PrEsto: An FPGA-Accelerated Power Estimation Methodology for Complex Systems

Dam Sunwoo, Gene Y. Wu, Nikhil A. Patil and Derek Chiou
Department of Electrical and Computer Engineering
The University of Texas of Austin
{sunwoo, wu, npatil, derek}@ece.utexas.edu

Abstract—**Reduced or bounded power consumption has become a first-order requirement for modern hardware design. As a design progresses and more detailed information becomes available, more accurate power estimations become possible but at the cost of significantly slower simulation speeds. Power simulation that is both sufficiently-accurate and fast would have a positive impact on architecture and design.**

In this paper, we propose PrEsto, a power modeling methodology that improves the speed and accuracy of power estimation through FPGA-acceleration. PrEsto automatically generates FPGA-based power estimators consisting of linear models that are designed to be integrated into fast, accurate FPGA-based performance simulators of microprocessors. Our prototype implementation predicts the *cycle-by-cycle* power dissipation of the LEON3 core and the ARM Cortex-A8 core to within 6% of a commercial gate-level power estimation tool, while running *several* orders of magnitude faster. The combination of simulation speed and accuracy is not only useful to architects and designers, it is fast enough to be useful for power-sensitive operating system and application developers.

I. Introduction

Reduced and/or bounded power dissipation has become a first-order requirement for virtually all engineered systems. Power has become especially important in microprocessors, ranging from the lowest-end embedded processors to the highest-end server processors, due to their ubiquity and tendency to consume a large fraction of any system's power budget. Thus, the better the available tools to predict the power dissipation of a particular design, the better the end-product.

Detailed power simulations using standard EDA tools run extremely slowly, making running full applications virtually impossible. Thus, though the simulator itself may be accurate, the overall results could be inaccurate if what is simulated is not representative. To improve power simulation speeds, *spreadsheet power modeling*, where the number of times the component is used by a sequence of operations is multiplied by a pre-computed fixed energy consumption per use, is often employed. This technique is, however, inaccurate because it does not account for either timing information or input data.

Detailed power simulations require detailed toggle activity (traditionally generated by RTL simulation) that itself is traditionally slow. The emergence of hardware-accelerated simulation technologies, such as Palladium, Quickturn, or academic projects accelerating computer system simulation [1], [2] have dramatically sped up the generation of toggle activity, opening up the opportunity to speed up accurate power simulation. To do so, however, requires power models that run at roughly the same speeds and can be tightly integrated into the same hardware-accelerated platforms.

There is a plethora of work [3]–[7] in the area of regression-based power estimation. Some model cycle-by-cycle power [4], and others model system-level power [7], but rarely both. These models are generally too complex to implement on hardware-accelerated platforms.

In this paper, we introduce PrEsto (PoweR ESTimatOr), a new power estimation methodology that systematically generates fast, accurate, easy-to-implement, and resource-efficient power models for FPGA platforms. PrEsto models are intended to generate accurately comparable power estimations, both at the architectural level and design phases, and run fast enough for even software developers to power-tune operating systems and applications.

The contributions of this paper are as follows:

- The introduction of a systematic method (PrEsto) to *automatically* generate simple, yet accurate, power models designed to accurately predict *cycle-by-cycle* power. The models are driven by actual control and data transitions. PrEsto power predictions for LEON3 and ARM Cortex-A8 cores are within 6% of gate-level predictions generated by commercial tools (Sections III, V).
- Efficient FPGA-based implementations of PrEsto models that run *several* orders of magnitudes faster than gate-level simulation and at least an order of magnitude faster than software-based architectural power estimation tools while being more accurate (Sections IV, V).
- The introduction and evaluation of FPGA implementations of *macromodels* that can complement the linear power models for irregular components (Section VI).

II. Reducing the Complexity of Power Models

Power consumption can be modeled accurately using detailed models with sufficient information. Accelerating SPICE models with FPGAs have been studied [8], but are impractical for large designs. Ideally, we would like to have power models that are very simple and easy to implement on FPGAs, but still very accurate. The challenge lies in reducing the complexity of detailed power models without sacrificing accuracy.

The power dissipation of a circuit is more sensitive to some primary inputs than others [3]. This observation suggests that the power dissipation of large complex designs, e.g., microprocessors, can be accurately estimated using a small number of *key contributor signals*. Examples of such signals in a microprocessor include cache read/write and hit/miss signals, pipeline stall signals, etc.

Linear Power Models: Given n signals (s_i) in a module as *predictor variables* and the power consumption (P_m) of the module as the *response variable*, the relationship between the response and the predictor variables can be modeled as a linear equation as follows:

$$\begin{aligned} P_m &= c_0 + c_1 s_1 + c_2 s_2 + ... + c_n s_n \\ &\quad + c_{n+1}(s_1 s_2) + c_{n+2}(s_1 s_3) + ... + c_{2^n}(s_1 s_2 ... s_n) \end{aligned} \quad (1)$$

where c_i is the coefficient of each term, capturing the constant portion ($CV^2 f$) of the CMOS dynamic power, $P = \alpha C V^2 f$. The activity factor, α, is determined by the actual transitions of signals, s_i. The predictor variables in Equation (1) include all possible *crosses* of factors, where a cross is the product of two or more signals, corresponding to certain events within the modules that are only triggered when more than one control signals are asserted. Thus, cross terms are also important. The total number of terms in the equation is 2^n. While more signals and thus more terms in the model will eventually result in a perfect complete equation, having more terms is costly both in terms of computation and storage.

We use three approaches to reduce the complexity of the linear model. First, we reduce the design space by using fewer signals (e.g., high-level signals such as signals at module boundaries). Second, for buses, we further reduce the number of bits by using the Hamming distances from cycle to cycle instead of using individual bits, since bits in buses are unlikely to affect power consumption outside the group. Third, we limit both (i) the maximum number of factors per cross term and (ii) the total number of terms in the linear model. The degree of optimization can be adjusted arbitrarily depending on the complexity of the design, desired accuracy and execution platform resource constraints.

III. Automatic Power Model Generation in PrEsto

We propose an automated process that can generate and optimize the power models. The algorithm in Figure 1 describes the Automated Power Model Generation (APMG) process for PrEsto. A group of high-level signals (S) and buses (B) are passed in as candidate factors to be used in the power models. There is no restriction on which signals can be candidate factors. Signals available in high-level models or input/output port signals at HDL module boundaries or internal registers keeping state are good choices for candidate factors. Using even gross architectural intuition and filtering out some rarely used signals (e.g., debug ports) from the list of input/output ports significantly helps reducing the runtime and memory requirement of APMG. We run a set of short training runs using detailed tools (e.g., gate-level simulation) and small training benchmarks. D_S, D_B and D_P are arrays of the data of the signals, buses, and power, respectively, populated over time from the training simulation. D_S and D_B can be multi-dimensional as they may contain more than one signal/bus.

```
Input: Candidate signals and training data($S, D_S$), Candidate buses
    and training data($B, D_B$), Power and training data($P, D_P$),
    Maximum number of terms per module($N$), Coverage
    Threshold($Th_{cov}$)
Output: TotalPowerModel
 1: for all module $m$ do
 2:     $lm \leftarrow GenLinearModel\big(P_m = (1 + \sum_{b_i \in B_m} HD(b_i)) \times (1 + \sum_{s_j \in S_m} s_j + \sum_{s_k \in S_m, s_l \in S_m, s_k \neq s_l} s_k s_l), D_{S_m}, D_{B_m}, D_{P_m}\big)$
 3:     for all term $t$ in $lm$ do
 4:         $MaxVal_t = |t.c| \times max(HD(t.b))$
 5:     end for
 6:     Sort terms by $MaxVal$ and select top $N$ terms
 7:     $T_{1...N} \leftarrow$ selected sorted terms
 8:     Calculate $Coverage$ using $T_{1...N}$
 9:     $PowerModel_m = \sum_{i=1}^{N} T_i$
10:     if $Coverage < Th_{cov}$ then
11:         Add more candidate signals to $S$ and $B$, increase $N$, or
            refine training benchmark
12:         goto line 2
13:     end if
14: end for
15: $TotalPowerModel = \sum_m PowerModel_m$
```

Fig. 1. Algorithm for Automatic Power Model Generation (APMG)

The function *GenLinearModel* in line 2 of the algorithm takes an input formula (without coefficients) and predictor/response variable values as input. It outputs the coefficients for the linear model using linear least squares regression methods. For each module in the design, an input formula is constructed using individual control signals and the Hamming distances (denoted as *HD* in the algorithm) of buses, including the crosses of the factors, as predictor variables and the power as the response variable. The input formula is shown in a factorized form in line 2. There is also an intercept (constant) term in the input formula. The subscripted B_m, S_m and P_m denote the variables specific to module m. $|B_m|$ denotes the number of buses in B_m and $|S_m|$ denotes the number of signals in S_m. If we consider the entire design space including *all* cross factors, the number of terms in the linear model follows $O(2^{|B_m|+|S_m|})$. Thus, the complexity grows exponentially with the number of buses and signals given.

To bound the complexity and to make FPGA implementation practical, we limit the number of factors allowed to form a cross in a term. For all terms that have two or more *HD* factors, additional multiplication operations that cost FPGA resources will be required, whereas one bit control signals are more cost-effective to use. For all the experiments shown in this paper, we limited the cross factors to a three-way product consisting of at most one bus *HD* and at most two control signals as shown in the input formula (line 2). A typical term in the linear model will have the form of $c \cdot HD(b_i) \cdot s_j \cdot s_k$, where c is the coefficient. By limiting the cross factors, the complexity of the model is reduced to $O(|B_m| \cdot |S_m|^2)$. This restriction can be adjusted to achieve better accuracy if FPGA resources and APMG runtimes permit.

APMG further attempts to reduce the number of terms in the linear model and thus reduce the complexity by keeping only the terms with larger potential values. Since terms with an HD component should weigh the coefficients more heavily, we calculate the maximum value of each term as the absolute value of the coefficient($t.c$) multiplied by the maximum HD of the bus component in that term ($t.b$) (line 4), and then select the top N terms that have greatest maximum values where N is specified as a parameter (lines 6-7). A larger value of N will improve the accuracy of the power model at the cost of higher complexity. Analyses on the impact of different values of N (both on accuracy and resource) are presented in Section V.

If certain components in the design are not exercised by the training benchmark, they may not be reflected in the power model. We measure the *static coverage* of the power models to ensure most of the design is covered. Coverage is defined as the proportion of nets that are reachable when propagating from the signals/buses that are chosen for the power models. The fact that a net is *covered* does not necessarily imply that we have the correct weightings for that net, but if a net not covered, its effect cannot be accounted for in the generated equation. If the resulting coverage of the power model is not satisfactory, the user can add more candidate signals and/or refine the training benchmark to exercise the uncovered area, and then repeat the power modeling for the given module (lines 10-12). Since coverage is generally satisfactory even without refinement, the coverage detector is just a safety net. Section V shows that PrEsto initially reaches high coverage. The total power of the design is estimated as a linear sum of the power models for all modules (line 15).

PrEsto automatically generates fast power models using existing power data either generated by a power simulation of RTL mapped to the appropriate process technology, or measured from real parts. Designers can extrapolate existing power models for designs on new process technologies that reuse existing components. Of course, RTL is not available for every next-generation component. In such cases, one can combine PrEsto power models with other power modeling techniques such as (i) dividing the component into smaller building blocks that have RTL or other power models, (ii) analytical models such as those in Wattch [9] or CACTI [10] or (iii) a combination of the two [11] using such models. The total power can be calculated by a linear sum of power estimates modeled by different modeling approaches.

IV. PRESTO POWER MODELS ON FPGAS

Accurate power models require signal toggle information generated by an accurate simulator. Such simulators are often slow, in the 1Hz-100Hz range for gate-level simulators, and in the 1KHz-100KHz range for behavioral RTL or architectural simulators. However, FPGA-based simulators that run in the 1MIPS-10MIPS range have recently been developed. Although software implementations of PrEsto models are significantly faster than gate-level power estimation (Section V), they were designed to be implemented in FPGA-based simulators, enabling accurate power estimation at 10MIPS and faster.

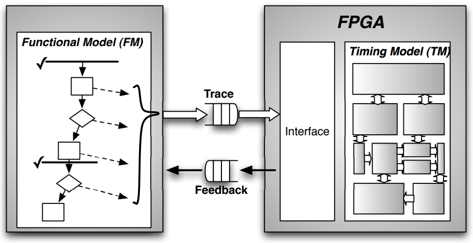

Fig. 2. A high-level view of a FAST simulator

A. Integrating with FAST Simulators

We integrated PrEsto models into an FPGA-Accelerated Simulation Technologies (FAST) [1], [2] simulator prototype to (i) demonstrate that they can be integrated into such simulators, (ii) measure their simulation performance, and (iii) determine the FPGA-resources they require.

A high-level view of a FAST simulator is shown in Figure 2. FAST simulators are partitioned into a Functional Model (FM) and a Timing Model (TM). The FM executes the functional aspect of a system, including the behavior of the ISA and peripherals, while the TM models the timing aspects of the system. The FM *speculatively* executes instructions and passes a trace of those instructions, in the order it fetches and executes them, to the TM that uses that trace to predict timing. Any discrepancy between the FM fetch/execute path and the TM-predicted target fetch/execute path (e.g., branch misprediction) is addressed by the FM rolling back and re-executing under the guidance of the TM. The FM runs efficiently on a microprocessor. The TM is parallelized by running on an FPGA that can efficiently emulate the parallel structures found in modern computer systems. The speculative nature of FAST simulators means the FM and TM can be fairly decoupled while maintaining excellent simulation speed as long as divergence occurs infrequently since rollbacks, and their overhead, only occur when there is divergence.

Since FAST simulators are simulators and not *emulators*, modules can take multiple FPGA cycles to model a single target cycle. For example, instead of trying to squeeze an eight-entry CAM into a single clock cycle, it can be modeled in eight FPGA cycles, reducing TM complexity while enabling high frequencies. By leveraging the multiple FPGA cycle support needed for flexible simulation, more accurate power models than would be possible in a single FPGA cycle can be used.

Also, full emulation approaches can consume a huge amount of FPGA resources. The Intel Atom processor port fills up a Xilinx Virtex-5 LX330 FPGA, even after removing the entire L2 cache [12], leaving no room for power models. The FAST TM is much more efficient, as a great amount of the functionality can be abstracted away. Thus, FPGA-based *simulators* are an ideal environment in which to integrate PrEsto power models.

B. Implementing PrEsto on FPGAs

Figure 3 illustrates the PrEsto power models designed for the FPGA. Each term in the linear model (depicted as T_i in

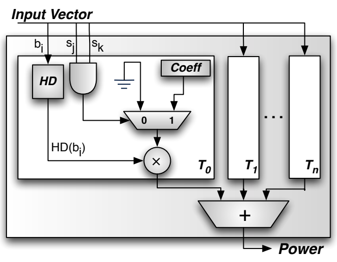

Fig. 3. Block diagram of a PrEsto power model implemented for an FPGA

Core (ISA)	LEON3 (SPARC V8)	Cortex-A8 (ARMv7)
Pipeline	Single-issue, 7-stage in-order	Superscalar (dual-issue) 13-stage
Branch Pred	N/A	Two-level dynamic branch pred
L1 I/D-Cache	4KB, 4-way, LRU	32KB, 4-way, Pseudo-random
L2 Cache	N/A	256KB, 8-way, Pseudo-random
I/D TLB	8 entries, Increment replc.	32 entries, fully assoc.

TABLE I
LEON3 AND CORTEX-A8 CONFIGURATION USED

the figure) takes at most one bus signal (b_i) and two control signals (s_j, s_k) as an input. The HD of a bus is calculated by first XORing the previous and current input vectors, then counting the number of ones. The HD is then multiplied by the coefficient when the selected control signals are true. The HD multiplicand is relatively short (only 6 bits for a 32-bit bus), requiring few resources. Multiplication can be done efficiently using the available DSP slices (192 in Virtex-5 LX330 parts) in most modern FPGA devices. If the user needs more multipliers or chooses not to use the dedicated DSP slices, the multiplication can be implemented with shifters and adders using general FPGA resources. The total predicted power is the sum of the terms.

V. Experimental Results

We applied the PrEsto methodology to two processor implementations. One is the open-source LEON3 [13] core (SPARC V8) using Chartered 130nm standard cell libraries and SRAM compilers. The other is the ARM Cortex-A8 [14] core using TSMC 65nm libraries. The configurations of the cores are shown in Table I. The Cortex-A8 design is roughly 10 times more complex than the LEON3 design. Both designs were synthesized using Synopsys Design Compiler. The Cortex-A8 implementation includes the parasitic capacitances extracted after the place-and-route phase. The gate-level power is estimated using Synopsys PrimeTime PX with VCD files generated from full RTL simulation. The PrEsto APMG algorithm uses the gate-level power and signal transitions from the VCD to generate power models. PrEsto APMG is currently implemented using the open-source statistical package R and Python scripts. The PrEsto training benchmark is a single, simple microbenchmark that utilizes various components in the core and executes to completion in 5K∼10K cycles. The training benchmark is run once to generate power models that are then used without any additional training. Figure 4 summarizes the tool flow for generating PrEsto power models.

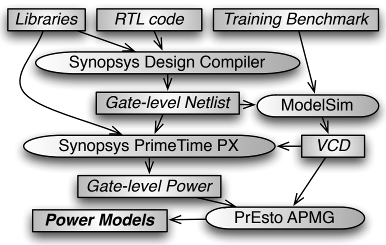

Fig. 4. Tool flow to generate PrEsto power models

The LEON3 and Cortex-A8 cores were divided into 20 and 50 non-leaf modules, respectively, at microarchitecutral component boundaries. A subset of the input/output port signals to each module is considered as *candidate* signals for the PrEsto auto-generated power models. Although this selection process was done manually, it is easy to enforce RTL designers to annotate rarely used signals/buses to automate the process. There is no need to repeat the process when reusing components whose models have already been generated. We are investigating methods to automate the candidate selection.

To measure the accuracy of the auto-generated PrEsto models, we used selected benchmarks from the MiBench suite [15] for the LEON3 core, and benchmarks from the EEMBC suite [16] for the Cortex-A8 core. We use the same power model generated by running 5K∼10K cycles on a single simple training microbenchmark to predict the power of all benchmarks. For the LEON3 core, one million cycles per benchmark were simulated generating both average power and cycle-by-cycle power estimates. For the Cortex-A8 core, we used five samples of 20K cycles evenly spread out throughout the entire run of each benchmark (100K cycles total measured per benchmark) generating the power estimates[1]. The results were compared against gate-level power estimates generated by Synopsys PrimeTime PX for the same simulated period. Although PrEsto models can predict power very quickly, the slow speed of simulations and prohibitive size of dump files needed to *validate* the accuracies prevented longer runs.

Power models generated from PrEsto were integrated into a FAST simulator prototype that was augmented to pass data values for use by the power models. The prototype runs on a Nallatech ACP system that consists of a four-socket Intel server system with Intel Xeon 7350 2.93GHz processors, where one or more of the sockets can be populated with ACP modules (with a Xilinx Virtex 5 LX110 base module and two LX330 compute modules) communicating over the 1066MHz FSB at up to 8GB/s. The FAST FM implementation is based on [17]. Figure 5 shows the high-level block diagram of FAST simulators implemented on an ACP platform. The *synchronization region* is a chunk of memory that the ACP module snoops on and is kept coherent with the processors. In our implementation, four 64B cache-lines (two for reads and two for writes) were used in the synchronization region.

[1]We used a Palladium platform to accelerate RTL simulation and sample throughout benchmarks for the Cortex-A8 experiments.

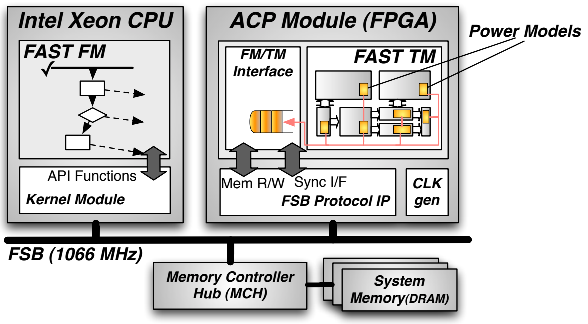

Fig. 5. Block diagram of FAST on an ACP platform

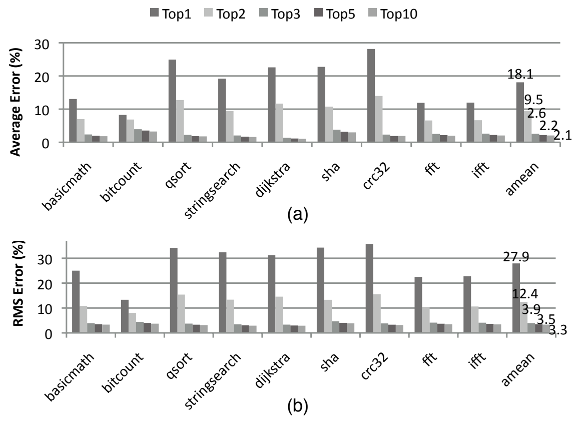

Fig. 6. Accuracy of modeling (a) average power and (b) cycle-by-cycle power, of a LEON3 core using different number of terms per module

These cache lines contain control information such as the head and tail pointers to the circular buffers for traces and statistics, and commands from the TM to the FM. The pointers are lazily updated to avoid cache-line ping-ponging and thus achieve high performance. To efficiently stream out the statistics from the FPGA at runtime, a statistics network was developed.

A. Power Model Accuracies

Figures 6 and 7 show the accuracy of modeling the average power and the cycle-by-cycle power of the LEON3 core and the Cortex-A8 core, respectively. The cycle-by-cycle error is measured as the RMS error of the power at every cycle. The figures show the accuracies when varying the number of terms in the linear models per module (N in Algorithm 1), is shown. The error bars in Figure 7 represent the best and the worst case errors out of the five sampled regions for each benchmark. As expected, more terms result in higher accuracy, subject to diminishing returns after three terms per module for the LEON3 core and ten terms per module for the Cortex-A8 core. For the LEON3 core using three terms per module, the average difference from PrimeTime for the average total processor power is 2.6%, while the average difference in cycle-by-cycle power for the entire processor is 3.9% for the LEON3 core. For Cortex-A8, using ten terms per module results in 1.7% error for average power and 5.8%

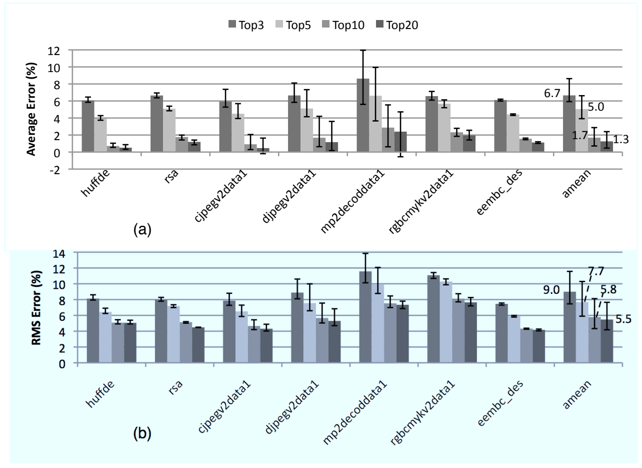

Fig. 7. Accuracy of modeling (a) average power and (b) cycle-by-cycle power, of an ARM Cortex-A8 core using different number of terms per module

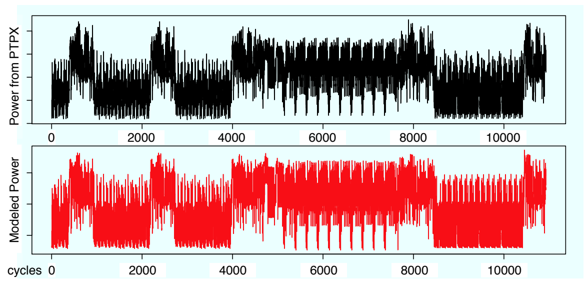

Fig. 8. Comparison of power waveforms (PTPX vs. PrEsto) for Cortex-A8

error for cycle-by-cycle power. However, the Cortex-A8 design is roughly ten times more complex than the LEON3 design, uses a newer technology, and the comparison numbers includes the parasitic capacitances. The worst-case error among the samples is within 10% when using ten terms, implying that high accuracy can be achieved even with resource-efficient power models. Figure 8 shows some cycle-by-cycle power waveforms modeled for the Cortex-A8.

The coverage (defined in Section III) of power models for some key modules were measured by propagating the selected signals through the netlist. Some results are shown in Table II. As expected, array structures have higher coverage than other control modules. Cortex-A8 has slightly lower coverage due to the added complexity. The uncovered nets include debug ports that are normally unused.

LEON3		Cortex-A8	
Module	Coverage	Module	Coverage
RegFile	95.35%	RegFile	92.37%
I/D Cache Data	93.67%	L2 Ctrl	85.87%
I/D Cache Tag	94.32%	LSQ Ctrl	78.83%
I-Cache Ctrl	92.08%	Decode	87.20%
D-Cache Ctrl	85.82%	Scoreboard	67.12%
Pipeline	84.06%	BrPred Ctrl	74.12%

TABLE II
POWER MODEL COVERAGE

# of Terms	2	3	5	10
Slice Registers	163 (0.07%)	227 (0.10%)	317 (0.15%)	569 (0.27%)
Slice LUTs	580 (0.27%)	899 (0.43%)	1464 (0.70%)	2915 (1.40%)

TABLE III
FPGA RESOURCE USAGE OF LINEAR POWER MODELS PER MODULE

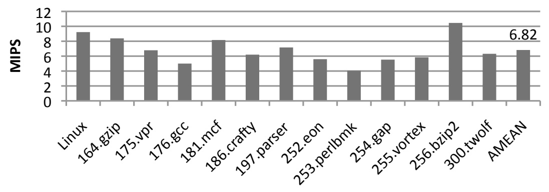

Fig. 9. Speed of PrEsto integrated with FAST

B. Resource Usage of FPGA Implementation

A simple timing model of the LEON3 core was implemented for the Xilinx Virtex-5 LX110 FPGA on the ACP platform. Although the LEON3 core is relatively simple, PrEsto can model arbitrarily complex targets as long as a timing model is available and the FPGA resources permit. More complex designs can be implemented on the LX330s. The PrEsto power models were implemented in Bluespec [18] in a highly parameterizable fashion. Table III shows the number of FPGA resources required for a set of linear models, each with a varying number of terms (N in Algorithm 1). Each term is assumed to include a 32-bit bus as one of the factors. This is conservative as there were an average of 17 bits of signals/buses for the LEON3 and 9 bits of signals/buses for the Cortex-A8, used as factors per term in the linear models. We did not use any DSP slices to implement the multiplications in these results. Had we done so, we would have used fewer general FPGA resrouces. As the results show, each power model takes up very few resources of the FPGA. The percentages of the resources of the Xilinx Virtex-5 LX330 are shown in parentheses.

C. Simulation Performance and Speedup

As mentioned earlier, the TM takes multiple FPGA cycles to model a single target cycle. PrEsto power models currently take seven FPGA cycles to generate a power estimate. This includes two cycles to enqueue a request to the power model request FIFO and dequeue the result from the power model reply FIFO. Our TM currently runs at 133MHz, allowing the power models to simulate around 19M (=133M/7) target cycles per second if the TM is not bottlenecked elsewhere. The power models can be further optimized to take fewer FPGA cycles, but PrEsto power models are not the current bottleneck.

PrEsto integrated with FAST can boot unmodified operating systems, including Linux, and run unmodified applications. The simulation performance of PrEsto while booting Linux and running the SPEC2000 benchmark suite is shown in Figure 9. On average, PrEsto is able to run at around 7 MIPS which is bottlenecked by the speed of the TM.

Synopsys PrimeTime PX simulates the LEON3 core at 100Hz on an Intel Xeon X3230-based 2.66GHz quad-core with 4GB of memory. Due to the complexity, PrimeTime simulates the ARM Cortex-A8 core at 10Hz. FAST can generate toggle information and pipe it to FPGA-based PrEsto power models at 6.8MIPS, or about around 70,000 times faster than PrimeTime PX for LEON3, assuming a target IPC of 1. Assuming a FAST simulator implementation of Cortex-A8 can also run at 7 MIPS[2], PrEsto would be 700,000 times faster than PrimeTime PX. Additionally, PrEsto is at least an order of magnitude faster than other architectural power estimation tools including Wattch (hundreds of KIPS) while potentially being more accurate.

VI. MACROMODELING

A. Overview

The power dissipation of some modules are not very linear with respect to only a few signals and thus are not modeled well with linear models. *Macromodeling* [5], [19] is a technique specifically designed to model complex irregular structures, such as ALUs, FPUs, instruction scheduling logic, etc., that are traditionally modeled poorly with linear models. These techniques can be adopted into the PrEsto flow in case the linear models do not perform well for such components.

We evaluate FPGA implementations of macromodeling technique described in [5] to model irregular structures. The technique uses input vector statistics including average input signal probability (P_{in}), average input transition density (D_{in}) and average input spatial correlation (SC_{in}). These input vector statistics are more formally defined as the following:

$$P_{in} = \frac{1}{n}\sum_{i=1}^{n} P_i, \qquad D_{in} = \frac{1}{n}\sum_{i=1}^{n} D_i \tag{2}$$

$$SC_{in} = \frac{2}{n(n-1)}\sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathcal{P}\{x_i = 1, x_j = 1\} \tag{3}$$

where n is the number of bits in the input vector. P_i is the input signal probability, or the probability the input is driven high, and P_{in} is the average input signal probability. D_i is the input transition density, or the probability the input switches from cycle to cycle, D_{in} is the average input transition density. SC_{in} is the average input spatial correlation and $\mathcal{P}\{x_i = 1, x_j = 1\}$ denotes the probability both x_i and x_j are high simultaneously, where x_i and x_j are the ith and jth bit in the input vector, respectively. P_{in}, D_{in} and SC_{in} are probabilities, ranging from zero to one.

The P_{in}, D_{in} and SC_{in} axes between zero and one are subdivided into intervals of size 0.1 to form a $10\times10\times10$ grid. For each valid grid point in the $(P_{in}, D_{in}, SC_{in})$ space, the power estimate is characterized by generating blocks of input vectors such that the average probability, density, and spatial correlation at the primary inputs are equal to P_{in}, D_{in} and

[2]FAST simulators do not necessarily slow down with more complex targets as long as there are sufficient FPGA resources.

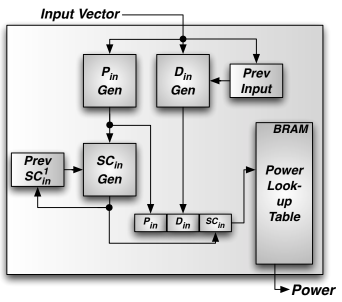

Fig. 10. Block diagram of PrEsto macromodels implemented for an FPGA

SC_{in}, respectively, and obtaining the average power for these input vectors by feeding them through detailed power simulation. We use the Markov chain sequence generator described in [20] to generate these specific blocks of input vectors. The generated power numbers are populated as a look-up table. Power is then estimated by gathering the three statistics at run-time and using them to index into the corresponding entry of the look-up table. The macromodeling technique is slightly changed from the original version [5] in the following ways:

- We modified the macromodels to enable on-the-fly cycle-by-cycle power estimation. In the original macromodeling study [5], which was intended to estimate only average power, the SC_{in} calculation requires knowledge of average input probability that depends on the entire input sequence. We approximate the average input probability by storing and using the results from the previous cycle. SC_{in} is calculated as the average of the SC^1_{in} for the current and the previous cycles, where $SC^1_{in} = \frac{P_{in}(P_{in}-1)}{n(n-1)}$ and n is the number of bits in the input vector.
- In our macromodels, the SC_{in} calculation is approximated by looking up a pre-computed table in FPGA block RAMs that is indexed by the significant bits of P_{in}, instead of actually computing the value. This results in better resource usage at the cost of some loss in accuracy.

B. FPGA Implementation of Macromodels

Figure 10 shows the block diagram of the PrEsto macromodel designed for FPGAs. Generating P_{in} and D_{in} is done by counting the number of set bits in a vector. We implemented bit-counting on an FPGA by partitioning the input vector into 32-bit chunks and operating on each chunk in parallel. The processing of a 32-bit chunk can be done in a single 133MHz clock cycle. The results from each chunk are summed up in the next cycle to obtain the total number of bits set. D_{in} is computed using the same logic as HD in linear models. SC_{in} is approximated by looking up a pre-computed table in a Block RAM that is indexed by the significant bits of P_{in}. If each power entry is a 32-bit value and the most significant 3 bits are used to represent each of the three input statistics (P_{in}, D_{in} and SC_{in}), a 512-entry look-up table could be implemented using only one 18Kb Block RAM per module ($2^9 \cdot 32 < 18K$). In our hardware prototype, SC^1_{in} is implemented as a look-up table indexed by P_{in}. SC_{in} can be better estimated by keeping more history, but that would require more resources.

Input Width	32-bit	64-bit	96-bit	128-bit
Slice Registers	117 (0.05%)	204 (0.09%)	283 (0.13%)	366 (0.17%)
Slice LUTs	275 (0.13%)	431 (0.20%)	585 (0.28%)	746 (0.35%)
Block RAMs	1 (0.34%)	1 (0.34%)	2 (0.69%)	2 (0.69%)

TABLE IV
FPGA RESOURCE USAGE OF MACROMODELS PER MODULE

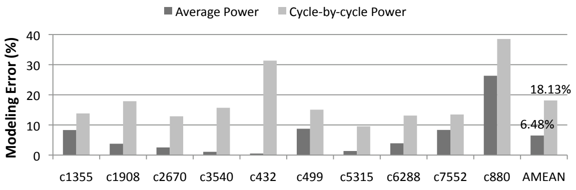

Fig. 11. Macromodeling Accuracy on ISCAS-85 Benchmark Circuits

Table IV summarizes the FPGA resource usage for macromodel with different input widths. The percentages of resource on the Xilinx Virtex-5 LX330 part are shown in parentheses. Macromodels generally require more input bits than the linear models, thus are evaluated with more input bits, roughly corresponding to the native datapath widths within the target. The Xilinx tools automatically chose to map the look-up table for SC^1_{in} in the macromodel to a block RAM for longer inputs and LUTs for shorter inputs.

C. Accuracies of Macromodels

The experiments in Section V were conducted using only linear models. A few small modules (adders, etc.) had slightly higher errors, but did not affect the overall accuracy much. For more complex targets with complex functional units, however, more modules may need macromodels. Thus, to show the accuracies of the proposed macromodeling technique, we conducted separate experiments on the ISCAS-85 benchmark circuits [21]. Synopsys PrimeTime PX with TSMC 0.13μm standard cell libraries was used to gather detailed gate-level power estimation to populate the look-up tables. The accuracies were validated by applying 2000 random input vectors to the macromodels and comparing with the gate-level power estimates for the same vectors.

After the macromodels were generated, random input vectors were used to verify the accuracy of the macromodels. The same randomly-generated vectors were used both for the macromodels and detailed RTL power simulation for comparison. Cycle-by-cycle power was estimated by using only the statistics from the current and previous target cycle. The result of the experiment is depicted in Figure 11. For average power, there was a 6.5% error, while a 18.1% error was found for the cycle-by-cycle power. Although macromodels have higher errors than linear models, there are no known high-level power estimation methods that achieve significantly better accuracies for such irregular modules.

VII. Related Work

Several FPGA-based power estimation methodologies [22]–[24] have been proposed recently. All are emulation-based, mapping the entire target RTL, along with power models, on FPGAs. [25] uses a software/FPGA co-simulation approach to accommodate larger SoC designs, but the partitioning was at module boundaries assuming infrequent communication between modules. The method would suffer from the communication overhead for tightly-coupled modules such as those found in a typical microprocessor. These emulation-based methods have several disadvantages including (i) the difficulty of porting most full-custom/ASIC designs to FPGAs due to very different structures, and (ii) the difficulty of integrating more accurate power models that take several FPGA cycles into designs implemented assuming a single cycle between pipeline registers.

Many regression-based power estimation methods exist [3]–[7], but they either do not model cycle-by-cycle power or were not evaluated at the system-level. Most methods are too complex to implement on FPGAs. PrEsto is much simpler than [4], yet reaches similar accuracies by exploiting knowledge on the RTL design (differentiating signals vs. buses, etc.) Others [26] have proposed using statistical methods to generate power models. PrEsto positions itself between these detailed methods and architectural tools such as Wattch [9], targeting accelerated *system-level* power estimation with simpler heuristics and reasonable accuracies, and is evaluated with full processor implementations.

VIII. Conclusions

We proposed PrEsto, an accelerated power estimation methodology with automated power model generation. The resulting tool runs significantly faster than other architectural or RTL simulators while maintaining high accuracy even when predicting *cycle-by-cycle* power. PrEsto models were designed to be easy to incorporate into FPGA-accelerated simulators, enabling them to model complex targets whose RTL could not map onto a single FPGA. We have validated the methodology on modern industry designs. We are studying how to build thermal and leakage models using PrEsto.

PrEsto has a wide range of uses, ranging from RTL designers that may want to evaluate the power impact of alternative designs, to software developers to power-tune their programs. We believe that such simulators could lead to entire *computer systems* that are systematically designed to have optimized power dissipation.

IX. Acknowledgments

This material is based upon work supported in part by the Semiconductor Research Corporation under Grant No. 2676957812 and by the National Science Foundation under Grant No. 0615352. We thank Jim Holt of Freescale for his generous support for this work, ARM, Inc. for permission to use their designs for our experiments, Bluespec for software donations, and the anonymous reviewers for their feedback.

References

[1] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. H. Reinhart, D. E. Johnson, and Z. Xu, "The FAST Methodology for High-Speed SoC/Computer Simulation," in *ICCAD*, Nov. 2007.

[2] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. H. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators," in *MICRO*, Dec. 2007.

[3] Z. Chen, K. Roy, and T. Chou, "Power sensitivity-a new method to estimate power dissipation considering uncertain specifications of primary inputs," *ICCAD*, 1997.

[4] Q. Wu, Q. Qiu, M. Pedram, and C.-S. Ding;, "Cycle-accurate macro-models for RT-level power analysis," *IEEE Transactions on VLSI Systems*, vol. 6, no. 4, pp. 520 – 528, Dec. 1998.

[5] S. Gupta and F. Najm, "Power modeling for high-level power estimation," *IEEE Transactions on VLSI Systems*, vol. 8, no. 1, pp. 18–29, 2000.

[6] A. Bogliolo, L. Benini, and G. D. Micheli, "Regression-based RTL power modeling," *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 337–372, 2000.

[7] X. Liu and M. Papaefthymiou, "HyPE: Hybrid power estimation for IP-based programmable systems," *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 606–609, Jan. 2003.

[8] N. Kapre and A. DeHon, "Accelerating SPICE Model-Evaluation using FPGAs," *FCCM*, pp. 37–44, 2009.

[9] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *ISCA*, Jun. 2000.

[10] S. Wilton and N. Jouppi, "An enhanced access and cycle time model for on-chip caches," *WRL Research Report 93/5*, Jan. 1994.

[11] X. Liang, K. Turgay, and D. Brooks, "Architectural power models for SRAM and CAM structures based on hybrid analytical/empirical techniques," *Proceedings of ICCAD*, pp. 824–830, Nov. 2007.

[12] P. H. Wang, J. D. Collins, C. T. Weaver, B. Kuttanna, S. Salamian, G. N. Chinya, E. Schuchman, O. Schilling, T. Doil, S. Steibl, and H. Wang, "Intel Atom processor core made FPGA-synthesizable," in *FPGA*, 2009.

[13] "Aeroflex Gaisler," http://www.gaisler.com/.

[14] "ARM webpage," http://www.arm.com/.

[15] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *IEEE International Workshop on Workload Characterization*, pp. 3 – 14, Nov. 2001.

[16] "The Embedded Microprocessor Benchmark Consortium," http://www.eembc.org/. [Online]. Available: http://www.eembc.org/

[17] D. Sunwoo, J. Kim, and D. Chiou, "QUICK: A Flexible Full-System Functional Model," in *Proceedings of ISPASS*, Apr. 2009, pp. 249–258.

[18] "Bluespec webpage," http://www.bluespec.com.

[19] Z. Chen, K. Roy, and E. Chong, "Estimation of power dissipation using a novel power macromodeling technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 11, pp. 1363–1369, 2000.

[20] X. Liu and M. Papaefthymiou, "A Markov chain sequence generator for power macromodeling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 7, pp. 1048–1062, 2004.

[21] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," *Proceedings of ISCAS*, pp. 695–698, Jun. 1985.

[22] J. Coburn, S. Ravi, and A. Raghunathan, "Power emulation: a new paradigm for power estimation," *DAC*, pp. 700–705, 2005.

[23] D. Atienza, P. D. Valle, G. Paci, and F. Poletti, "A fast HW/SW FPGA-based thermal emulation framework for multi-processor system-on-chip," *DAC*, Jan. 2006.

[24] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-system chip multiprocessor power evaluations using FPGA-based emulation," *Proceedings of ISLPED*, Jan. 2008.

[25] M. Ghodrat, K. Lahiri, and A. Raghunathan, "Accelerating System-on-Chip power analysis using hybrid power estimation," *DAC*, pp. 883–886, 2007.

[26] N. Bansal, K. Lahiri, and A. Raghunathan, "Automatic Power Modeling of Infrastructure IP for System-on-Chip Power Analysis," *Proceedings of VLSI Design*, pp. 513–520, 2007.