18] or a gate-level [2, 4] simulator that reports power consumed on demand at cycle-level accuracy. Cache simulation is not performed with the ISS simulator (it just assumes 100% cache hits); instead, it is performed by a fast cache simulator attached directly to the PTOLEMY simulator [19]. The PTOLEMY simulator simulates the discrete-event model of the entire system, synchronizes and transfers the necessary data to/from the HW simulator and ISS, and provides source-level graphical interface and debugging capabilities. Thus, the PTOLEMY simulator has a global view of the entire system under simulation, as opposed to the HW and SW simulators which view only their respective parts [20]. Having a single simulation master not only makes it easier to provide synchronization between the various simulators, but also facilitates some of the speedup techniques that are described in the next section.

The co-estimation is performed as follows. Initially, the simulation master (PTOLEMY) invokes the HW and SW power estimators, instructs them to load the appropriate netlists and target executables, and inserts breakpoints at appropriate locations to enable synchronization with them [20]. The unit of synchronization is a CFSM transition [3], but it can also be specified as a clock cycle in general. As mentioned previously, PTOLEMY simulates the discrete-event behavioral model of the entire system. Whenever a process executes in the discrete-event model (in response events at its inputs), PTOLEMY sends a sequence instructions to the appropriate simulator, which results in the lower-level simulator simulating the CFSM transition and returning performance and energy statistics for the instructions/vectors simulated. PTOLEMY collects the cycles and energy statistics for each invocation of the lower-level simulators, performs the necessary book-keeping, and can display energy and power waveforms for the various parts of the system.

Hardware Power Estimation

The hardware power estimator must accept from the simulation master a sequence of input vectors (that correspond to the synchronization period), simulate the netlist for the appropriate component, and return a cycle-by-cycle report of the energy dissipated during the process. A popular approach to hardware power estimation, that fits in well with the above requirements is through simulation of the hardware netlist. The hardware netlist may be represented at the RT-level or the gate-level, depending on the accuracy/efficiency requirements. It is also possible to use HW power estimation techniques that use aggregate signal statistics (*e.g.* probabilistic or statistical power estimation techniques [1, 2, 18, 3, 4]) in our framework, when the user does not desire cycle-by-cycle power information for the HW parts.

Software Power Estimation

Power estimation for embedded software can be performed using a hardware power estimator simulating an architectural/RTL HW model of the target processor [7], or using an instruction set simulator (ISS) that has been enhanced with an instruction-level power model [6]. The former approach, although accurate, may be time-consuming. Besides, a detailed HW model for the embedded processor is often not available to the system designer. The latter approach (instruction-level power models) relates power consumption in the processor to each instruction or sequence of instructions it can execute. Additional refinements are made to the power consumption estimate using statistics such as cache hits/misses, pipeline stalls, *etc*. An Instruction Set Simulator for the target processor may be enhanced using such a model.

For the purpose of our work, we currently use the software power estimation framework described in [11] for an embedded SPARC target processor. However, it could be replaced with any ISS or HW model of the processor that supports symbolic break-pointing and debugging, and reports energy consumption and clock cycle statistics at each breakpoint.

Estimating Power Consumed in the SOC bus/integration architecture

While the SOC integration architecture can itself contribute significantly to the total system power consumption, equally important is how it affects the power consumption in the other system components. The SOC integration architecture consists of the bus itself (interconnect and bus drivers/repeaters), and the bus arbiter. The power consumed in the bus interconnect and drivers is computed using the simple formula $P_{bus} = \frac{1}{2}.V_{dd}^2.f.\sum_{bus\ lines} C_{eff}(line_i).A(line_i)$. The user is required to specify the expected/budgeted dimensions of the bus from a system-level floorplan or budget, from which the effective capacitance $C_{eff}(line_i)$ is calculated by accounting for the wiring as well buffers/repeaters. The switching activity $A(line_i)$, however, needs to be computed during co-simulation. We use parameterizable, behavioral models of the SOC integration architecture [21] to generate a trace of all bus transactions. This bus trace is used to compute the switching activity at each line of the bus. The behavioral bus architecture models allow the user to dynamically change values for various parameters (such as bus access priorities, data/address widths, whether DMA is supported, the maximum allowed DMA block size, *etc.*), and to re-run power co-estimation without requiring a re-compilation of the system description.

4 Power Co-estimation speedup techniques

While the power co-estimation procedure described in the previous section is fairly accurate for supporting system-level optimizations, it can be quite slow especially when invoked in an iterative man-

[2] The ISS is enhanced to report power/energy consumption in addition to clock cycle statistics.

[3] A CFSM transition is an atomic operation that can be performed by each process in response to events at its inputs [16].

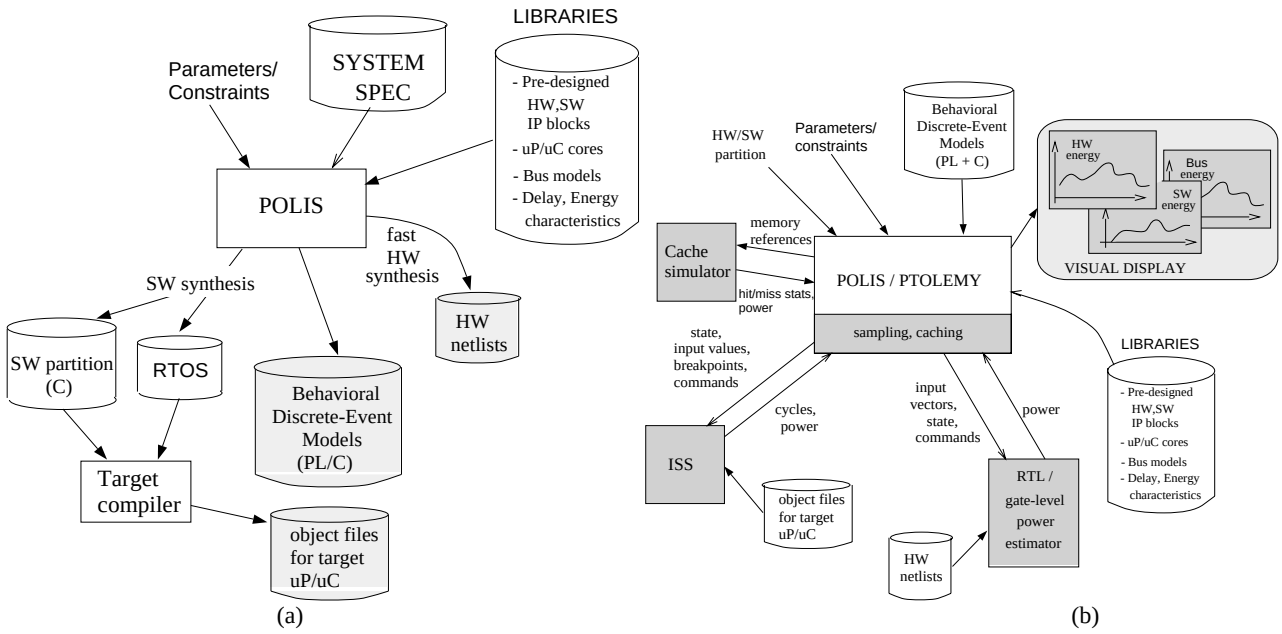


Figure 2: Power estimation framework for HW/SW systems (a) Compilation flow, and (b) Simulation flow

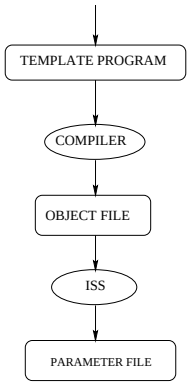
ner for design exploration. This section describes novel techniques that can be employed to improve the efficiency of the power co-estimation process, while attempting to keep the loss in accuracy as low as possible.

4.1 Macromodeling

The idea of macro-modeling is derived from RT-level hardware power estimation [2, 18, 5]. In this section, we illustrate how we have applied the idea of macro-modeling to embedded software. However, the approach in the case of hardware is quite similar.

Software macro-modeling refers to the pre-characterization of a comprehensive set of high-level macro-operations in terms of various metrics such as code size, performance, and power. For example, a macro-operation could be an arithmetic operation with assignment to a variable, an emission of an event, *etc.*. Characterization is performed by compiling each macro-operation down to a sequence of assembly-level instructions for the target processor, and computing its power dissipation and delay using an instruction-level simulator. The data resulting from the characterization process is stored in a macromodel library. Prior to HW/SW co-simulation, the software parts of the system are mapped in terms of the pre-characterized macro-operations. The delay and energy costs from the library are automatically annotated into the behavioral discrete-event models for each software process. During co-simulation, when a macro-operation is executed, the pre-computed numbers for energy and delay are used, without invoking the ISS. Since this approach raises the level of abstraction at which the embedded software is modeled, it can be quite efficient. However, the efficiency does come at a cost in terms of accuracy. Architectural effects such as pipeline, cache and target-specific compiler optimization effects, are difficult to model at this level. However, as shown in Section 5, the relative accuracy of this approach can still be reasonable, making it useful when exploring coarse tradeoffs in HW/SW co-design.

Figure 3 shows the characterization flow used for software macro-modeling in our co-estimation tool on the left, and a part of the resulting parameter file on the right. Macro-operations in POLIS are characterized through a library of benchmark programs written in C that consist of about 20 functions. Synthesized programs can also contain pre-defined arithmetic, relational and logical functions (the POLIS software library currently includes about




```
.unit_time    cycle
.unit_size     byte
.unit_energy     nJ
 .....
.time AVV         5
.time TIVART     11
.time TIVARF      9
.time AEMIT      12
 .....
.size AVV         7
.size TIVAR       8
.size AEMIT       8
 .....
.energy AVV     110
.energy TIVAR   210
.energy AEMIT   680
```

Figure 3: The software macromodeling flow in POLIS and a portion of a parameter file

30 such functions, *e.g.*, `ADD(x1,x2)`, `NOT(x1)`, `EQ(x1,x2)`, *etc*). Each of these functions is pre-characterized in term of delay, code size, and energy dissipation and inserted into the parameter file. For example, the parameter file shown in Figure 3 contains information for three macro-instructions: assignment of a variable to another variable (`AVV`), event emission (`AEMIT`), and tests on a variable value (`TIVART`, `TIVARF`, `TIVAR`).

4.2 Energy and delay caching

In our experiments with HW/SW power co-estimation, we observed that a few "paths of computation" in the hardware and software components were executed a large number of times, while a large number of paths were executed relatively few times. This conforms to the empirical observation made for application programs that a small fraction (*e.g.* 10%) of the code accounts for a large part (*e.g.* 90%) of the total execution time [22]. In general it is possible that each execution of a segment of code results in a distinct delay and energy consumption. However, in practice, we also observed that the number of distinct energy and delay values for a single code segment is typically much smaller than the number of times it is simulated, leading to several simulations of a code segment computing

similar energy/delay numbers. We propose a technique, called *energy caching*, that exploits the above observation to significantly enhance the efficiency of power co-estimation. The basic idea is applicable to HW simulation as well as SW (ISS) simulation, but is explained in the context of software below.

The idea of energy caching is illustrated in Figure 4. Consider the small example code fragment shown as a control-flow graph in Figure 4(a). We focus on two specific paths, $1, 3, 6, 8$, and $1, 4, 7, 8$ since they are executed a large number of times during co-simulation. The energy dissipated during each execution of these two paths during a long system co-simulation is presented in the form of energy histograms in Figure 4(b). Note that, the histogram for path $1, 4, 7, 8$ is highly clustered around the mean, indicating a low variance in energy dissipation. The histogram for path $1, 3, 6, 8$, on the other hand, is quite spread out, which implies that the energy dissipation in this path varies to a greater extent. The intuition behind energy caching is that for low-variance paths (such as $1, 4, 7, 8$ in the above example), it may be acceptable to approximate the energy dissipation of the path by the mean of the histogram, while for other paths such as $1, 3, 6, 8$ it is better to use the lower-level simulator (ISS) each time the path is executed.

Energy caching is implemented in our co-simulation tool as described in Figure 4(c). We dynamically create a lookup table, or cache, containing energy and delay characteristics for the most frequently executed segments of code. The pseudo-code for constructing and using the energy cache is shown in Figure 4(c), along with a snapshot of the energy cache for the example of Figure 4(a). The energy cache is constructed using the results reported from the ISS for the first few times the code path or segment is executed. Segments could be specified at any level of granularity, including basic blocks, or threads of computation involving sequences of basic blocks. For each path, we store only the mean and variance of the data computed using the calls to the ISS. The cached value of energy is used only if the path has been simulated by the ISS a certain minimum number of times, and has displayed a variance below a certain threshold.

Two user-specified parameters are provided to determine the aggressiveness of the caching technique, and can hence be used explore the tradeoff between accuracy and efficiency. A parameter $thresh\_variance$ is used to specify a variance threshold - caching will be used only for paths whose variances fall below this threshold. A second parameter, $thresh\_iss\_calls$, is used to ensure that the ISS is called a certain minimum number of times for a segment before cached data is used for it. Once the above conditions are satisfied for a segment/path, the lookup table is used to estimate its delay and energy for all future simulations (this can eliminate the need to run the ISS for a large number of path executions, leading to a significant speedup in power co-estimation time).

4.3 Statistical Sampling / Sequence Compaction

The techniques of statistical sampling and sequence compaction have been extensively used in the context of hardware power estimation [1, 2, 18, 3, 4, 5]. A related idea was proposed in the context of software power estimation [23], where the aim was to synthesize a small program to match certain characteristics derived from the profile of a larger benchmark program, such as average block length, cache miss rate, branch prediction miss rate, *etc.*

As mentioned in Section 1, our objective in this work is not to improve upon or develop new techniques for hardware or software power estimation, but to assemble existing power estimation techniques for individual SOC components into a tool for SOC power co-estimation. Similarly, some of the speedup techniques such as statistical sampling have been studied by previous researchers in the context of power estimation for individual SOC components. We illustrate how to apply sampling in the context of an SOC power estimation framework.

In the context of SOC power estimation work, the problem of sequence compaction is as follows: Given a long sequence I of input vectors (instructions) generated from the master simulator during co-simulation, construct another sequence I' such that $length(I') \ll length(I)$, and the average power consumption of I' is as close as possible to that of I. We construct I' by composing small sub-sequences of I, such that the power consumption reported by the HW simulator (ISS) for the compacted sequence I' is as close as possible to the power consumption for the original sequence I. Sequence compaction may be either static or dynamic. In static sequence compaction, the I' is composed only when the complete original sequence I is available. In dynamic compaction, I is only available incrementally, and I' is generated without waiting to see all of I. Clearly, static compaction is more powerful than dynamic compaction since we are allowed to observe and manipulate the entire original sequence I. However, when the length of I is very large, dynamic compaction may be preferable. We use the following parameterizable *K-memory dynamic compaction* procedure in our co-estimation tool to compact the vector/instruction sequence fed to the lower-level simulator each time it is called:

- Store the input vectors (instructions) that are received from the simulation master (PTOLEMY), until the number of vectors stored becomes K.
- Then deterministically select a subset of the instructions from among those K instructions, to be dispatched to the HW power simulator (ISS).

The subset of input vectors (instructions) is chosen in the second step above so that it "represents" the sequence of K vectors (instructions) as well as possible. For HW, the criterion used is to preserve single-cycle (signal value probabilities, spatial correlations) and two-cycle (signal transition probabilities, lag-one temporal and spatio-temporal correlations) statistics of the primary inputs. For SW, the criterion is also to preserve the single-instruction and two-instruction statistics as much as possible.

5 Applications and Experimental Results

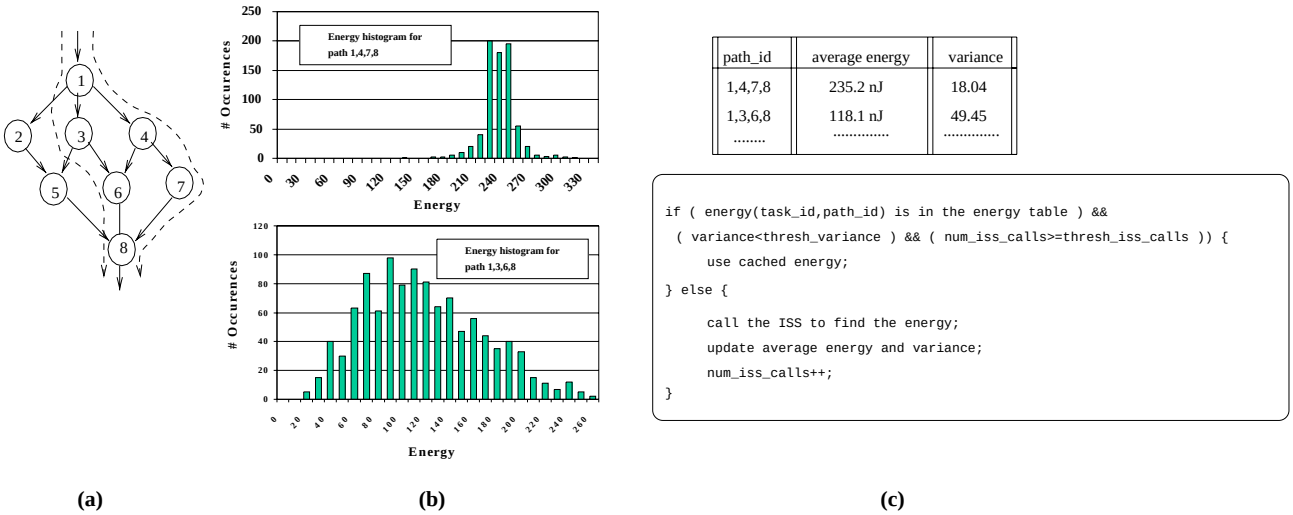
We have implemented the various techniques described in this paper in a SOC power co-estimation tool, and augmented it with the acceleration techniques described in Section 4. We performed several experiments to evaluate accuracy and efficiency of our SOC power estimation framework when the various speedup techniques described in this paper are employed. We also demonstrate the utility of our power co-estimation framework in driving system-level power tradeoffs, using a TCP/IP subsystem from a wireless Network Interface Card [21].

The rest of this section is organized as follows. Section 5.1 describes our implementation and experimental setup, and gives a brief introduction to the two example systems that are used in our experiments. Section 5.2 presents results on the accuracy and efficiency of various speedup techniques employed in our power estimation tool. Section 5.3 describes the use of our power co-estimation tool to explore the effects of customizing the communication architecture for the TCP/IP subsystem.

5.1 Experimental Methodology

The basic power estimation framework was constructed by adapting the following tools: (i) The POLIS/PTOLEMY system simulation environment [16] as the simulation master, (ii) a modified power simulator from the SIS [15] logic synthesis environment for hardware power estimation, (iii) an instruction-set simulator, SPARCSIM, for the SPARClite embedded processor that has been enhanced with the measurement-based instruction-level power model from [6], and (iv) a behavioral power model of the bus architecture described in [21].

The instruction set simulator accurately models timing behavior taking into account register interlocks, pipeline flushes in case of branches, delayed branches, register windowing *etc*. The instruction-level energy model from [6] was based on actual measurements, and has been shown to be accurate to within 5% compared to actual current measurements. For hardware power estimation, we have modified the SIS power estimation tool [4] to report power consumption cycle-by-cycle for a given input vector se-



path_id	average energy	variance
1,4,7,8	235.2 nJ	18.04
1,3,6,8	118.1 nJ	49.45
.....	.....	.....

```
if ( energy(task_id,path_id) is in the energy table ) &&
 ( variance<thresh_variance ) && ( num_iss_calls>=thresh_iss_calls )) {
     use cached energy;
} else {
     call the ISS to find the energy;
     update average energy and variance;
     num_iss_calls++;
}
```

(a) **(b)** **(c)**

Figure 4: Illustration of energy and delay caching (a) an example code fragment, (b) energy histograms, and (c) pseudo-code for the application of caching

quence. It is possible to run power estimation interactively during co-simulation instead of waiting for the end of the co-simulation session, and also run hardware power analysis in batch-mode on a long traces. We have implemented software power macro-modeling for the embedded SPARC processor in our power co-estimation framework. This required delay and energy characterization of several template programs using the SPARCsim ISS, as explained in Section 4. Finally, we have enhanced the co-estimation tool to incorporate the caching and sampling based speedup techniques.

We next describe briefly the example system used in our experiments.

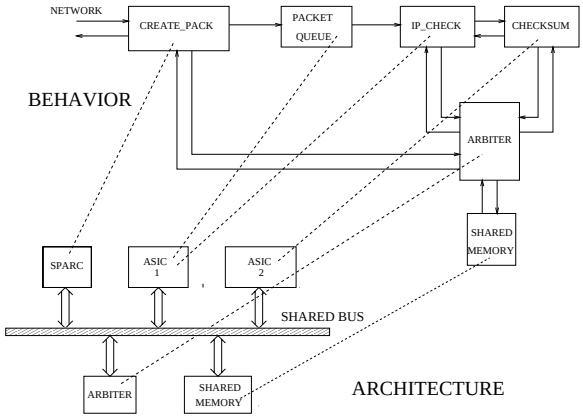


Figure 5: Specification and Implementation Architecture for the TCP/IP system

TCP/IP Network Interface Card Subsystem
The system behavior and one possible implementation architecture for the TCP/IP sub-system is shown in Figure 5. This sub-system consists of the parts of the TCP/IP protocol that perform checksum computation. For incoming packets, the module `create_pack` receives a packet from the lower layer (in this case, the IP layer), and stores it in the shared memory. When it finishes, it sends the information about the starting address of the packet in memory, the number of bytes and the checksum header to a queue (`packet queue`). From this queue, the module `ip_check` retrieves a new packet, overwrites parts of the checksum header (which should not be used in the checksum computation) with 0s, and signals to the `checksum` process that a new packet can be checked for checksum consistency. The `checksum` process performs the core part of the checksum computation, accessing the packet in memory through the arbiter and accumulating the checksum for the packet body. When it is done, it sends the computed 16-bit checksum back to the `ip_check` process, which then compares the computed checksum with the incoming transmitted checksum, and flags an error if they do not match. The flow for outgoing packets is similar, but in the reverse direction, and there is no need for comparison of the final checksum.

5.2 Results on Accuracy vs. Efficiency for the Acceleration Techniques

As mentioned in earlier sections, it is important that SOC power estimation techniques be efficient in order to be used iteratively in an interactive/automatic system-level design space exploration framework. In this section, we present results to illustrate the efficacy of our speedup techniques in reducing co-estimation time while sacrificing as little accuracy as possible.

Table 1: Speedup and accuracy of the caching approach

DMA size	Orig.		Caching	Speedup
	Energy (mJ)	CPU time (s)	CPU time (s)	
2	0.54	8051.52	428.92	18.8
4	0.44	4023.36	248.13	16.2
8	0.39	2080.77	156.91	13.3
16	0.36	1398.77	117.90	11.9
32	0.35	852.25	90.88	9.4
64	0.34	680.78	78.88	8.6

Table 1 reports energy and CPU time results for the SOC power co-estimation framework without any acceleration technique (major column **Orig.**), and with the use of caching to speedup SOC power co-estimation (major column **Caching**). Sub-columns **Energy** and **CPU time** present the total system energy consumption and simulation time on a Sun Ultra Enterprise 450 workstation with 256MB of memory. The speedup ($\frac{CPU\ time\ for\ Orig.\ case}{CPU\ time\ for\ Caching\ case}$) is presented in the last column. The rows represent different implementations of the TCP/IP subsystem with different values of the bus

DMA size.

It bears mentioning that in the case of this example, the caching acceleration technique did not result in any loss of accuracy with respect to the base case without any acceleration technique. That is the reason why no separate **Energy** sub-column is reported for the **Caching** case. Upon investigation, we found that the energy numbers were the same because the instruction-level power model used for the SPARClite embedded processor [6] does not model the dependence of the power consumed by an instruction sequence on the actual data values used in each instruction (such variations were empirically shown to be very small for this processor [6]). In addition, since the memory references are issued to the cache simulator directly by PTOLEMY from the simulation of the Discrete Event model of each CFSM, not invoking the ISS for a CFSM path (and using cached delay/power numbers instead) will not result in any change in the sequence of references issued to the cache simulator. However, in general, for processors where the ISS models the dependency of power on the data used in each instruction (*e.g.* for DSPs), there will be some non-zero error introduced due to caching the energy/delay numbers from the ISS. However, as mentioned in Section 4.2, the parameters used in the caching procedure can be used to minimize such errors.

Table 2 reports energy and simulation CPU time results for the SOC power estimation framework without any acceleration technique (major column **Orig.**), and with the use of energy macro-modeling to speedup SOC power estimation (major column **Caching**). Sub-columns **Energy** and **CPU time** present the total system energy consumption and simulation time on a Sun Ultra Enterprise 450 workstation with 256MB of memory. The speedup ($\frac{CPU\ time\ for\ Orig.\ case}{CPU\ time\ for\ Macromodeling\ case}$) is presented in the sixth column, while the absolute $\%$ error in the energy estimate of the macro-modeling approach with respect to the base case (**Orig.**) is presented in the last column. As before, the rows represent different implementations of the TCP/IP subsystem with different values of the bus DMA size.

Table 2: Speedup and accuracy of the macro-modeling approach

DMA size	Orig.		Macromodeling		Speedup	Error $\%$
	Energy (mJ)	CPU time (s)	Energy (mJ)	CPU time (s)		
2	0.54	8051.52	0.72	92.44	87.1	32.9
4	0.44	4023.36	0.56	63.46	63.4	27.4
8	0.39	2080.77	0.48	48.73	42.7	23.7
16	0.36	1398.49	0.44	41.08	34.0	21.6
32	0.35	852.25	0.42	37.71	22.6	20.4
64	0.34	680.78	0.41	36.02	18.9	19.6

The tables indicate that:

- The acceleration technique based on caching results in simulation speedups of between **8.6X** and **18.8X** (average of **13X** compared to the base case without any acceleration technique.
- The macro-modeling based acceleration technique results in speedups of between **18.9X** and **87.1X** (average of **44.8X**) with respect to the base case, with an average error around 24.3% in the estimated system energy dissipation. The results with macro-modeling are conservative (over-estimate power consumption), especially for the software parts of the system. That is because they apply an additive model to compute the energy consumed by the execution of a sequence of CFSM paths, and hence do not take into account the effects such as the partial overlap their execution due to pipelining. The level of pessimism introduced, in general, depends on the processor architecture, and the granularity/size of the CFSM paths. However, as shown below, our experiments indicate that the results of the macro-modeling approach do have high *relative accuracy* ("tracking fidelity"), since they result in the same ranking of the different candidate sizes with respect to power consumption.

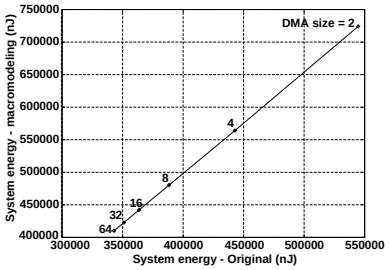


Figure 6: Relative accuracy of energy macro-modeling in predicting variation of system energy with varying DMA size

The relative accuracy of an estimation technique refers to how accurately it compares/ranks different variants of a design. In order to study the relative accuracy of SOC power estimation with energy macro-modeling, we considered the various implementations of the TCP/IP subsystem implementation corresponding to varying bus DMA size. Figure 6 shows a plot of the energy estimate for each system configuration as estimated by our SOC power estimation framework when macro-modeling was employed for speedup *vs.* the energy estimate obtained using the vanilla SOC power estimation framework with no speedup technique. The plot indicates that SOC power estimation with macro-modeling preserve the ranking of the different system configurations with respect to power. An additional desirable trend that can be observed is that a linear relationship exists between the energy reported by SOC power estimation with macro-modeling and SOC power estimation without any acceleration technique. We have obtained similar results in various other experiments (*e.g.* by attempting to rank several different HW/SW partitions).

5.3 Exploring SOC communication architecture tradeoffs

In order to demonstrate the utility of our power co-estimation tool in an iterative design exploration framework, we performed the following experiment. We ran an exhaustive search of all possible meaningful assignments of priority and DMA sizes for the TCP/IP example, invoking the power analysis technique for each configuration. Overall, there 6 priority assignments and 7 possible DMA sizes, leading to a total of 48 points in the design space. Figure 7 shows the variation of energy consumed in the TCP/IP system for processing 3 network packets. The main parameter values that have been used during power estimation are: Voltage supply (V_{dd}) = 3.3 V, Bus Line capacitance per bit (C_{bit}) = 10 nF, Address bus width = 8 bits, Data bus width = 8 bits. The minimum energy point was found to be when the DMA size is set to 128, and the priorities are assigned so that $Create\_Pack$, IP_Check, and $Checksum$ are in descending order of priority. The entire design space exploration took about 180 minutes on a Sun Ultra Enterprise 450 workstation with 256MB RAM.

Note that the SOC integration architecture significantly affects the power consumption in the entire system. Changing the DMA size affects the HW power and SW power even though the HW and SW parts are unchanged. Such effects are not obvious to evaluate without a power co-estimation tool such as ours. Another useful application of our environment is that it can highlight peak periods in power consumption, and correlate functional information with power information. For example in our case we observed during the experiment with the TCP/IP example that the peaks in power consumption are associated with the points in time when the modules handshake with the arbiter.

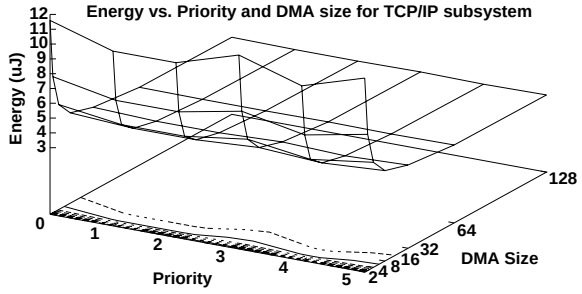


Figure 7: Efficient exploration of the communication architecture design space for the TCP/IP system using our co-estimation tool

6 Conclusions and Future Work

We have presented a power co-estimation framework for System-on-Chip (SOC) designs, based on a concurrent and synchronized execution of power estimators for the different components of the SOC. We have demonstrated the necessity of co-estimation in order to capture the timing and power inter-dependencies at the system level. In order to facilitate efficient SOC power estimation, we proposed acceleration techniques based on energy and delay caching and macro-modeling, and statistical sampling. Our experiments demonstrate the utility of our SOC power estimation techniques in a system-level design framework, and the accuracy/efficiency tradeoffs offered by the acceleration techniques.

References

[1] A. R. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Publishers, Norwell, MA, 1995.

[2] J. Rabaey and M. Pedram (Editors), *Low Power Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, 1996.

[3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, 1997.

[4] J. Monteiro and S. Devadas, *Computer-Aided Design Techniques for Low Power Sequential Logic Circuits*. Kluwer Academic Publishers, Norwell, MA, 1996.

[5] E. Macii, M. Pedram, and F. Somenzi, "High-level power modeling, estimation, and optimization," in *Proc. Design Automation Conf.*, pp. 504–511, June 1997.

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

[7] T. Sato, Y. Ootaguro, M. Nagamatsu, and H. Tago, "Evaluation of architecture-level power estimation for CMOS RISC processors," in *Proc. Symp. Low Power Electronics*, pp. 44–45, Oct. 1995.

[8] L. Benini and G. De Micheli, "System-level power optimization: techniques and tools," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 1999.

[9] D. Kirkovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," in *Proc. Design Automation Conf.*, pp. 697–702, June 1997.

[10] B. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of embedded systems," in *Proc. Design Automation Conf.*, pp. 703–708, June 1997.

[11] Y. Li and J. Henkel, "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems," in *Proc. Design Automation Conf.*, pp. 188–193, June 1998.

[12] J. Henkel, "A low power hardware/software partitioning approach for core-based embedded systems," in *Proc. Design Automation Conf.*, pp. 122–127, June 1999.

[13] T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," in *Proc. Design Automation Conf.*, pp. 867–872, June 1999.

[14] J. Rowson, "Hardware/software co-simulation," in *Proc. Design Automation Conf.*, pp. 439–440, June 1994.

[15] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," in *IEEE International Conference on Computer Design*, pp. 328–333, October 1992.

[16] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jureska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, Norwell, MA., 1997.

[17] J. Buck, S. Ha, E. Lee, and D. Masserchmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal on Computer Simulation Special Issue on Simulation Software Management*, Jan. 1994.

[18] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.

[19] M. Lajolo, L. Lavagno, and A. Sangiovanni-Vincentelli, "Fast instruction cache simulation strategies in a hardware/software co-design environment," in *Proc. Asia and South Pacific Design Automation Conf.*, Jan. 1999.

[20] J. Liu, M. Lajolo, and A. Sangiovanni-Vincentelli, "Software timing analysis using hw/sw cosimulation and instruction set simulator," in *Proc. Int. Workshop on Hardware/Software Codesign*, Mar. 1998.

[21] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, and A. Sangiovanni-Vincentelli, "Modeling shared memory access effects during performance analysis of hw/sw systems," in *Proc. Int. Workshop on Hardware/Software Codesign*, Mar. 1998.

[22] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers - Principles, Techniques and Tool*. Addison-Wesley, 1986.

[23] C. T. Hsieh, M. Pedram, G. Mehta, and F. Rastgar, "Profile-driven program synthesis for evaluation of system power dissipation," in *Proc. Design Automation Conf.*, pp. 576–581, June 1997.