PowerProbe: Run-time Power Modeling Through Automatic RTL Instrumentation

Davide Zoni, Luca Cremona and William Fornaciari
Politecnico di Milano – Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133 Milano, Italy
davide.zoni@polimi.it, luca2.cremona@mail.polimi.it, william.fornaciari@polimi.it

***Abstract*—Online power monitoring represents a de-facto solution to enable energy- and power-aware run-time optimizations for current and future computing architectures. Traditionally, the performance counters of the target architecture are used to feed a software-based, power model that is continuously updated to deliver the required run-time power estimates. The solution introduces a non-negligible performance and energy overhead. Moreover, it is limited to the availability of such performance counters that, however, are not primarily intended for online power monitoring.**

This paper introduces *PowerProbe*, a run-time power monitoring methodology that automatically extracts and implements a power model from the RTL description of the target architecture. The solution does not leverage any performance counter to ensure wide applicability. Moreover, the use of ad-hoc hardware that continuously updates the power estimate minimizes both the performance and the power overheads. We employ a fully compliant OpenRisc 1000 implementation to validate *PowerProbe*. The results highlight an average prediction error within 9% (standard deviation less than 2%), with a power and area overheads limited to 6.89% and 4.71%, respectively.

***Index Terms*—Low Power, Run-Time, Power Modeling, RTL**

I. Introduction

The continuous increase in performance requirements which has been imposed by current applications motivated the widespread adoption of multi-core architectures ranging from embedded to High Performance Computing (HPC) solutions. In this scenario, power consumption represents a major challenge for the entire *computer continuum*. The performance of both embedded and high-end computing architectures is in fact limited by both the available energy budget and the power related technology constraints. Servers' processors are typically constrained by both the max junction temperature and the cooling systems, thus imposing energy-aware server consolidation techniques [1] to limit the average power consumption while optimizing the energy-performance trade-off. Conversely, the limited energy budget of battery powered embedded systems imposes the design of complex energy-performance resource allocation schemes [2] and aggressive power saving infrastructures [3], [4].

To this extent, providing accurate power estimates to both the software and the hardware architects is of paramount importance to enable *i)* power-aware architectural optimizations at design time and *ii)* a continuous monitoring of the power profile of the architecture, thus enabling run-time schemes for power-performance resource allocation. The majority of available Computer-Aided Design (CAD) tools already support some sort of power modeling feature to fulfill *i)*, while *ii)*, which is less standard, is still evolving to cope with the ever increasing constraints in terms of accuracy and low latency imposed by the applications.

The online power monitoring emerged as a viable solution to fulfill requirement *ii)*. It leverages the *performance counters* [5] to provide a run-time power estimate of the target architecture. The generic methodology is made of two steps. First, the most promising performance counters are selected from the ones available on the target architecture, to develop the power model. Then, the power model is implemented as a software subroutine that periodically updates the power estimate. The use of performance counters demonstrated a good fit between the predicted and the real power consumption of the target. It also provides a low cost and flexible solution to the run-time power monitoring problem while introducing two drawbacks. First, performance counters are not primarily intended for power modeling purposes and, even worse, they cannot be always available, thus limiting the feasibility of the related power monitoring solutions. Second, the update of the power estimate is achieved through a software routine that consumes CPU time and energy. The overheads become more severe with both the increase in the update frequency and the decrease in the computational power of the target architecture.

Contributions - This paper proposes *PowerProbe*, a run-time power monitoring methodology that automatically extracts and implements a power model from the RTL description of the target architecture. The extracted power model is embedded in the target's RTL description of the target to provide online power monitoring support. In particular, the extracted power estimates are exported at the software level via dedicated special purpose registers. The proposed solution is useful for peripherals and hardware accelerators, as well as for general purpose embedded CPUs, because it is not limited to the available performance counters. As a representative validation scenario, the *PowerProbe* has been applied to an RTL CPU description that is fully compliant with the *OpenRisc 1000* ISA [6], allowing the experimental assessment of three major contributions.

- **Accurate Power Model -** The extracted power model is tailored to the instance of the target architecture from which suitable probes are selected for the power computation. In contrast with the state-of-the-art no performance counters are considered. Experimental validation shows an average prediction error that is lower than 9% (standard deviation limited to 2%) between the real power and the power estimates obtained from a post-synthesis implementation of the *PowerProbe* power model.
- **Automatic Power Model Extraction -** Given a target RTL instance, *PowerProbe* consists of three stages. First, a simulation phase collects the architectural statistics and the related time-based power traces. Second, a power model is obtained by means of a linear regression approach. Finally, the additional RTL structures used to implement the derived power model are added to the target RTL description, thus allowing the software level -at runtime- to readout a continuously updated power estimate by means of special purpose registers.
- **Fast Power Estimate Evaluation with Low Overheads -** *PowerProbe* leverages ad-hoc hardware components to continuously update the power estimates without affecting the performance of the target architecture. The validation results show area and power overheads limited to 6.89% and 4.71%.

978-3-9819263-0-9/DATE18/©2018 EDAA

The rest of the paper is organized in four parts. Section II overviews the state-of-the-art on performance counter based power models and the literature on power model extraction from RTL descriptions. Section III presents the *PowerProbe* approach, detailing its working flow. The experimental validation is discussed in Section IV describing the *PowerProbe* methodology applied to an *OpenRisc 1000* compliant embedded CPU, while Section V draws the relevant conclusions.

II. Related works

The literature on power modeling approaches for computing architectures can be organized in two classes: *run-time* and *design time*. Run-time approaches provide an accurate online estimate of the device's power consumption while the latter deliver an accurate power model to be used during the design stages.

Run-time Approaches. Due to their flexibility, performance counter based models have been extensively used for CPU power estimation, since they are software implemented and it is possible to estimate the power model after the computing device has been shipped. Such power models rely on the connection between the operations performed by the CPU, monitored through the performance counter, and the power consumption *correlated* to such activities.

[7] assesses the possibility to identify a universal subset of performance counters that can be used to estimate the power consumption of any computing architecture. The analysis is provided on both the Intel Atom and the Intel Nahalem processors, highlighting an accuracy below 5%. Despite the accuracy of the proposed solution, the possibility to extract performance counter information in the embedded multi-cores is limited by both the actual number of concurrently monitored counters and the effective availability of counters. In particular, [8] puts in evidence the scarcity of power modeling solutions for embedded and mobile multi-cores, and it presents a tool for system level power estimation. The tool leverages a performance simulator to extract the performance counters to feed the identified power model. However, different simplifications and mismatches between the simulated and the real architecture diminish the accuracy of the final power model.

[5] presents a run-time power modeling methodology for mobile multi-cores leveraging the architectural performance counters. The evaluation is conducted on a real big.LITTLE architecture [9] that features a clustered multi-core. In contrast with [5], *PowerProbe* embeds a custom power model within the RTL sources to later synthesize architectures with online power monitoring capabilities. Since the power estimates are computed employing dedicated hardware structures, *PowerProbe* does not affect the performance of the running tasks.

Intel RAPL [10] delivers hardware-side online power monitoring solutions. The power model is implemented as part of the RTL description of the CPU. Moreover, the power estimates are made available to the software level by means of dedicated registers. Despite the similarity with *PowerProbe*, *Intel RALP* represents an ad-hoc solution for Intel CPUs only. In contrast, *PowerProbe* proposes a methodology to automatically derive a power model for a generic RTL instance. The model is automatically added to the RTL description of the target and the power estimates are exposed at software level by means of a special purpose register.

Design Time Approaches. McPAT [11] is a power, area and timing modeling framework for computing architectures. It is suitable for early *Design Space Exploration* (DSE) due to its high flexibility in adapting to different design specifications at the cost of a reduced accuracy, as demonstrated in [12]. *Orion3.0* [13], DSENT [14] and CACTI [15] aim at the same goal, although they model a specific portion of the computing architecture. However, their complex structure and, in general, the lack of accuracy compared to customized power models directly extracted from the computing architecture instance, prevent their use in on-line power monitoring scenarios.

A different research direction aims at deriving the power model from the RTL instance to increase the accuracy, at the cost of a reduced reusability. [16] presents a linear regression based methodology to estimate the average power consumption of generic RTL components. These are then used at design time without addressing the problem of post-delivery power monitoring. [17] elaborates on the power modeling errors due to the wrong selection of model family and provides a multi-model framework to increase the accuracy of the identified power model for a specific RTL instance. Several proposals focus on more flexible power models by abstracting away the implementation details. [18] discusses an high level power modeling framework that separates the activity of the RTL module from its implementation, thus allowing a greater accuracy regardless of the actual operating frequency and the target technology that are known to influence the implementation of the RTL component. [19] proposes a design time methodology that enables the modeling of the dynamic current profile, in the absence of any geometry information and estimation of SoC power noise.

Differently from the literature on the design time power modeling, the *PowerProbe* methodology delivers an accurate power model of the RTL instance at hand, and automatically inserts such model in the RTL description of the target. Online power monitoring capability is made accessible to the software layer through a dedicated special purpose register.

III. The PowerSpy Architecture

This section presents *PowerProbe*, a novel solution that can automatically instrument the RTL of a generic design to provide online power monitoring to the software level. The goal is to deliver a flexible and generic run-time power model within the RTL of the target design to overcome two limitations of current performance counter based power models: *i)* hindrance of the execution of the platform tasks by subtracting precious CPU cycles that are used to update the power estimate and *ii)* leveraging on the available performance counters that are not primarily intended for power modeling, thus possibly inducing inaccurate results.

Figure 1 shows up the four-stage *PowerProbe* simulation flow. It is general enough to allow the power model creation and RTL instrumentation of any architecture for which the HDL description is available, thus making the methodology suitable for both peripherals or hardware accelerators and for general purpose CPUs.

Starting from the HDL description of the target architecture, the *Logic Synthesis, Mapping and Simulation Stage* employs *Vivado 2017.1* to synthesize, map and simulate the design to extract the Value Change Dump (VCD) information as well as the architectural statistics that are later used for the power model estimation. The toggle counts of the primary input and output signals for each module in the target architecture are collected, in addition to the microarchitectural statistics (see *uarch Stats* in Figure 1). The extensive literature on *performance counter* power models demonstrate the effectiveness of relating the toggle count of a signal to the power consumption. In particular, we followed such principle to analyze the correlation between the power trace and any possible primary input and output signal for each module in the target architecture.

The VCD file is then pre-processed and fed into the *Vivado Report Power* tool to obtain the time-based power trace (see *Power Trace*

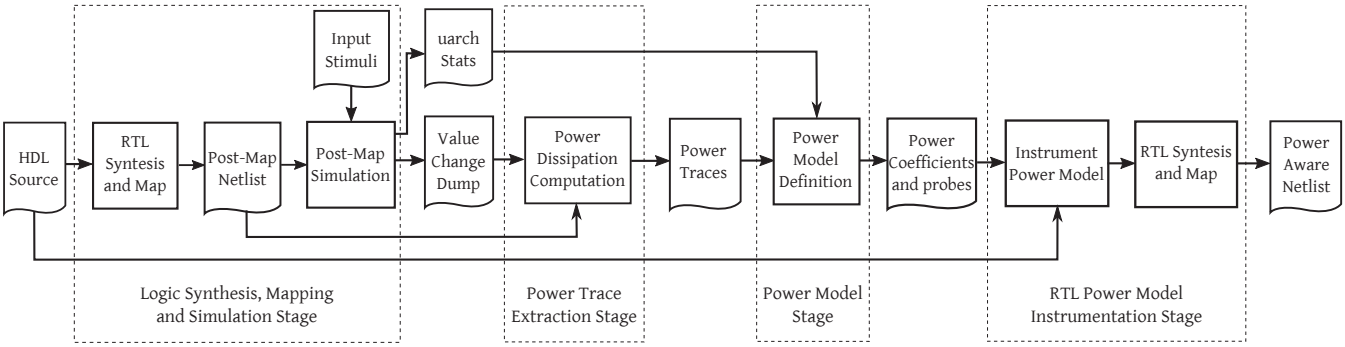


Fig. 1: Overview of the simulation flow used within *PowerProbe*. The post-map logic simulation provides the power traces and the microarchitectural statistics used to derive the power model. The RTL is instrumented with the derived power model and then synthesized again.

Extraction Stage in Figure 1). Indeed, *Vivado Report Power* provides the average power consumption for the design at hand starting from the Switching Activity Interchange Format (SAIF) file and the target netlist. To this extent, a specific VCD to SAIF converter has been developed to interface *Vivado Report Power* by iterating on the single VCD file and to then produce multiple SAIF files. Each SAIF contains data to compute power on a time window set to 100 clock cycles to balance the computational effort and the accuracy of the power trace. The final power trace for each module in the hierarchy is obtained as a series of power samples where each one integrates the power consumption of the target architecture within the time window.

The *Power Model Stage* takes the power traces and the microarchitectural statistics to deliver the power model for the target architecture. The power model is made of both a set of coefficients and the related RTL signals of the target architecture for which we collect the toggle count to feed the model. The selected RTL signals are a set of primary inputs and/or outputs of one or more modules within the design hierarchy of the target architecture, for which the *Power Model Stage* estimates a good fit with the real power consumption of the target (see Section III-A).

Starting from the obtained power model and the HDL source, the *RTL Power Model Instrumentation, Synthesis and Map Stage* instruments the target architecture to embed the power model in the final netlist. In particular, Section III-B discusses the instrumentation flow to insert the ad-hoc toggle counter modules, the additional glue logic to interconnect the power model components, and the actual implementation of the power model computational RTL.

A. Power Model Stage

PowerProbe automatically derives a linear power model for the target architecture starting from the RTL description and the time-based power trace. First order models are adopted to ease the hardware implementation and to minimize area and power overheads.

The extraction and RTL implementation of a power model is a multi-objective problem for which *PowerProbe* considers four different dimensions: accuracy as well as performance, power and area overheads. The prediction error of the power model for the module m ($\mathbf{e}_m$) is defined as the average of the relative errors between the real power and the power estimated from the model (see Equation 1 where $p_{i,m}$ and $\hat{p_{i,m}}$ are the real and estimated power values for each time sample i in module m).

$$\mathbf{e}_m = \frac{\sum_{i=1}^{S} \frac{\|\hat{p_{i,m}} - p_{i,m}\|}{p_i}}{S} \tag{1}$$

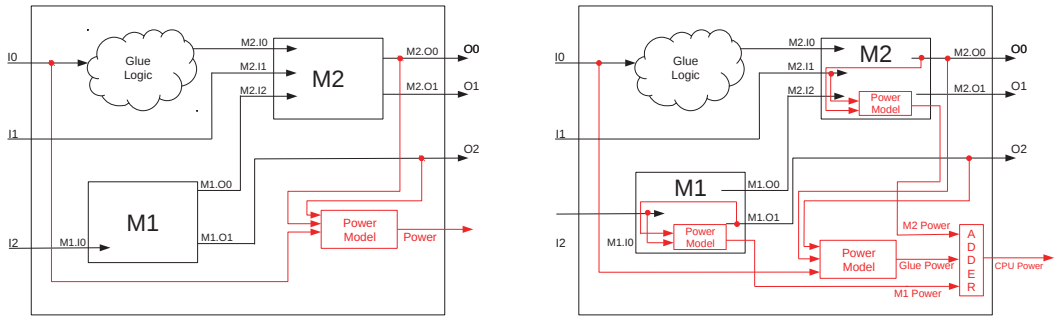
The performance overhead is defined as the CPU time to update the power model estimates. The ad-hoc RTL implementation runs the power model update in parallel with the CPU execution. This means that the performance overhead is almost zero regardless the complexity of the power model. Conversely, the power and area overheads (Θ_P and Θ_A, respectively) are a function of the number of variables used in the extracted power model. In fact, for any sampled signal, a toggle counter and a multiplier have to be added side by side with the additional logic used to connect the two components.

PowerProbe leverages the deterministic structure of the module hierarchy within the HDL sources to automatically derive the power model of the target architecture. Indeed, regardless of the complexity of the HDL description at hand, a generic module within the hierarchy can be a leaf or an intermediate module. The power model for both the leaf and the intermediate modules can be estimated as a function of their primary input and/or output signals, i.e., using the *direct modeling technique* (see Figure 2a). However, the power model of an intermediate module can be also obtained by adding the power contributes from all its submodules and the power of its glue logic, namely by using the *indirect modeling* approach (see Figure2b).

$$R'_m = \frac{\Delta_m(\mathbf{e})}{\Delta_m(\Theta_{Area})} = \frac{\mathbf{e}_m / \mathbf{e}_{m.submod}}{\Theta_{m.submod,Area} / \Theta_{m,Area}} \tag{2}$$

Algorithm 1 is the outer loop of the methodology. It visits the module hierarchy of the target architecture in a top-down fashion and exploits two driving metrics for the exploration: the prediction error ($\mathbf{e}$) as defined by Equation 1 and the marginal revenue (R'). The marginal revenue (R') is defined by Equation 2 and represents the cost, in terms of area and power, to increase the power model accuracy by moving one level deeper in the module hierarchy to identify the power model of module m. In general, a model extracted from a deeper level of the hierarchy shows a better accuracy, paid in terms of a higher area and power consumption. This happens because of the increased number of signals used to model the system power. The R' metric considers area overhead only, without referencing power since, as discussed in Section IV, they are strongly correlated.

Algorithm 1 starts evaluating the root module M_0 by extracting two power models, namely *i)* the direct model, and *ii)* the indirect model. The $\mathbf{e}_m$ and R'_m are computed and evaluated for M_0. Depending on the result of the evaluation, the algorithm can visit the next level of the hierarchy. In particular, the visit terminates if two conditions are met: $\mathbf{e}_m$ is within the accuracy threshold that is a design time constraints and R'_m is less than 1. The second requirement verifies if the improvement of model accuracy is higher than the extra power and area costs for the estimation of a power model considering M_0 submodules. Otherwise, the visit to the hierarchy continues by returning a set of power estimates of the modules or by terminating



(a) Direct Power Model.

(b) Indirect Power Model.

Fig. 2: The power model of a module in the hierarchy is built from a weighted combination of its primary inputs and outputs (a) or by adding the power contributions of all of its submodules plus the power contribution of its glue logic (b).

Algorithm 1 Top-down hierarchical visiting algorithm.

1: **function** $[cnts, coefs]$ VISIT(M_0)
2: $[C, L_{cnts}, \mathbf{e}_{M_0}] = ComputePwr(M_0)$;
3: $[\Theta_A, \Theta_A] = ComputeOH(cnt)$;
4: **for** $j = 1 : max\_depth$ **do**
5: **for** $i \in m.sub$ **do**
6: $[C_j, L_{cnts,j}, \mathbf{e}_{m,j}] = ComputePwr(i)$;
7: $[\Theta_{A_t,j}, \Theta_{A_s,j}] = ComputeOH(cnt_j)$;
8: $\mathbf{e}_{m_s,j} = Combine(\mathbf{e}_{m_s,j}, \mathbf{e}_{m_t,j})$;
9: $\Theta_{A_s,j} = Combine(\Theta_{A_s,j}, \Theta_{A_t,j})$;
10: $\Theta_{P_s,j} = Combine(\Theta_{P_s,j}, \Theta_{P_t,j})$;
11: **end for**
12: **if** $\mathbf{e}_m < Th$ **then**
13: $R' = \frac{\mathbf{e}_{m.sub}/\mathbf{e}_m}{\Theta_{A,m.sub}/\Theta_A}$;
14: **end if**
15: **if** $R' < 1$ **then**
16: $break$;
17: **else**
18: $cnts = cnts_j;\ coefs = coefs_j$;
19: **end if**
20: **end for**
21: **end function**

Algorithm 2 Power model computation for module m.

1: **function** $[C, L_{cnt}, \mathbf{e}]$ COMPUTEPWR(m)
2: **for** $b = 1 : 10$ **do**
3: $r = rand(0, 1)$;
4: $[T_{pwr}, D_{pwr}] = partition(Trace_{pwr}, r)$;
5: **for** $i = 1 : cnts$ **do**
6: **for** $L_{cnt,tmp} = \binom{cnts}{i}$; **do**
7: $[R_t^2, C_t] = linReg(L_{cnt,t}, T_{pwr}, m)$;
8: **if** $R^2 < R_t^2$ **then**
9: C_b=C^t; R_b^2=R_t^2; $List_{cnt,b}$=$List_{cnt,t}$;
10: **end if**
11: **end for**
12: **end for**
13: $//UpdateAccuracy$
14: $\mathbf{e}_b = ComputeErr(D_{pwr}, C_b, List_{cnt,b}, S[List_{cnt,b}])$;
15: **if** $\mathbf{e}_b < \mathbf{e}$ **then**
16: $\mathbf{e} = \mathbf{e}_b;\ C = C_b;\ L_{cnt} = L_{cnt,b}$;
17: **end if**
18: **end for**
19: **end function**

with an *error*, due to the impossibility to achieve the target accuracy.

Algorithm 2 details the steps to compute the power model given a netlist and the related power trace. The coefficient (R^2) measures the capability of the extracted power model to follow the variations in the real power value and it is used to compare different power models for the same module.

$$R^2 = \frac{\Sigma_t(\hat{y}_t - \bar{y})^2}{\Sigma_t(y_t - \bar{y})^2}, t \in TSs \tag{3}$$

The algorithm exploits the bootstrap re-sampling approach [20] to increase the accuracy of the final power model by executing ten iterations. At the beginning of each iteration the benchmarks are randomly selected to be part of either the training or the test set using a $7:3$ proportion. The training set is used to extract a power model for each combination of counters ($cnts$), namely primary inputs and outputs, for which the one with higher R^2 is saved. The procedure is repeated considering tuples of n counters each, where $n \in 1, .., cnts$ (see lines 5-12 in Algorithm 2). The prediction error ($\mathbf{e}_{m,temp}$) of the obtained power model is computed on the test set ($testSet_{pwr}$) and compared with the currently best E_m to decide if the newly computed power model better fits the power trace of module m. The procedure terminates by returning the best identified power model or an empty model structure.

B. RTL Power Model Instrumentation Stage

This stage of the *PowerProbe* methodology augments the target architecture with the RTL components used to implement the power model obtained in the *Power Model Extraction Stage*. The aim is to implement the power model in the target architecture also considering three different aspects: *i)* computation logic *ii)*, power information flow, and *iii)* signal monitoring logic.

Computation Logic - *PowerProbe* leverages fixed point arithmetic [21] by using twelve bits for the coefficients of the power model and the power values, i.e. $Q12, 8$, and a seven bit signal to drive the toggle counter values. The fixed point arithmetic reduces the complexity of the RTL implementation dramatically, since it allows to compute the power model by means of fixed point multipliers and

adders. We observed an increase in the prediction error lower that 1% in comparison to the use of the floating point.

Power Information Flow - Each module in the hierarchy is automatically augmented with an additional twelve bit output port to drive the computed power value to the upper modules in the hierarchy. In particular, the power module computation logic is inserted within the module that has at least one of its primary input or output listed in the counters selected during the model definition. Indeed, the required toggle counters and multipliers are directly instantiated within the module, thus the power value is the only signal that traverses the module hierarchy.

Software Monitoring Interface - The power values are conveyed to the top module of the design hierarchy where an adder computes the final power for the top design instance. The final value is then linked to the special purpose register module that implements an additional register to make the results visible at software level.

IV. Results

This section quantitatively validates the *PowerProbe* methodology in terms of the accuracy of the power model, considering also the power, performance and area overheads. Another goal is to pinpoint the need for an ad-hoc and low cost RTL-based infrastructure to support dynamic resource management or thermal control without imposing a significant reduction in the overall system performance.

Validation Setup - We employed the *Mor1kx* System-on-Chip (SoC) [22] that implements the royalty-free OpenRISC 1000 architectural specification [6] and provides a representative low-end embedded CPU use case. In particular, the *Mor1kx-SoC* embeds a single 32-bit, single-issue, in-order CPU with a 5-stage pipeline, the main memory and a Wishbone compliant bus [23] implementing a Single Read/Write data transfer protocol. The SoC instance does not provide the cache and the Floating Point Unit (FPU) for which the power model extraction is left as a future work. A software-based version of the same power model extracted by *PowerProbe* is compiled and executed on the target CPU to highlight its performance overhead compared to the RTL-based power model.

Instead of collecting the power traces directly on the FPGA, we employ a "clean room" fine grain environment, i.e., post place and route (PAR) netlist, to accurately evaluate the power contributions of the different modules in the target architecture. Starting from the HDL description, the SoC is synthesized and mapped onto the *Artix 7 XC7A100T* Xilinx FPGA employing the Xilinx Vivado 2017.1 toolchain and setting the synthesis clock frequency at 50MHz without enforcing area constraints. the post-synthesis simulation are executed at 42MHz. The obtained netlist is simulated using Xilinx XSim 2017.1 to extract the power trace and the required statistics for the model identification. The statistics are the toggle counts for the primary input and output of each module in the hierarchy and they are periodically sampled every 100 cycles.

Model Accuracy and RTL Power and Area Overheads - Table I shows the area and power overheads for the RTL power model of the considered *OpenRisc 1000* CPU. Results are normalized with respect to the power and area of the CPU and three scenarios are considered. First, a single power counter reports an area and power overheads of 1.08% and 0.78%, respectively. The power model extracted using the direct modeling approach 2a from the top level module using 6 counters (see *Pwr Model (Lev:0 - Top)* in Table I) increases the area by 6.89% and the power consumption by 4.71%. Conversely, the indirect power model obtained from the submodules of the top

	single power counter (1 cnt + 1 mult + glue)	Pwr Model (Lev:0 - Top) (6 counters)	Pwr Model (Lev: 1) (60 counters)
Area (%)	1.08	6.89	68.01
Power (%)	0.78	4.71	54.69

TABLE I: Relative power and area overheads normalized to the considered RISC CPU. Results are for three scenarios. Single counter as the contribution of the toggle counter, the fixed point multiplier and the interconnecting glue logic. as well as the power models obtained from the top module and the sum of power models for all the its submodules.

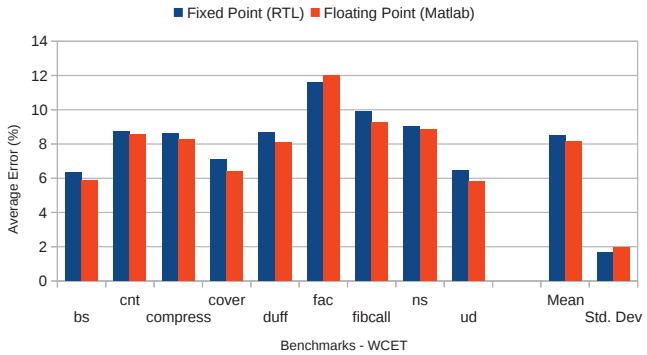


Fig. 3: The average prediction errors, using both floating point and fixed point arithmetic, of the best power model generated using *PowerProbe* compared with the real power consumption of the entire RISC CPU. The prediction error is computed on 9 benchmarks and the Fixed Point (RTL) results refers to the RTL implementation of the power model, while Floating Point (Matlab) shows the results of the same model executed in Matlab.

module requires 60 power counters with an area and power overheads up to 68.01% and 54.69%, respectively.

The results of the prediction error (e) considering the *Pwr Model (Lev:0 - Top)* power model are reported in Figure 3 for 9 benchmarks of the WCET suite [24]. The same power model has been implemented in both floating point (Matlab) and fixed point arithmetic (RTL) and the prediction error results are reported for both to pinpoint the limited increase in the prediction error (less than 1% for all the benchmarks) due to the use of the latter arithmetic implementation.

The prediction error is below 12% with an average of 8.12% and 8.48% for the floating point and fixed point implementations, respectively. Moreover, standard deviation below 2% assesses the validity of the *PowerProbe* methodology that allows accurate and low cost implementations of online power monitoring RTL infrastructures.

```
//mfspr function and SPR_PCMx addresses
#include "./risc-utils.h"
// power model known term
#define C0 0.8394
float updatePwr (){
        float res=0;
        // power model coefficients
        // from PowerProbe
        float coeff[6]={0.3466, 0.2890,
                        -0.2279, 0.3088,
                        0.3388, 0.2853};

        for(i = 0; i<6; i++){
                unsigned long x=0;
                x = mfspr(SPR_PCM_BASE + (i));
                res = res + x*coeff[i];
        }
        return res+C0;
}
```

Listing 1: Software-based power model using ad-hoc performance counters to mimic the RTL power model provided by PowerProbe.

Online Power Monitoring: Performance Implications - The model obtained using *PowerProbe* has been coded in C (see Listing 1) and executed on the target *OpenRisc 1000* CPU to compare the performance overheads of the hardware and software-based power modeling. The C code is compiled with the *O3* optimization flag employing the *or1k-elf-gcc-4.9* compiler and it is executed as a bare metal application. We obtain an execution time of 227us to run the *updatePwr* C function (without considering the initialization code and the context switch), namely for a single update of the power estimate. Such software implements the very same power model of the RTL implementation. Since the fixed point support within *or1k-elf-gcc* is not optimized, the use of floating point instead of fixed point does not affect the performance of the binary. The update of the power estimate severely affects the CPU performance and prevents the exploitation of the online power monitoring for the tasks controlling fast dynamic phenomena. In contrast, *PowerProbe* provides an online power monitoring infrastructure to satisfy fast dynamic control systems. The hardware update of the power estimates is in fact almost a zero latency procedure, for which the computing capacity of the target is not affected.

V. Conclusions

This paper presented *PowerProbe*, a run-time power monitoring methodology that automatically extracts and implements a power model from the RTL description of the target architecture. The methodology has been validated considering an *OpenRisc 1000* compliant CPU for which the instrumented power model obtains an average prediction error below 9% (standard deviation within 2%) and an area and power overheads limited to 6.89% and 4.71%, respectively.

PowerProbe has been compared with a performance counter-based power monitoring solution that executes at software level. Such results confirm the limited usability of the traditional performance counter-based power models to feed any resource allocation policy that controls fast dynamic processes. In contrast, *PowerProbe* is fast in computing the power estimates without affecting the target performance. It is worth noticing that the process variability aspects that can impact the accuracy of the power estimates within *PowerProbe* are not considered and left as future work.

VI. Acknowledgments

This work was partially supported by two EU grants within the EU H2020 Research and Innovation Programme: "MANGO" Grant agreement no. 671668 and "M^{2}DC" Grant agreement no. 688201.

References

[1] A. Sansottera, D. Zoni, P. Cremonesi, and W. Fornaciari, "Consolidation of multi-tier workloads with performance and reliability constraints," in *2012 International Conference on High Performance Computing Simulation (HPCS)*, July 2012, pp. 74–83.

[2] S. Libutti, G. Massari, P. Bellasi, and W. Fornaciari, "Exploiting performance counters for energy efficient co-scheduling of mixed workloads on multi-core platforms," ser. PARMA-DITAM '14. New York, NY, USA: ACM, 2014, pp. 27:27–27:32.

[3] D. Zoni, A. Canidio, W. Fornaciari, P. Englezakis, C. Nicopoulos, and Y. Sazeides, "Blackout," *J. Parallel Distrib. Comput.*, vol. 104, no. C, pp. 130–145, Jun. 2017.

[4] D. Zoni, J. Flich, and W. Fornaciari, "Cutbuf: Buffer management and router design for traffic mixing in vnet-based nocs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1603–1616, June 2016.

[5] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, Jan 2017.

[6] OpenRISC Project, "OpenRISC 1000 Architectural Manual," OPENCORES.ORG, Tech. Rep., 2014.

[7] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu, "A study on the use of performance counters to estimate power in microprocessors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 882–886, Dec 2013.

[8] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman, "System-level power estimation tool for embedded processor based platforms," in *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '14. New York, NY, USA: ACM, 2014, pp. 5:1–5:8.

[9] ARM Ltd., "The big.LITTLE Architecture," Tech. Rep., October 2011. [Online]. Available: http://www.arm.com/products/processors/technologies/biglittleprocessing.php

[10] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-management architecture of the intel microarchitecture code-named sandy bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, March 2012.

[11] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 469–480.

[12] S. L. Xi, H. Jacobson, P. Bose, G. Y. Wei, and D. Brooks, "Quantifying sources of error in mcpat and potential impacts on architectural studies," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2015, pp. 577–589.

[13] A. B. Kahng, B. Lin, and S. Nath, "Orion3.0: A comprehensive noc router estimation tool," *IEEE Embedded Systems Letters*, vol. 7, no. 2, pp. 41–45, June 2015.

[14] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic, "Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, May 2012, pp. 201–210.

[15] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Architecting efficient interconnects for large caches with cacti 6.0," *IEEE Micro*, vol. 28, no. 1, pp. 69–79, Jan 2008.

[16] A. Bogliolo, L. Benini, and G. De Micheli, "Regression-based rtl power modeling," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, no. 3, pp. 337–372, Jul. 2000.

[17] F. Klein, R. Leao, G. Araujo, L. Santos, and R. Azevedo, "A multi-model engine for high-level power estimation accuracy optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 5, pp. 660–673, May 2009.

[18] M. Rogers-Valle, M. A. Cantin, L. Moss, and G. Bois, "Ip characterization methodology for fast and accurate power consumption estimation at transactional level model," in *2010 IEEE International Conference on Computer Design*, Oct 2010, pp. 534–541.

[19] B. Ko, J. Kim, J. Ryoo, C. Hwang, J. Song, and S. W. Kim, "Simplified chip power modeling methodology without netlist information in early stage of soc design process," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 6, no. 10, pp. 1513–1521, Oct 2016.

[20] M. Ahmed, "On bootstrap estimation of system parameters and states," *IEEE Transactions on Automatic Control*, vol. 28, no. 7, pp. 805–806, Jul 1983.

[21] E. Bocchieri, *Fixed-Point Arithmetic*. London: Springer London, 2008, pp. 255–275.

[22] O. Project, "Mor1kx Cappuccino: OpenRISC compliant SoC," 2016. [Online]. Available: https://github.com/openrisc/mor1kx

[23] OpenCores, "Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores," 2016.

[24] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The malardalen wcet benchmarks: Past, present and future." in *WCET*, ser. OASICS, B. Lisper, Ed., vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010, pp. 136–146.