Fast System-Level Power Profiling for Battery-Efficient System Design *

Kanishka Lahiri
Dept. of ECE
UC San Diego
klahiri@ece.ucsd.edu

Anand Raghunathan
C&C Research Labs
NEC USA
anand@nec-lab.com

Sujit Dey
Dept. of ECE
UC San Diego
dey@ece.ucsd.edu

ABSTRACT

An increasing disparity between the energy requirements of portable electronic devices and available battery capacities is driving the development of new design methodologies for battery-efficient systems. A crucial requirement for battery efficient system design is to be able to efficiently and accurately estimate battery life for candidate system architectures. Recently, efficient techniques have been developed to estimate battery life under given profiles of system power consumption over time. However, techniques for generating the power profiles themselves are either too cumbersome for system level exploration, or too inaccurate for battery life estimation.

In this paper, we present a new methodology for efficiently and accurately generating power profiles for different system-level architectures. The designer can specify the manner in which (i) system tasks are mapped to a set of available implementations, and (ii) system communications are mapped to a specified communication architecture. For a given architecture, a power profile is automatically generated by analyzing an abstract representation of the system execution traces, while taking into account the selected implementations of the system's computations and communications.

Experiments conducted on the design of an IEEE 802.11 MAC processor indicate that the power profiling approach offers run times that are several orders of magnitude lower than a simulation based power profiling technique, while sustaining negligible loss of accuracy (average profiling error was observed to be less than 3.4%).

1. INTRODUCTION

Advances in wireless communication and semiconductor technologies are contributing to the rapid growth of the portable electronics market. However, projections of energy demands of such devices substantially exceed the capacity of the batteries that power them, necessitating the deployment of new architectures and design methodologies for battery-efficient systems [1]. While research on low power design [2]-[5] has largely been motivated by the need to improve battery life, it has mainly focussed on minimizing average power consumption. However, it has been shown that minimizing average power (or total energy) does not necessarily maximize battery life. That is because, in practice, the amount of energy delivered by a battery can vary significantly, depending on the profile of the load system's power consumption over time [6]-[9].

An important aspect of battery-efficient system design is the availability of automatic tools to estimate battery life at different points in the system design space. Since most existing techniques for battery life estimation are based on analyzing battery discharge under a given system power consumption profile, evaluation of the battery efficiency of a candidate architecture is a two step process. In the first step, the power profile of the architecture is generated under a typical workload, and in the second step, battery life estimation is performed using a suitable battery model. While research in battery modeling and analysis addresses the second step [6]-[9], the focus of this paper is on the first, *i.e.*, efficient and accurate generation of power profiles for candidate HW/SW system architectures.

*This work was supported by the National Science Foundation, grant 99-12414

1.1 Related Work

A large body of work exists on analyzing power consumption at the circuit, logic, register-transfer and architectural levels [2, 3, 4]. Recent work in system-level power estimation has yielded (i) techniques for estimating the power consumption of individual system components, such as embedded CPUs, the memory hierarchy, system busses, peripheral components (*e.g.*, [10]-[13]), and (ii) techniques for estimating the power consumption of the system as a whole. The latter set of techniques can be categorized as follows. The first category consists of techniques that statically characterize the power consumption of system tasks, using simple power models for alternate implementations, and inter-task communication (*e.g.*, [14, 15]). The drawback is that they often assume statically scheduled systems, and hence do not accurately account for dynamic effects (*e.g.*, bus conflicts, cache misses *etc*). The second category includes techniques based on system simulation, with power models for individual components, memories, and system busses [16, 17, 18]. These techniques can demonstrate high accuracy, but are computationally expensive, making them unsuitable for system-level design space exploration. Raising the level of abstraction improves efficiency, but sacrifices accuracy.

Our work belongs to a third category, and is based on a two step hybrid methodology. Our technique derives accuracy from a one-time detailed simulation and power analysis (step 1), and computational efficiency from fast processing of abstract execution traces (step 2). It bears mentioning that the approach described in [13] also analyzes data collected from execution traces using analytical equations. While this approach can accurately estimate average power, it is not well suited for generating power profiles over time. In addition, their work applies to tuning of component parameters, and assumes that the mapping of the system's functionality and communications to the architecture is fixed.

1.2 Paper Overview and Contributions

In this paper, we present a fast power profiling methodology to drive the design of battery-efficient systems. Power profiling refers to the analysis of system power consumption over time (as opposed to average power or total energy estimation), and is required for battery life estimation [6]-[9]. Our methodology provides the system designer with a framework to generate the system power profile while varying the system architecture along two broad dimensions. In our methodology, the designer can: (i) select different implementations for system tasks, varying how the system's *computations* are mapped to system components, and (ii) select different system-level communication architectures, and experiment with alternative mappings of the system's *communications*.

Our methodology consists of two pre-processing phases (each of which is performed only once), followed by a fast power profiling phase (which is performed for each candidate system architecture).

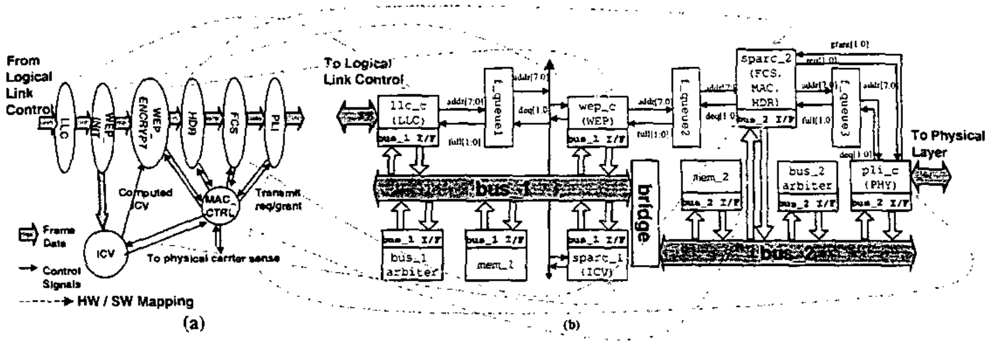

Figure 1: (a) Functional view of the 802.11 MAC processor system (b) Candidate HW/SW Architecture

In the first phase, execution traces from HW/SW co-simulation are abstracted into a specialized graph. In the second phase, basic computation blocks and system-level communications are characterized under potential implementations using detailed power models. In the third phase, the power profiler processes the graph to generate the system power profile, while taking into account the mapping of computation tasks, inter-task communications, and dynamic effects of the communication architecture. The profiler is efficient, demonstrating several orders of magnitude improvement over simulation based techniques, while providing comparable profiling accuracy (average error observed in our experiments was 3.4%).

In the next section, we present examples that motivate the proposed power profiling methodology, and illustrate the benefits it provides. In Section 3, we present the power profiling methodology, including details of the various steps. In Section 4, we present experiments conducted on several candidate architectures for an IEEE 802.11 MAC processor system, and compare the proposed techniques with HW/SW co-simulation in terms of efficiency and accuracy.

2. MOTIVATION

In this section, we motivate the need for fast power profiling techniques using the design of an IEEE 802.11 MAC processor as an example.

2.1 IEEE 802.11 MAC Processor System

The specification of the IEEE 802.11 MAC processor [19] consists of a set of communicating tasks, shown in Figure 1(a) (details are available in [19]). The LLC task receives MAC frames from the Logical Link Control Layer, and stores them in the system memory. For each frame, the WEP task encrypts frame data, while the ICV task computes an Integrity Checksum Vector. The HDR task generates the MAC header. The FCS task computes a CRC-32 checksum over the encrypted frame and header. MAC_CTRL implements the CSMA/CA algorithm, and signals the PLI task when it is "okay to transmit". PLI retrieves encrypted frames from the MAC memory, and writes them on to the Physical Layer Interface.

An architecture implementing these tasks is shown in Figure 1(b), which consists of two programmable SPARCLite cores [20], 3 HW co-processors, 2 memories, and queue buffers. The LLC, WEP and PLI tasks are each mapped to dedicated HW units. The ICV task is mapped to an embedded SPARCLite core. A second embedded CPU implements the HDR, FCS, and MAC_CTRL tasks. The system components are connected through a communication architecture consisting of two busses connected by a bridge.

2.2 Impact of Architecture on Battery Life

In this example, we consider two variations of the architecture presented in Figure 1(b). In the first variation (case 1), ICV is implemented as embedded software while FCS is implemented using dedicated HW. In the second variation (case 2), the mapping is swapped: FCS is implemented as embedded software, while ICV is implemented using dedicated HW. All other task and communication mappings are left unchanged.

We first measured the average power consumed by each of these architectures under typical workloads using a HW/SW power co-estimation technique [18]. Cases 1 and 2 were found to consume (on average) 596.2 *mW* and 590.7 *mW*, respectively. Thus, the two architectures are equivalent from the point of view of average power or total energy consumption. Next, we measured the battery life for each of the two architectures using a stochastic model of a lithium-ion battery [9]. Battery life measurements for the two cases were found to be 3673 seconds and 4783 seconds respectively, a difference of 30%.

The example highlights that (i) average power is not a good metric for designing battery-efficient systems, and (ii) the system architecture (in this example, the mapping of tasks to components) can have a significant impact on battery life. This motivates the need for techniques to enable automatic and efficient battery life estimation for different system-level architectures.

2.3 Importance of System Power Profiling

Batteries have been found to exhibit rate capacity phenomena, which result in low efficiencies during periods when the load current (and hence, the system power consumption) exceeds a specified rated value [6]. This phenomenon explains the discrepancy between the battery efficiencies of the two architectures considered above. In this system, violations of the rated current can potentially arise when the ICV computations and WEP computations proceed in parallel. That is because, at the times at which these computations overlap, they contribute to an increase in overall system power consumption, causing violation of the battery's rated current. When the ICV task is mapped to SW (case 1), the duration and extent of rated current violations are greater than when it is mapped to HW (case 2). Hence, case 2, which has a more "battery friendly" power profile (relative to case 1), leads to more efficient battery discharge.

The example shows that the system architecture can influence the power profile in a way that significantly impacts battery life. This highlights the importance of accurately generating system power profiles in order to support battery-efficient system design.

2.4 Evaluation of System-level Power Profiles

In the above example, we considered only two architectures from a large design space consisting of many alternative mappings of tasks to components, and communication architectures. In reality, to arrive at the most battery-efficient design, a more thorough exploration of the design space is desirable.

The time taken by the HW/SW power co-estimation tool to generate power profiles for each architecture of the MAC processor (even with speedup techniques such as those described in [18]), was in excess of 3 hours, for an input trace consisting of 11 MAC frames. Clearly, the computational requirements of such techniques make them unsuitable for exploring a potentially large design space.

To illustrate the advantage of using the proposed power profiling technique, we used our profiling tool iteratively to evaluate a total of 10 different architectures, including those described earlier (Section 2.2). The total time taken by the tool was 3.14 *seconds*, indicating the high computational efficiency of our approach as compared to co-simulation based power estimation.

3. POWER PROFILING METHODOLOGY

In this section, we first present the overall power profiling methodology, highlighting the various steps. Next, we present detailed description of key steps, including the related algorithms and tools employed.

3.1 Overview

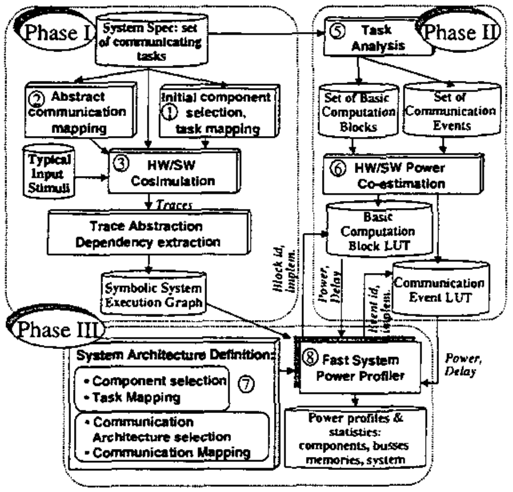
Phase I
System Spec: set of communicating tasks
② Abstract communication mapping
Initial component ① selection, task mapping
Typical Input Stimuli
③ HW/SW Cosimulation
Traces
Trace Abstraction Dependency extraction
Symbolic System Execution Graph
Phase II
⑤ Task Analysis
Set of Basic Computation Blocks
Set of Communication Events
⑥ HW/SW Power Co-estimation
Basic Computation Block LUT
Communication Event LUT
Block id, implem.
Power, Delay
Event id, implem.
Phase III
System Architecture Definition:
• Component selection
• Task Mapping
⑦
• Communication Architecture selection
• Communication Mapping
⑧ Fast System Power Profiler
Power, Delay
Power profiles & statistics: components, busses memories, system

Figure 2: Fast power profiling methodology

The inputs to the power profiling methodology are highlighted in Figure 2. They include:

- The system specification, comprising a set of communicating tasks, each of which can potentially be mapped to an embedded CPU (software) or dedicated hardware;
- Typical input stimuli, or a simulation model of the system's operating environment (testbench);
- System architecture definition, which includes (i) the mapping of system tasks to a set of selected architectural components, and (ii) the mapping of inter-component communications to a specified system-level communication architecture;

The output of the methodology is the power consumption profile of the system under the selected system architecture, system-level power consumption statistics, including individual power profiles of each component, and their relative contribution to total system power.

The methodology in Figure 2 has three phases, labeled accordingly. In Phase I, HW/SW co-simulation of the system specification is performed with provided input stimuli (Step 3), and an initial mapping of system tasks to selected components (Step 1). During this initial co-simulation, communication between components is modeled in an abstract manner (Step 2) (*e.g.*, through the exchange of abstract communication events). While the resulting system execution trace is "timing-inaccurate" (owing to the abstract modeling of communication), it contains critical information with respect to dynamic communication and computation dependencies. However, these traces are also quite large, containing significant detail, a lot of which is irrelevant for our purpose. Hence, in Step 4, information that is *necessary and sufficient* for subsequent analysis is extracted from the traces, and captured in a *Symbolic System Execution Graph* (henceforth called the SYM-GRAPH). The SYM-GRAPH is an abstract and compact representation of the system's execution, which includes essential information about the system's computations, communications, and their inter-dependencies, both within and across tasks. This step is critical to enabling highly efficient analysis without compromising accuracy.

In Phase II, each system task is decomposed into a set of basic computation blocks and communication events (Step 5). In Step 6, for each computation block and communication event identified in the previous step, power consumption and execution time under potential implementations are analyzed using detailed simulation based performance analysis and power estimation techniques. The results are stored in lookup tables in the form of absolute data, or simple parameterized expressions.

Phase III consists of the fast power profiler (Step 8). The power profiler takes as input (i) the SYM-GRAPH generated in Phase I, (ii) a definition of the system architecture (Step 7), and (iii) the lookup tables generated in Phase II. The power profiling technique consists of algorithms that efficiently process the SYM-GRAPH to take into account the effects of the selected task and communication mappings, including dynamic effects such as bus contention and inter-component synchronization. These algorithms result in (i) creating/splitting SYM-GRAPH vertices, (ii) modifying time-stamps of SYM-GRAPH vertices, and (iii) annotating SYM-GRAPH vertices with power estimates. The resulting SYM-GRAPH is an abstract yet accurate representation of the system's execution under the selected architecture, from which the system power profile and other useful power statistics are easily derived.

Note that, in order to obtain power profiles for different architectures, one merely needs to repeatedly execute Phase III. The advantage is that the underlying algorithm operates on an abstract graph, rather than detailed traces. Hence, it is orders of magnitude faster than a complete simulation of the system. Moreover, it is accurate, because (i) the SYM-GRAPH is annotated with power and timing information gathered from detailed simulation and power estimation, and (ii) the processing algorithm accurately incorporates effects of different computation/communication mappings.

3.2 Power Profiling Methodology Details

In this subsection, we present details of certain key steps in the methodology of Figure 2.

3.2.1 Task Analysis (Step 5)

In this step, from the control flow graph of each task T, we identify the set of constituent basic computation blocks $B_T = \{b_1, b_2, \dots, b_n\}$. A basic computation block b_i is a straight line segment of the task specification that can be entered only at the beginning, and exited only at the end. Note that b_i cannot have embedded conditionals, loops, or inter-task communication events emanating from, or terminating within its body. When an architecture is defined, task T, and consequently all the elements of B_T, are mapped to either an embedded CPU, or dedicated hardware. [1]

Next, we identify the set of system-level communication events $C_T = \{c_1, c_2, \dots, c_m\}$ initiated by task T. A communication event c_i is any data transfer or control signal that results in an access to system-level communication resources. Different $c_i \in C_T$ could potentially be mapped to different communication resources. For example, all control signals generated by a component could be

[1] The assumption that the granularity of partitioning and mapping is at the task level does not restrict the methodology in any way, since tasks can be easily broken down into multiple tasks to obtain more fine-grained mapping control.

mapped to a point-to-point communication channel, while data transfers could be mapped to a high bandwidth bus.

3.2.2 *Lookup Table Construction (Step 6)*

In this step, detailed simulation based performance analysis and power estimation is carried out for the basic computation blocks and communication events identified in the previous step, for all target implementations. In our work, we use the framework presented in [18] for this step.

Each basic computation block of a task is exercised with a pseudo-exhaustive set of inputs to derive its average execution time and average power consumption under each implementation. The results are used to populate a lookup table (LUT), which is indexed by a basic computation block identifier and implementation choice.

For each communication event, the average execution time and power consumption per bus word is measured using simulation and power estimation of the associated master and slave interfaces, and the bus itself. Note that, typically bus line capacitance estimates are not available until components have been selected and an initial floorplan has been generated. Hence, during this step, power consumption for communication events under a target bus is calculated assuming nominal values for bus line capacitances. Once the system architecture is defined, the designer specifies actual line capacitances, (*e.g.*, from a system floorplan), which are used to appropriately scale the power consumption of each communication event.

Next we describe Phase III of the methodology presented in Figure 2, where the role of the LUTs will be made clear.

3.2.3 SYM-GRAPH *Construction (Step 4)*

The basic approach of constructing the SYM-GRAPH is extended from [21], which focussed solely on the effects of communication architecture design. We briefly describe the procedure here, highlighting enhancements made in order to support power profiling, and task mapping.

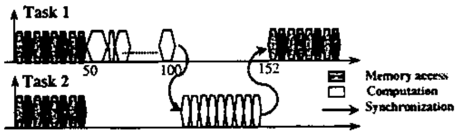

Figure 3: Segment of a detailed HW/SW co-simulation trace

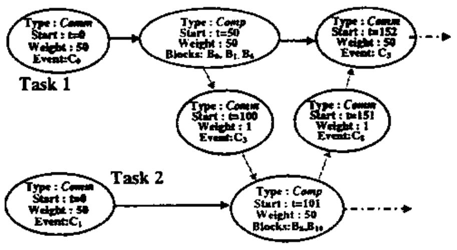

Figure 4: Segment of corresponding SYM-GRAPH

To construct the SYM-GRAPH, the execution trace of each task (obtained via HW/SW co-simulation in Step 3), is analyzed. Contiguous computations are grouped into computation vertices to represent specific control flow paths in the task that do not contain any system-level communication events. Each computation vertex is labeled with the sequence of basic computation blocks it represents. Next, the procedure groups contiguous system-level communications that correspond to a single communication event (*e.g.*, burst data transfers, inter-task synchronization) into a single communication vertex. Each communication vertex is labeled with a communication event identifier. Edges are inserted in the graph to capture all the run-time control flow dependencies. An example system execution trace and its corresponding SYM-GRAPH are shown in Figures 3 and 4, respectively.

3.2.4 *System Architecture Definition (Step 7)*

The system architecture definition includes:

- A mapping from the set of system tasks to a set of selected system components;
- A specification of the communication architecture topology, which could range from a shared bus to an arbitrary network of busses / dedicated channels interconnected by bridges;
- Line capacitances of each bus;
- Mapping of system communications to communication paths in the selected topology;
- Communication protocol parameters (bus priorities, burst sizes, *etc.*)

3.2.5 *Power Profiling Algorithm (Step 8)*

In this step, the power profiling tool manipulates the SYM-GRAPH generated in Step 4, based on information obtained from the system architecture definition (Step 7) and the delay and power LUTs (Step 5). The algorithm adds new vertices, splits existing vertices, and scales the execution times and power consumption of vertices in the SYM-GRAPH in order to take into account task and communication mappings, as well as dynamic effects introduced by the communication architecture.

```
Procedure pprof
inputs:    SYM-GRAPH $G'$, Task Mapping, Communication
Architecture Specification, Communication
Mapping, Delay and Power Lookup Tables
outputs: SYM-GRAPH $G'$, Power profile, statistics
initial
        Initialize queues : $Q_C, Q_{B_1}, Q_{B_2}, \cdots$ ;
        /* $Q_C$: global queue of vertices, sorted by vertex start times */
        /* $Q_{B_i}$: pending communication vertices mapped to bus $B_i$*/
        /* All queues initialized with pending vertices at time t=0 */
begin
    do
            $v := serve\_ready\_comp(Q_C)$;
            $v.delay := get\_delay(v)$;
            $v.power := get\_power(v)$;
            $execute\_comp(v)$;
            $add\_enabled\_vertices(v)$;
            for each bus $B_i$:
                $w := serve\_ready\_comm(Q_{B_i})$;
                $w.delay := get\_delay(w)$;
                $w.power := get\_power(w)$;
                $execute\_comm(w)$;
                $add\_enabled\_vertices(v)$;
            end for
    until no more ready vertices
    $generate\_profiles()$;
end
```

Figure 5: Power profiling algorithm pprof

The algorithm pprof (Figure 5) maintains a queue Q_C, containing "ready" vertices sorted by time-stamp, which include computation vertices from various system tasks. Additionally, for each communication channel (or bus) B_i, the algorithm maintains a queue Q_{B_i} of ready communication vertices. The procedure performs the following actions in a loop.

On dequeuing a computation vertex v, the call to $get\_delay(v)$ consults the LUTs to compute $delay(v)$, the execution time of v. This is given by $\sum_{b_i \in v} delay(b_i)$ where b_i is a basic computation block associated with v. Similarly, $get\_power(v)$ returns $power(v)$, which is given by $\frac{\sum_{b_i \in v} power(b_i) * delay(b_i)}{\sum_{b_i \in v} delay(b_i)}$.

For each system bus (or communication channel) B_i, the corresponding queue Q_{B_i} is serviced in accordance with B_i's protocol. Dequeued communication vertices are assigned execution times and power values using similar lookup based techniques (or simple calculations using supplied capacitance values). Communication vertices are then processed by *execute_comm*(), a procedure that transforms the SYM-GRAPH to incorporate the effects of the communication architecture [21]. While we do not go into the details of the procedure here, it bears mentioning that its execution may result in splitting of some communication vertices into multiple vertices (due to maximum burst transfer sizes set by the bus protocol), or creation of new vertices (due to handshaking defined by the bus protocol). Clearly, this affects the sequencing of data on the bus, leading to potential inaccuracies in the power profile of the bus at communication vertex boundaries. However, in our experiments we observed this to have an insignificant impact on the system power profile, and hence do not consider it to be a serious issue.

The procedure *add_enabled_vertices*(v) assigns time-stamps to vertices that become ready as a result of executing v, and populates the appropriate queues. The algorithm terminates when all the queues are empty.

4. EXPERIMENTS

In this section, we describe experiments that study the accuracy and efficiency of the proposed methodology in generating power profiles of alternate architectures of the IEEE 802.11 MAC processor described in Section 2. We first describe the design space of system architectures that we considered. Next, we describe the methodology used to conduct experiments. Finally, we present and discuss the results of our experiments.

4.1 Design Space for the MAC Processor

For system tasks, we considered two implementations for each of the WEP, ICV, and FCS tasks, namely synthesized hardware, and software running on the SPARCLite processor. The mapping of the remaining tasks was kept fixed, as in Figure 1(b). Two different communication architectures were considered for mapping system communications. The first (shown in Figure 1(b)) is a high-performance architecture consisting of two busses connected by a bridge. In this case, all data transfers are mapped either to (i) the bus on which the initiating component resides, or to (ii) a path comprising two busses. In the second architecture, the two busses are replaced by a single priority based shared system bus. In both cases, control signals are mapped to dedicated communication channels (as shown in Figure 1(b)). In all we considered 10 candidate architectures, and applied our profiling technique to each.

4.2 Experimental Methodology

The MAC processor was designed in the POLIS [22] environment using a combination of Esterel [23] and C to specify the system tasks, and PTOLEMY [24] for system level simulation. A modified version of the power co-estimation technique of [18] was used to generate power profiles for 10 candidate system architectures. These simulations were driven by actual 802.11 workloads obtained by running the ETHEREAL packet capture software [25] on a laptop PC connected to a wireless LAN while receiving video data. Next, we applied our profiling tool to generate power profiles for the same architectures and input stimuli. The results of various experiments conducted are described next.

4.3 Average Power Estimation

The aim of the first set of experiments was to evaluate the accuracy of the proposed power profiler in estimating average power. Table 1 reports on the results of conducting experiments on 10 different architectures for the MAC processor. Column 1 describes the architecture under evaluation. Column 2 reports the average power estimated using co-simulation based power estimation [18], and Column 3 reports the average power as estimated using our power profiling technique. Column 4 reports the error between the two estimates. The results in Table 1 indicate that the average error with respect to power co-estimation was only 0.38%, and demonstrate that the proposed methodology is capable of measuring average system power with a very high degree of accuracy.

4.4 Profiling Accuracy

In the next experiment, we compared the system power consumption profile over time, $p(t)$, generated by our power profiling technique, with $p_{sim}(t)$, generated using HW/SW co-estimation. To do this, both $p(t)$ and $p_{sim}(t)$ were averaged over a constant size window, to obtain a set of discrete points t_i, at intervals of $\tau = 0.5$ *ms*. We define the absolute error at time t_i to be $\varepsilon(t_i) = | p(t_i) - p_{sim}(t_i) |$, and the profiling error at time t_i to be $\varepsilon_P(t_i) = \frac{|p(t_i) - p_{sim}(t_i)|}{p_{sim}(t_i)} \times 100$.

Columns 5 and 6 of Table 1 report on the profiling accuracy of our technique. Column 5 indicates the the average absolute error (averaged over the length of the trace) $\bar{\varepsilon}$, while Column 6 indicates the average profiling error $\bar{\varepsilon}_P$. Note that, since we take the modulus in computing both $\varepsilon(t_i)$ and $\varepsilon_P(t_i)$, positive and negative errors will not cancel out. From the table it is clear that the power profile $p(t)$ closely tracks the profile $p_{sim}(t)$. Over all 10 architectures, the average profiling error is 3.8%.

To investigate how profiling errors of different magnitude contribute to the average, we plotted the frequency distribution of the profiling error $\varepsilon_P(t_i)$ for one of the candidate architectures (Figure 6). We observe that for all the time intervals, the errors are below 20%, while 98% of the time, they are below 8%. This shows that not only does the profile $p(t)$ track $p_{sim}(t)$ with low average error, the deviation of profiling error from the mean is also quite reasonable.

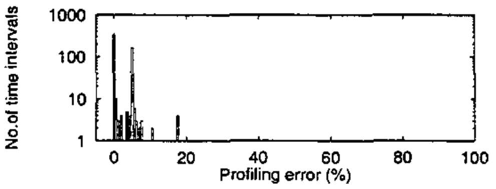

Figure 6: Distribution of the profiling error ($\tau = 0.5$ *msec*) in the obtained power profile of a candidate architecture

4.5 Effect of Profiling Granularity

In this experiment, we illustrate how the average profiling error $\bar{\varepsilon}_P$ varies with increasing profile granularity. Figure 7 illustrates this dependence for one of the candidate architectures. τ is varied from 0.5 *ms*, at which the error is maximum (4.73%), up to the length of the entire profile, when the error is minimum (0.49%). This shows that the power profiling technique yields even higher accuracy for coarser grained profiles. This matches requirements from battery life estimation, where large internal time constants of batteries and power supply circuitry often make fine-grained profiles redundant.

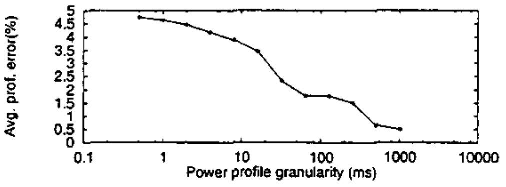

Figure 7: Effect of granularity on average profiling error

4.6 Computational Efficiency

Finally, we report on the computational efficiency of the proposed methodology. In Table 1, Column 7 reports on the elapsed

Table 1: Results from power profiling of candidate architectures for the IEEE 80 2.11 MAC processor

Architecture	*Average Power*			*Power Profiling*		*CPU time*
	HW/SW co-estimation (mW)	Trace based profiling (mW)	Relative error (%)	Avg. absolute error (mW)	Avg. relative absolute error (%)	Trace Based Profiling (sec)
All HW, Single bus	380.7	380.8	0.03	1.8	0.4	0.21
All HW, Two busses	418.2	414.9	0.80	4.1	2.2	0.36
ICV in SW, Single bus	544.2	543.0	0.18	24.0	4.1	0.26
ICV in SW, Two busses	596.2	592.9	0.55	13.0	3.4	0.42
WEP in SW, One bus	613.8	614.0	0.16	45.0	7.3	0.25
WEP in SW, Two busses	628.3	625.5	0.45	41.8	6.2	0.37
FCS in SW, Single bus	517.2	512.2	0.97	19.0	3.2	0.22
FCS in SW, Two busses	590.7	587.4	0.22	4.6	2.2	0.37
FCS, ICV in SW, Single bus	672.2	672.2	0.00	26.3	3.5	0.26
FCS, ICV in SW, Two busses	766.8	763.5	0.43	23.3	4.3	0.42

time in running HW/SW power co-estimation for each specified architecture. Column 8 indicates the elapsed time in generating power profiles for each of the candidate architectures using the proposed power profiler. Measurements of elapsed time were made on a 400*Mhz* Sun Ultra-II workstation with 128 *MB* RAM. Clearly, CPU times of the power profiling tool are significantly superior to the co-estimation approach. On average, the time taken to obtain the power profile for each architecture is 0.31 *s*. In comparison, the time required to generate a system power profile for each architecture using HW/SW co-simulation based power profiling exceeded 3 hours. This indicates that using our approach yields CPU time improvements of 4 orders of magnitude compared to power profiling based on co-simulation. While the time consumed in the pre-processing phases of the proposed methodology were significant (4-5 hours), it should be noted that this is a non-recurring cost. We expect that for systems with complex design spaces, amortization of this cost over a large number of candidate architectures will result in significantly superior efficiency.

5. CONCLUSIONS

In this paper, we presented a new methodology for efficiently and accurately generating power profiles for different system-level architectures. We conclude that using an abstract representation of system execution traces can provide for several orders of magnitude efficiency improvements over simulation based power profiling techniques. Additionally, accurate pre-characterization, coupled with careful incorporation of interactions between components, results in negligible loss of accuracy in the power profiles. We believe that our approach, coupled with efficient battery modeling techniques, will be an invaluable aid in designing battery-efficient systems.

6. REFERENCES

[1] K. Lahiri, A. Raghunathan, S. Dey, and D. Panigrahi, "Battery driven system design: a new frontier in low power design," in *Proc. ASP-DAC/Int. Conf. VLSI Design*, pp. 261–267, Jan. 2002.

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

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

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

[5] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer, 1998.

[6] M. Doyle, T. F. Fuller, and J. S. Newman, "Modeling of galvanostatic charge and discharge of lithium/polymer/insertion cell," *J. Electrochem. Soc*, vol. 140, pp. 1526–1533, June 1993.

[7] S. Gold, "A PSPICE macromodel for lithium-ion batteries," in *Proc. Annual Battery Conference on Applications and Advances*, pp. 9–15, 1997.

[8] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "A discrete-time battery model for high-level power estimation," in *Proc. Design Automation & Test Europe (DATE) Conf.*, pp. 35–39, Mar. 2000.

[9] D. Panigrahi, C. F. Chiasserini, S. Dey, R. R. Rao, A. Raghunathan, and K. Lahiri, "Battery life estimation for mobile embedded systems," in *Proc. Int. Conf. VLSI Design*, pp. 55–63, Jan. 2001.

[10] 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.

[11] M.B.Kamble and K.Ghose, "Analytical models for energy dissipation in low power caches," in *Proc. Int. Symp. Low Power Electronics & Design*, pp. 143–148, 1997.

[12] W. Fornaciari, D. Sciuto, and C. Silvano, "Power estimation for architectural exploration of HW/SW communication on system-level buses," in *Proc. Int. Symposium on Hardware/Software Codesign*, pp. 152–256, May 1999.

[13] T. D. Givargis, F. Vahid, and J. Henkel, "Trace-driven system-level power evaluation of system-on-a-chip peripheral cores," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 306–311, Jan. 2001.

[14] 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.

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

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

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

[18] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Efficient Power Co-Estimation Techniques for System-on-Chip Design," in *Proc. Design Automation & Test Europe (DATE) Conf.*, pp. 27–34, Mar. 2000.

[19] "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications." IEEE Computer Society LAN MAN Standards Committee, IEEE Std 802.11-1999 Edition.

[20] Fujitsu Microelectronics. Inc., "Sparclite series MB86934 addendum." 1994.

[21] K.Lahiri, A.Raghunathan, and S.Dey, "System level performance analysis for designing on-chip communication architectures," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 768–783, June 2001.

[22] F. Balarin, *et.al.*, *Hardware-software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, Norwell, MA, 1997.

[23] "Esterel: a Synchronous Reactive Programming Language." http://www.esterel.org.

[24] J. Buck, *et.al.*, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. Journal on Computer Simulation, Special Issue on Simulation Software Management*, vol. 4, pp. 155–182, Apr. 1994.

[25] "Ethereal 0.8.18." http://www.ethereal.com.