Efficient Runtime Power Modeling with On-Chip Power Meters

Invited Paper

Zhiyao Xie
Hong Kong University of Science and Technology
eezhiyao@ust.hk

ABSTRACT

Accurate and efficient power modeling techniques are crucial for both design-time power optimization and runtime on-chip IC management. In prior research, different types of power modeling solutions have been proposed, optimizing multiple objectives including accuracy, efficiency, temporal resolution, and automation level, targeting various power/voltage-related applications. Despite extensive prior explorations in this topic, new solutions still keep emerging and achieve state-of-the-art performance. This paper aims at providing a review of the recent progress in power modeling, with more focus on runtime on-chip power meter (OPM) development techniques. It also serves as a vehicle for discussing some general development techniques for the runtime on-chip power modeling task.

CCS CONCEPTS

• **Hardware** → **Power estimation and optimization**; *On-chip resource management*;

KEYWORDS

Power modeling and estimation, on-chip power meter, machine learning, voltage droop

1 INTRODUCTION

Power efficiency has become one of the primary design objectives for modern compute systems, ranging from low-end embedded systems, mobile computing to high-end data centers. Accurate while efficient power estimation is not only essential for *design-time* hardware design decisions, but also vitally important for power, energy, and voltage management during circuit *runtime*.

For the *design-time* power modeling, it provides power evaluation as feedback to designers and thus enables power optimizations in the design flow. Accurate power simulations rely on industry-standard power analysis tools [45, 48] to replay simulation vectors at the gate level with back-annotated parasitics. Besides standard simulations, many other works propose earlier-stage and faster design-time power estimations at the micro-architectural level [4, 58] or the register-transfer level (RTL) [7, 60]. Some of them [58, 60] are based on data-driven and machine learning techniques.

For the *runtime* circuit management, it needs to support various runtime circuit management techniques. For instance, runtime on-chip power meters (OPMs) can guide dynamic voltage and frequency scaling (DVFS) [17, 21]. The DVFS only requires coarse-grained temporal resolution in power tracing, where each sample represents power for epochs that can be microseconds in duration. Some runtime OPMs also support fast power management [14, 25] and voltage boosting [18]. These applications require more fine-grained temporal resolution. For instance, voltage-noise effects such as Ldi/dt noise develops in less than 10 cycles. Mitigating the impact of fast voltage noise requires fine-grained temporal resolution for the runtime power modeling.

Different types of runtime OPMs have been proposed to address aforementioned various requirements from different applications. Runtime OPMs based on hardware performance counters [1, 2, 6] provide low-cost solutions for guiding DVFS, but suffer from limited temporal resolution, rendering them unsuitable for other applications like voltage management. In comparison, recent works propose data-driven techniques [52, 53] to develop OPMs by automatically selecting on-chip signals (i.e. proxies) in the target design. They claim per-cycle temporal resolution as well as low hardware overhead.

In summary, power modeling is undoubtedly an increasingly important research topic. Despite extensive prior explorations, new techniques still keep emerging and achieve state-of-the-art performance. Considering the wide variety of application scenarios, power model objectives, and types of solutions, a review of existing power modeling technologies will help mitigate confusion and add understanding of

ISPD '23, March 26–29, 2023, Virtual Event, USA
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9978-4/23/03…$15.00
https://doi.org/10.1145/3569052.3578927

Power Model Types	Accuracy Level ↑	Hardware Overhead if can be Implemented on Hardware ↓	Automation Level ↑	Temporal Resolution ↑	Application Scenario
Standard power simulation [10, 45, 48]	High	N/A (Slow in Simulation)	High	High	Design-time
Micro-architectural-level estimation [8, 21, 30, 33, 41]	Low	N/A (Fast in Simulation)	Medium	N/A or High	
RTL power estimation [7, 28, 29, 31, 44, 50, 54, 57, 59, 60]	Medium	High	High	N/A or High	
Design-time emulation [11, 26, 47, 56]	Medium	Medium to High	High	Medium to High	
Counter-based OPMs [1, 2, 6, 13, 15, 19, 20, 22, 24, 38, 40, 42, 43, 46, 49]	Medium	Low	Low	Low	Runtime
Proxy-based OPMs [12, 35, 39, 52, 53, 61, 62]	Medium	Low to Medium	High	Medium to High	

Table 1: An analysis of different types of power models. Please notice that it only represents the author's opinion on the most general cases and exceptions often exist. 'N/A' in the *Hardware Overhead* column means the design-time models are rarely implemented on hardware. We mark their simulation speed in parentheses. 'N/A' in the *Temporal Resolution* column means it only generates one averaged power for the whole testbench or vectorless power.

relevant techniques to our research community. Prior power modeling survey papers [36, 37] mostly focus on design-time methods without covering runtime power models. This work targets providing a review of both design-time and runtime power modeling methods, with a focus on representative proxy-based runtime OPM solutions. It will cover a general development flow of proxy-based runtime OPM solutions and some rules of thumb during the development.

2 POWER MODELING OVERVIEW

2.1 Power Model Objectives

The design of power models involves multiple objectives. These objectives can be summarized below.

- **Accuracy:** Power modeling accuracy is the most fundamental objective. The accuracy can be measured by the correlation or absolute error between power estimations and ground-truth power values from accurate simulation or hardware measurements.
- **Automation-level:** In practice, many power models heavily rely on human expertise when applied to new circuit designs, while some others can be fully automatically applied. The level of automation can be measured by the amount of effort and expertise required to apply the power model to a new design.
- **Temporal resolution:** Temporal resolution decides suitable applications of a power model. The measurement window size is a hyper-parameter, which typically ranges from single-cycle to multi-microseconds. The power model will provide averaged power within each timing window.

The above three objectives generally apply to both design-time and runtime power models. Next, we further summarize the different objectives for the two types of models.

- **Hardware overhead (runtime):** Runtime OPMs will finally be implemented on the hardware as part of the target circuit. Therefore, the hardware overhead of this OPM implementation is a key objective. Such overhead includes area overhead, power overhead, and impact on the overall design performance.
- **Simulation speed (design-time):** For design-time power simulation models, a short simulation time is preferable to provide fast feedback to designers for design optimization.

Such different objectives lead to different design philosophies between runtime and design-time power models. In order to reduce the hardware implementation overhead, runtime power models are generally simpler, more coarse-grained in temporal resolution, and hardware-friendly. In comparison, design-time simulation can be more complex, primarily targeting higher accuracy, as long as the speed is acceptable. Also, different from the limited on-chip resources of runtime models, design-time simulation methods can execute on much more powerful computation platforms like high-performance servers, GPUs, FPGAs, and emulators. This alleviates the requirement on the speed of design-time simulation methods.

Obvious trade-offs exist among the aforementioned objectives. Take runtime power models as an example, more complex power models with more inputs generally can achieve higher accuracy, but will lead to larger hardware overhead when implemented on-chip. Also, for coarser-grained temporal resolution, the average power over a large power measurement window will cancel out minor per-cycle modeling errors, leading to higher accuracy values [52, 53]. In addition, more human engineers' expertise can customize power models for each new design. This may lead to better accuracy and hardware cost, but significantly reduces the automation level. Considering all these trade-offs, good power models should jointly optimize multiple objectives.

Next, we will briefly summarize representative and recent works on both design-time and runtime power modeling. For design-time power simulation, we separate models running on normal computation servers and emulators. For runtime on-chip power modeling, besides a few analog solutions [3], popular previous methods can be categorized into two major types, *counter*-based and *proxy*-based. Table 1 provides a summary of representative power models and a qualitative analysis of their strength in each objective.

2.2 Design-time Power Simulation

We start with introducing design-time power simulations.

Accurate standard power simulation. The most accurate power simulation can be performed with industrial-standard tools [10, 45, 48]. These tools replay simulation vectors on the target design with back-annotated parasitics. They support simulation at different design stages, including register-transfer level (RTL), gate-level netlist, and layout. The power is computed from the switching statistics of individual signal nets and the capacitive load that they drive. Such simulation at the final layout stage is accurate enough to serve as the signoff standard, but it comes with a high computational cost.

Micro-architectural-level power estimation. Many design-time approaches [8, 21, 30, 33, 41] construct analytical models for micro-architectural power estimation by collecting statistics from performance simulators [4, 5]. Wattch [8] is an architectural dynamic power simulation tool using a linear model, and McPAT [33] integrates power, area, and timing in a modeling framework. Each functional unit is characterized and attributed a power value when activated. Multiple active units are then added together to compute the overall power [16]. These models are preferably used as an average over thousands or millions of CPU clock cycles. Additionally, inaccuracies have been observed [41, 51] for McPAT and ML-based solutions [32, 58] are proposed to calibrate the original McPAT for better accuracy.

Register-transfer-level power estimation. Compared with architecture-level, power modeling is generally more accurate at the register-transfer level (RTL), when more design details become available. Early works [7, 50, 57] construct macro-models to abstract power estimations for small circuit modules with thousands of gates. In recent years, machine learning (ML) techniques are exploited [28, 29, 31, 44, 54, 59, 60]. PRIMAL [60] predicts per-cycle power by processing transitions of all registers with the convolutional neural network (CNN). GRANNITE [59] makes use of graph neural network [27] to estimate the average power of each workload. Some latest works [44, 54] modeled the vectorless power number without specific testbenches, together with timing and area at the RTL stage. In addition, some ML-based works [28, 31] claim to be micro-architectural-level, but given existing techniques, they still require RTL simulation in real implementation.

2.3 Design-time Power Emulation

Emulation [11, 26, 47, 56] is a popular approach to accelerating power simulations for large designs. The term "emulation" is used in a broad sense to include techniques that deploy power models on FPGA or other emulator platforms at design-time. According to our taxonomy, they still belong to design-time power simulation techniques.

Since these design-time power models will be implemented as 'hardware' on emulation platforms, they share many similarities with proxy-based runtime power models. Therefore, some practitioners often confuse them. These design-time power models for emulation are indeed extensible to runtime power models when implemented as part of the target design. But since they initially target emulation at design-time, they generally tolerate higher hardware overhead, as long as it does not exceed the limit of emulation platforms.

A pioneering power emulation work [11] has 300% hardware overhead. Another work [56] employs singular value decomposition (SVD), which can be computationally expensive. Both [11] and [56] are demonstrated only at block-level designs. A microprocessor-level application of FPGA emulation is Simmani [26], whose temporal resolution is 128 clock cycles. PrEsto [47] achieves cycle-accuracy, but its hardware cost is also quite significant (e.g. 50% of LUTs).

2.4 Counter-based Runtime Power Models

Different from design-time simulation, minimizing hardware overhead is a key objective when developing runtime power models. A classical runtime power modeling approach is to estimate power dissipation based on performance counters [1, 2, 6, 13, 15, 19, 20, 22, 24, 38, 40, 42, 43, 46, 49]. These counter-based runtime power models utilize already existing performance counters in target designs like microprocessors or digital signal processors (DSPs). Such counters can be treated as free and the associated area overhead is minimum. However, the development of such counter-based power models typically requires extensive designer's knowledge of the specific design to define related hardware events. It significantly restricts the automation level of these power modeling methods. Moreover, these counter-monitored hardware events manifest multiple cycles after the causal trigger event. It restricts the temporal resolution of estimations to thousands to millions of cycles.

2.5 Proxy-based Runtime Power Models

Instead of relying on existing performance counters, proxy-based runtime power models [12, 35, 39, 52, 53, 61, 62] are free to select the most power-correlated signals in the target design as power model inputs (i.e. power proxies). Due to

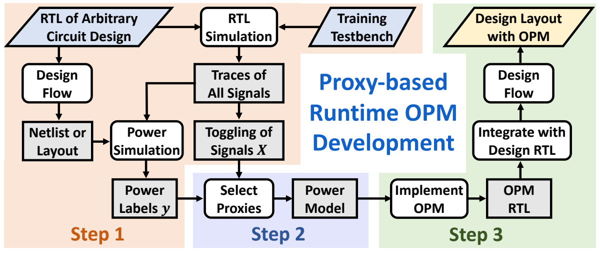

Figure 1: The general framework for proxy-based runtime OPM development. Image adapted from [52].

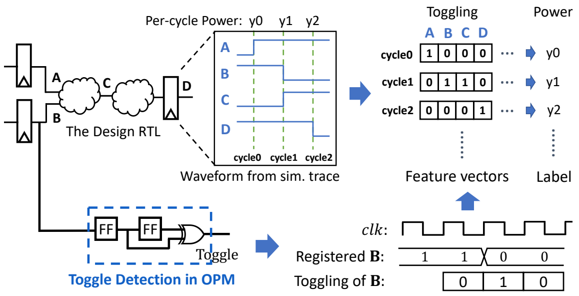

Figure 2: Toggling activities of signals as runtime OPM proxies. Image adapted from [52, 53].

such higher flexibility, these models have a higher potential to better optimize the multiple objectives. On the other hand, since these methods face a larger number of power model candidates, optimizing power models in this huge space is more challenging.

Some earlier proxy-based models [12, 35, 39] are coarse-grained with the temporal resolution of thousands of cycles. Their area overhead ranges from 1.5% to 20%. Some other methods [61, 62] improve temporal resolution to 100 cycles. They restrict proxies mostly to primary I/O signals of design modules at the selected hierarchy level, significantly reducing the freedom of proxy selection and the underlying power model. Their area overhead is still $> 4\%$ [61, 62]. In [23], a manually-designed digital power meter technique is introduced to address voltage-droop in DSP engines. It takes advantage of predictable dataflow patterns that are not available for general-purpose CPUs.

Recently, APOLLO [53] claims to achieve $< 1\%$ area overhead for per-cycle on-chip power modeling. This work allows flexible selection from all available RTL signals in the target design. It is validated on cutting-edge commercial microprocessors. A most recent work DEEP [52] further reduces the hardware overhead to be 0.1% by selecting individual bits from the target design and improving the proxy selection algorithms.

It is worth noting that some proxy-based runtime power models can also be applied for design-time power simulation. Due to their simplicity and hardware-friendly design, they can achieve an extremely high estimation speed on both simulation and emulation platforms. For instance, APOLLO [53] claims unprecedented simulation speed (i.e. per-cycle power estimation on millions of cycles in minutes, including RTL simulation) on emulation platforms [9].

3 PROXY-BASED OPM DEVELOPMENT

As summarized in the previous overview section, proxy-based runtime OPMs demonstrate great potential in optimizing multiple objectives including accuracy, hardware overhead, automation, and temporal resolution. New proxy-based solutions keep emerging in the latest works. In this section, we introduce the general development flow or framework of proxy-based OPMs.

3.1 OPM Development Framework

We first introduce a general and widely-adopted framework or flow in developing proxy-based runtime OPM. As shown in Figure 1, it mainly includes three major stages.

- **Step 1. Dataset Generation:** Given an arbitrary design RTL, collect corresponding testbenches, then generate signal waveforms and ground-truth power values through simulation.
- **Step 2. Power Model Development:** Based on the generated dataset, a power model is developed for the given design with data-driven techniques.
- **Step 3. Hardware Implementation:** The power model is implemented on hardware as the OPM and integrated as part of the target design.

3.2 Dataset Generation as Step 1

To develop OPMs with data-driven methods, dataset generation is the first step. It generates toggling activities $X \in \mathbb{R}^{N\times M}$ and corresponding power values $y \in \mathbb{R}^N$. N is the total number of cycles, and M is the number of power model input candidates. This setup is illustrated in Figure 2.

Testbench generation. Power consumption is dependent on the testbench/workload that executes on the target design. Therefore, the flow starts with collecting sufficient testbenches of the target design. The collection of sufficient representative testbenches is not a simple task. Many prior works do not disclose details about how they generate testbenches for each design. APOLLO [53] proposes to automatically generate random testbenches for the training dataset. In this process, an evolutionary algorithm is adopted to produce high-power-consumption testbenches [53].

It is difficult to judge whether generated testbenches are sufficient for accurate power model development. Ideally, generated testbenches as training data should cover the diverse behavior reflected in togging activity in test scenarios (i.e. coverage in feature space). But it is difficult to achieve such coverage due to the huge number of signals in target designs. The author's rule of thumb is to generate at least thousands of power samples from as many different testbenches as possible. More importantly, make sure the maximum and minimum power values of training testbenches cover the range of power consumption in target test scenarios (i.e. coverage in label space). Such test scenarios include all realistic workloads that may execute on the target design hardware in the future.

Toggling activity collection. By simulating generated testbenches on the target design RTL, per-cycle toggling activity/waveform can be generated and saved in *.fsdb* or *.vcd* file format. These collected toggling activities cover all available RTL signals in the design. As we know, the dynamic power is linearly proportional to the charging or discharging of gate- and wire-capacitance, which is reflected in signal transitions/toggling. Due to signal toggling activities' high correlation with dynamic power consumption, they are often used as input candidates X to power models.

Ground-truth power generation. Ground-truth power values y are necessary to train data-driven power models. Based on toggling activities, power values can be generated by accurate power simulation with industrial standard tools [10, 45, 48] or even hardware measurement. As mentioned, such simulation at gate-level with back-annotated parasitics produces accurate power values.

Training and testing data split. After data generation, the dataset needs to be split into training and testing dataset. For simplicity, we do not clearly distinguish validation set and testing set here. The testing dataset is used for evaluating the power model performance. Designers should make sure it reflects actual workloads that execute on real hardware.

3.3 Power Model Development as Step 2

After data generation, the power model design is the key step and distinguishes different prior works. As mentioned, both accuracy and hardware overhead are essential objectives. To minimize the hardware overhead while maximizing accuracy, the power model development process should borrow ideas from the hardware-software co-design.

Model selection. To reduce hardware overhead, many prior OPM solutions [26, 52, 53, 62] are based on the linear power model due to its simplicity. Some others [34] adopt fast non-linear models like the decision tree, with customized hardware implementations to reduce overhead.

Temporal resolution selection. Another key power model configuration is its temporal resolution (i.e. size of the power-measurement timing window). Its selection depends on the target runtime application using this power model. As mentioned, coarse-grained estimation is sufficient for DVFS decisions, while fast power management or voltage-droop mitigation may require fine-grained temporal resolution.

Hardware overhead evaluation. During the OPM development process, the model accuracy can be easily measured based on generated datasets. In comparison, the hardware overhead evaluation is not as straightforward without real hardware implementation and simulation, which can be time-consuming.

A common practice is to simply assume a linear correlation between the number of proxies and hardware cost. Doing this simplifies the hardware overhead optimization problem to minimizing the number of model inputs. For a linear power model, it means minimizing the number of (i.e. the l_0 norm of) weights, which is an existing optimization problem.

Despite the simplicity, the number of proxies cannot fully represent OPM hardware cost. When using signals as proxies, different signals' bit width leads to significantly different hardware costs [52]. More importantly, the actual overhead also depends on the specific OPM hardware design, including the arithmetic logic design, the number of bits in weights, and the physical locations of selected proxies. Therefore, better hardware overhead evaluation method is desired.

Model input selection. Selecting appropriate model inputs (i.e. proxies) is the most important step in developing proxy-based power models. This process involves two key decisions. First, developers should define the candidate proxies in the target design. Second, developers should choose appropriate algorithms to select proxies from candidates.

The proxy candidate setup decides the solution space of proxy selection. As mentioned, more proxy candidates lead to a more flexible OPM solution. As the summary of prior works in Table 2 shows, we can observe a trend in using an increasing number of available signals as proxy candidates.

The proxy-selection algorithm is the key part of the OPM development process and directly decides the ultimate OPM accuracy and overhead. Already explored algorithm types include but not limited to clustering [26], heuristic-based selection [62], pruning [52, 53], best subset selection [52]. Their relative advantage may vary slightly for different target designs and workloads, thus should be thoroughly verified with testing datasets. Here we provide some personal experience in proxy selection methods. First, a supervised selection of proxies is more efficient than unsupervised methods like clustering. Second, the algorithm should reduce the collinearity or correlation among selected proxies. In prior works [52, 53], the feature collinearity is measured with metrics like variance inflation factor (VIF) [55].

Works	Proxy Candidate	Temporal Resolution	Claimed Overhead
[56] 2015	Registers	Per-cycle	16%
[39] 2018	Registers	> 1K cycles	7%
[62] 2018	Module I/O signals	100s cycles	$4-10\%$
[26] 2019	All RTL signals	100s cycles	N/A
[12] 2020	Module I/O signals	>1K cycles	2-20%
[53] 2021	All RTL signals	Per-cycle	< 1%
[52] 2022	All bits of RTL signals	Per-cycle	< 0.1%

Table 2: Overview of representative works for proxy-based runtime OPMs. Some works [26, 56] are initially proposed for design-time emulation.

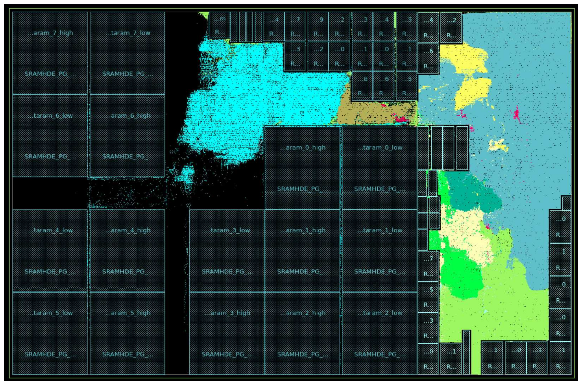
Figure 3: Microprocessor layout with DEEP [52] OPM integrated. The red region is OPM. Area overhead is 0.04%. The MAE = 9.5%. Adapted from [52].

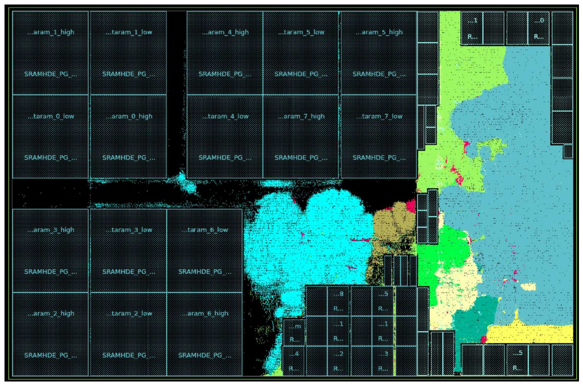
Figure 4: Microprocessor layout with APOLLO [53] OPM integrated. The red region is OPM. Area overhead is 0.16%. The MAE = 9.5%. Adapted from [52].

3.4 Hardware Implementation as Step 3

After the design and development of a power model, it needs to be incorporated into the original design in hardware.

Reduction of hardware cost. Minimizing the hardware overhead is the most important goal in this step. Weight quantization is a commonly adopted technique during hardware implementation of the OPM [52, 53, 56]. According to APOLLO [53], if using a fixed bit number for all weights, adopting 10 bits leads to negligible accuracy loss. For variable bit width, DEEP [52] supports most weights less than 7 bits and [56] supports most weights less than 9 bits.

Another popular design option is to reduce the usage of multipliers and counters [39, 52, 53] in the OPM hardware. By adopting binary values for signal toggling activities X, multiplying weights with binary toggling will only require AND gates, without using multipliers and counters. This hardware implementation setup requires co-design with the power model design in step 2. Please notice such a binary proxy setup is not limited to per-cycle temporal resolution. The solution for multi-cycle resolution is given in [53].

Integration with target design. The OPM ultimately needs to be integrated as part of the hardware design. A common practice is to simply initiate the OPM in the top module, with selected signals from the target design RTL (i.e. proxies) connected to the OPM RTL. In practice, this may require some engineering efforts in revising target design's module interfaces to expose the selected proxies to OPM. Then the whole design is implemented by going through the original design flow. The OPM hardware overhead can be accurately measured after the layout finishes.

4 PERFORMANCE OVERVIEW

In this section, we briefly discuss the performance of recent works for runtime OPM. Please notice that due to the significant difference between the target design and testbench, claimed performance numbers are not directly comparable.

Table 2 summarizes recent representative works for developing runtime proxy-based OPMs. Some design-time emulation works [26, 56] are included since they can be easily extended as runtime OPMs. An obvious improvement over time can be observed. In Table 2, the claimed overhead improves down to 0.1% and the temporal resolution reaches per-cycle, the finest granularity in resolution. Such improvement benefits from the increasing proxy candidates and better proxy selection algorithm.

Finally, we present the visualized OPM solutions after integration with the target industrial design. Here we focus on the most recent and state-of-the-art solutions [52, 53]. The layout of the target microprocessor with DEEP [52] OPM design is shown in Figure 3. In this layout, all cells in the OPM are colored in red. The large macros are L2 data RAMs and small macros are for L1 cache, TLB, and tag RAMs. The overhead is 0.04%.

In comparison, the layout of the same microprocessor with APOLLO [52] OPM is shown in Figure 4. Note that the macro locations are not fixed and are automatically placed by IC Compiler II, leading to a slightly different floorplan solution. The overhead in Figure 4 is 0.16%.

5 CONCLUSION

In this paper, we present a review of recent power modeling techniques. It covers both design-time and runtime models, with our taxonomy and qualitative analysis of the strength of each type of power modeling solution. In addition, we introduce the general framework for developing runtime proxy-based OPMs, which effectively co-optimize multiple objectives, including accuracy, hardware cost, temporal resolution, and the design automation level.

6 ACKNOWLEDGEMENT

This work was supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Frank Bellosa. 2000. The benefits of event: driven energy accounting in power-sensitive systems. In *ACM SIGOPS European Workshop (EW)*.
[2] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. 2010. Decomposable and responsive power models for multicore processors using performance counters. In *ACM ICS*.
[3] Srikar Bhagavatula and Byunghoo Jung. 2013. A power sensor with 80ns response time for power management in microprocessors. In *CICC*.
[4] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* (2011).
[5] Nathan L Binkert, Ronald G Dreslinski, Lisa R Hsu, Kevin T Lim, Ali G Saidi, and Steven K Reinhardt. 2006. The M5 simulator: Modeling networked systems. *IEEE Micro* (2006).
[6] W Lloyd Bircher and Lizy K John. 2007. Complete system power estimation: A trickle-down approach based on performance events. In *IEEE ISPASS*.
[7] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. 2000. Regression-based RTL power modeling. *ACM TODAES* (2000).
[8] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News* (2000).
[9] Cadence. 2021. Palladium® Z1 Enterprise Emulation Platform. https://www.cadence.com/en_US/home/tools/system-design-and-verification/acceleration-and-emulation/palladium-z1.html.
[10] Cadence. 2022. Joules RTL Power Solution. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html.
[11] Joel Coburn, Srivaths Ravi, and Anand Raghunathan. 2005. Power emulation: a new paradigm for power estimation. In *DAC*.
[12] Luca Cremona, William Fornaciari, and Davide Zoni. 2020. Automatic identification and hardware implementation of a resource-constrained power model for embedded systems. *Elsevier Sustainable Computing: Informatics and Systems* (2020).
[13] C Gilberto and M Margaret. 2005. Power prediction for intel xscale processors using performance monitoring unit events. In *ISLPED*.
[14] Waclaw Godycki, Christopher Torng, Ivan Bukreyev, Alyssa Apsel, and Christopher Batten. 2014. Enabling realistic fine-grain voltage scaling with reconfigurable power distribution networks. In *MICRO*.
[15] Bhavishya Goel, Sally A McKee, Roberto Gioiosa, Karan Singh, Major Bhadauria, and Marco Cesati. 2010. Portable, scalable, per-core power estimation for intelligent resource management. In *International Conference on Green Computing (IGCC)*.
[16] Ed Grochowski, David Ayers, and Vivek Tiwari. 2002. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *HPCA*.
[17] Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben Asher, and Avi Mendelson. 2016. Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems. *TACO* (2016).
[18] Chang-Hong Hsu, Yunqi Zhang, Michael A Laurenzano, David Meisner, Thomas Wenisch, Jason Mars, Lingjia Tang, and Ronald G Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *HPCA*.
[19] Wei Huang, Charles Lefurgy, William Kuk, Alper Buyuktosunoglu, Michael Floyd, Karthick Rajamani, Malcolm Allen-Ware, and Bishop Brock. 2012. Accurate fine-grained processor power proxies. In *MICRO*.
[20] Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*.
[21] Hans Jacobson, Alper Buyuktosunoglu, Pradip Bose, Emrah Acar, and Richard Eickemeyer. 2011. Abstraction and microarchitecture scaling in early-stage power modeling. In *HPCA*.
[22] R. Joseph and M. Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *ISLPED*.
[23] Vijay Kiran Kalyanam, Eric Mahurin, Keith Bowman, and Jacob Abraham. 2020. A Proactive Voltage-Droop-Mitigation System in a 7nm Hexagon™ Processor. In *VLSI*.
[24] Vijay Kiran Kalyanam, Peter G Sassone, and Jacob A Abraham. 2017. Power prediction of embedded scalar and vector processor: Challenges and solutions. In *ISQED*.
[25] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *MICRO*.
[26] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection. In *MICRO*.
[27] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[28] Ajay Krishna Ananda Kumar, Sami Alsalamin, Hussam Amrouch, and Andreas Gerstlauer. 2022. Machine learning-based microarchitecture-level power modeling of CPUs. *IEEE Trans. Comput.* (2022).
[29] Ajay Krishna Ananda Kumar and Andreas Gerstlauer. 2019. Learning-Based CPU Power Modeling. In *MLCAD*.
[30] Benjamin C Lee and David M Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *ACM SIGOPS operating systems review* (2006).
[31] Dongwook Lee, Lizy K John, and Andreas Gerstlauer. 2015. Dynamic power and performance back-annotation for fast and accurate functional hardware simulation. In *DATE*.
[32] Wooseok Lee, Youngchun Kim, Jee Ho Ryoo, Dam Sunwoo, Andreas Gerstlauer, and Lizy K John. 2015. PowerTrain: A learning-based calibration of McPAT power models. In *ISLPED*.
[33] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*.
[34] Zhe Lin, Sharad Sinha, and Wei Zhang. 2018. An ensemble learning approach for in-situ monitoring of FPGA dynamic power. *IEEE TCAD* (2018).
[35] Mohamad Najem, Pascal Benoit, Mohamad El Ahmad, Gilles Sassatelli, and Lionel Torres. 2017. A design-time method for building cost-effective run-time power monitoring. *IEEE TCAD* (2017).
[36] Farid N Najm. 1994. A survey of power estimation techniques in VLSI circuits. *IEEE VLSI* (1994).
[37] Yehya Nasser, Jordane Lorandel, Jean-Christophe Prévotet, and Maryline Hélard. 2020. RTL to transistor level power modeling and estimation techniques for FPGA and ASIC: A survey. *IEEE TCAD* (2020).
[38] Fabian Oboril, Jos Ewert, and Mehdi B Tahoori. 2015. High-resolution online power monitoring for modern microprocessors. In *DATE*.
[39] Daniele Jahier Pagliari, Valentino Peluso, Yukai Chen, Andrea Calimera, Enrico Macii, and Massimo Poncino. 2018. All-digital embedded meters for on-line power estimation. In *DATE*.
[40] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Power-performance modeling on asymmetric multi-cores. In *CASES*.
[41] Santhosh Kumar Rethinagiri, Oscar Palomar, Rabie Ben Atitallah, Smail Niar, Osman Unsal, and Adrian Cristal Kestelman. 2014. System-level power estimation tool for embedded processor based platforms. In *RAPIDO*.
[42] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. 2013. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs (TCAS-II)* (2013).

[43] Mark Sagi, Nguyen Anh Vu Doan, Martin Rapp, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf. 2020. A Lightweight Nonlinear Methodology to Accurately Model Multicore Processor Power. *IEEE TCAD* (2020).

[44] Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. 2022. How Good Is Your Verilog RTL Code? A Quick Answer from Machine Learning. In *ICCAD*.

[45] Siemens. 2022. PowerPro® RTL Low-Power. https://www.mentor.com/hls-lp/powerpro-rtl-low-power/.

[46] Karan Singh, Major Bhadauria, and Sally A McKee. 2009. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News* (2009).

[47] Dam Sunwoo, Gene Y Wu, Nikhil A Patil, and Derek Chiou. 2010. PrEsto: An FPGA-accelerated power estimation methodology for complex systems. In *FPL*.

[48] Synopsys. 2022. PrimePower: RTL to Signoff Power Analysis. https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html.

[49] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-Hashimi, and Geoff V Merrett. 2016. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE TCAD* (2016).

[50] Qing Wu, Qinru Qiu, Massoud Pedram, and Chih-Shun Ding. 1998. Cycle-accurate macro-models for RT-level power analysis. *VLSI* (1998).

[51] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *HPCA*.

[52] Zhiyao Xie, Shiyu Li, Mingyuan Ma, Chen-Chia Chang, Jingyu Pan, Yiran Chen, and Jiang Hu. 2022. DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters. In *ICCAD*.

[53] Zhiyao Xie, Xiaoqing Xu, Matt Walker, Joshua Knebel, Kumaraguru Palaniswamy, Nicolas Hebert, Jiang Hu, Huanrui Yang, Yiran Chen, and Shidhartha Das. 2021. APOLLO: An Automated Power Modeling Framework for Runtime Power Introspection in High-Volume Commercial Microprocessors. In *MICRO*.

[54] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. 2022. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In *ISCA*.

[55] Zhang Xuegong. 2000. Introduction to statistical learning theory and support vector machines. *Acta Automatica Sinica* (2000).

[56] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai, and Tin-Fook Ngai. 2015. Early stage real-time SoC power estimation using RTL instrumentation. In *ASPDAC*.

[57] Wu Ye, Narayanan Vijaykrishnan, Mahmut Kandemir, and Mary Jane Irwin. 2000. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *DAC*.

[58] Jianwang Zhai, Chen Bai, Binwu Zhu, Yici Cai, Qiang Zhou, and Bei Yu. 2021. McPAT-Calib: A microarchitecture power modeling framework for modern CPUs. In *ICCAD*.

[59] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. GRANNITE: Graph Neural Network Inference for Transferable Power Estimation. In *DAC*.

[60] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. PRIMAL: Power Inference using Machine Learning. In *DAC*.

[61] Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, and William Fornaciari. 2018. PowerTap: All-digital power meter modeling for run-time power monitoring. *Elsevier Microprocessors and Microsystems (MICPRO)* (2018).

[62] Davide Zoni, Luca Cremona, and William Fornaciari. 2018. Powerprobe: Run-time power modeling through automatic RTL instrumentation. In *DATE*.