A survey on run-time power monitors at the edge

DAVIDE ZONI*, Politecnico di Milano, Italy
ANDREA GALIMBERTI, Politecnico di Milano, Italy
WILLIAM FORNACIARI, Politecnico di Milano, Italy

Effectively managing energy and power consumption is crucial to the success of the design of any computing system, helping mitigate the efficiency obstacles given by the downsizing of the systems while also being a valuable step towards achieving green and sustainable computing. The quality of energy and power management is strongly affected by the prompt availability of reliable and accurate information regarding the power consumption for the different parts composing the target monitored system. At the same time, effective energy and power management are even more critical within the field of devices at the edge, which exponentially proliferated within the past decade with the digital revolution brought by the Internet of things. This manuscript aims to provide a comprehensive conceptual framework to classify the different approaches to implementing run-time power monitors for edge devices that appeared in literature, leading the reader toward the solutions that best fit their application needs and the requirements and constraints of their target computing platforms. Run-time power monitors at the edge are analyzed according to both the power modeling and monitoring implementation aspects, identifying specific quality metrics for both in order to create a consistent and detailed taxonomy that encompasses the vast existing literature and provides a sound reference to the interested reader.

CCS Concepts: • **Hardware** → **Power estimation and optimization**; • **Computer systems organization** → *Embedded systems*; • **General and reference** → Surveys and overviews.

Additional Key Words and Phrases: Power modeling, power monitoring, power management, run-time, run-time power modeling, run-time power monitoring, run-time power management, edge computing, survey

ACM Reference Format:
Davide Zoni, Andrea Galimberti, and William Fornaciari. 2023. A survey on run-time power monitors at the edge. *ACM Comput. Surv.* 1, 1, Article 1 (January 2023), 33 pages. https://doi.org/10.1145/3593044

1 INTRODUCTION

Edge computing brings data storage and processing at the edge of the network, i.e., physically closer to users, drastically reducing the communication bandwidth and latency and avoiding the need to send information which might be private, sensitive, or safety-critical [95, 98]. The edge computing paradigm mitigates therefore the concerns typical of cloud computing about the inefficient usage of costly, centralized resources, their environmental impact, and the privacy risks posed by storing users' data in remote places [5, 68]. Edge gadgets can often be battery-powered mobile devices or even resort to energy-harvesting solutions, and in general it is crucial to deal with their energy and power constraints. At the same time, as the applications and services they provide vary in complexity, edge gadgets comprise a wide and heterogeneous range of devices with different processing capabilities [50].

*Corresponding and leading author

Authors' addresses: Davide Zoni, davide.zoni@polimi.it, Politecnico di Milano, P.zza L. Da Vinci, Milano, Italy, 20133; Andrea Galimberti, andrea.galimberti@polimi.it, Politecnico di Milano, P.zza L. Da Vinci, Milano, Italy, 20133; William Fornaciari, william.fornaciari@polimi.it, Politecnico di Milano, P.zza L. Da Vinci, Milano, Italy, 20133.

The processing components of devices at the edge are in charge of efficiently performing a large variety of computation- and communication-intensive tasks within tight energy constraints [21], and satisfying the energy- and power-efficiency requirements is a key aspect of their design for two main reasons. On the one hand, the ever-stricter time-to-market constraints, that impose to accommodate a large variety of application scenarios, have been driving the design of over-provisioned computing platforms, causing a significant fraction of the available computing resources to be left unused in real-world scenarios. On the other hand, the requirements for newer and more complex heterogeneous applications impose pairing multi-core processors with specialized hardware accelerators, thus making the overall computing platform even more complex and difficult to optimize.

To this end, run-time optimization techniques and the related monitoring infrastructures are foreseen as the solution to optimize the energy efficiency of modern computing platforms. Indeed, the effectiveness of any run-time energy optimization technique is strongly related to the quality of the measurements or estimates of power consumption provided by a run-time power monitoring system. The latter leverages either a direct measurement strategy or an indirect estimation one. Direct methods ensure high accuracy at the cost of poor scalability, due to the need to adopt a mixed analog-digital design, and high implementation costs. Indirect methods leverage the correlation between the power consumption and a carefully selected set of the platform's run-time statistics to deliver periodic power estimates. In general, indirect methods are more scalable and cheaper due to the possibility of selecting the platform's statistics to design the power monitor at different level of abstractions, i.e., ranging from the architectural performance monitoring counters down to the switching activity of the microarchitecture.

Notably, the usage of run-time power monitors has been extensively studied within the scenarios of high performance computing (HPC) [82] and datacenters [54, 65], while several state-of-the-art contributions specifically target the design of run-time power monitors for high-performance CPUs [69] and GPUs [13]. These surveys highlighted the complex taxonomy in the fields of power monitoring systems for HPC as well as for high-performance CPUs and GPUs, while, in contrast, the large amount of work targeting run-time power monitors at the edge is still unstructured and is missing a comprehensive taxonomy. The ever-increasing needs to ease the design of effective monitoring systems and to better identify the areas that require further research make it paramount to deliver a complete and comprehensive classification of the current state-of-the-art solutions in the field of run-time power monitors at the edge.

Contributions. This manuscript provides a comprehensive taxonomy of the solutions proposed in the literature of run-time power monitors targeting the processing elements of edge devices, with the final goal of helping the reader select and implement the proper infrastructure fitting the computing platform at hand as well as the system and application requirements.

The manuscript is organized according to three key principles to maximize its readability. First, each analyzed run-time power monitor is explored in terms of both the power modeling and power monitoring aspects, discussing its quality metrics for each aspect and how they correlate to each other to produce the actual implementation. Second, the proposed run-time power monitor classification considers both software- and hardware-implemented monitoring solutions that target edge computing platforms, dividing the latter into three classes, i.e., CPUs, GPUs, and hardware accelerators. Hardware accelerators can be either human-designed or generated through high-level synthesis (HLS). Third, the analyzed power monitors are evaluated in terms of functional and non-functional requirements. Functional requirements include the quality of the estimate and the temporal resolution, while non-functional ones comprise the performance, area, and power overheads due to the addition of the monitoring infrastructure.

Structure of the manuscript. The rest of this survey is organized into five parts. Section 2 introduces the theoretical background of power consumption and the core principles guiding its reduction. Section 3 overviews the state-of-the-art proposals, while a detailed discussion on power modeling and monitoring aspects is provided in Section 4 and Section 5, respectively. Section 6 draws some concluding remarks draws some concluding remarks discussing this survey's key findings and highlighting future research directions in the field.

2 BACKGROUND

This section overviews the background on run-time power and energy management by giving definitions for energy and power consumption, highlighting the factors that impact the latter, and introducing the general structure of a run-time power and energy management framework, as well as the concepts of run-time power modeling and monitoring.

2.1 Power consumption in CMOS devices

The energy consumption E can be defined as the total amount of electricity used to perform some work by a computing platform over a specific period T. Differently, the power consumption P can be defined as the rate at which a system consumes electricity. The relationship between energy and power consumption can be formulated as shown in Equation (1).

$$E = \int_0^T P(t)\,dt \tag{1}$$

Energy consumption is indeed the integral of the power consumption over time, where power consumption $P(t)$ refers to a specific time instant t. Equivalently, power consumption can be defined as the time derivative of energy consumption, according to Equation (2).

$$P = \frac{dE}{dt} \tag{2}$$

Estimating the energy consumption associated with a computing task requires predicting its power consumption at each instant. Without loss of generality, all models, modeling, and monitoring methods concerning both power and energy consumption can thus be referred to as power models, power modeling, and power monitoring methods, respectively. In CMOS devices, power consumption P can be split into two distinct components, dynamic and static power consumption [18, 61], as reported in Equation (3).

$$P = P_{dynamic} + P_{static} \tag{3}$$

The two power components are additive, and, generally, both need to be addressed to optimize the overall power consumption. The dynamic component of power consumption $P_{dynamic}$ can be traced back primarily to switching power, which derives from the load capacitance charging and discharging during switches in CMOS gates [17]. During the charging phase of the CMOS gate, the output voltage swings from 0 to V_{DD}, dissipating energy taken from the power supply while, during the discharging phase, the energy stored in the load capacitance is dissipated. In both cases, some heat is generated. Switching power consumption is generally approximated as shown in Equation (4), where α is the switching activity, i.e., the number of switching transitions per clock cycle, C_L is the equivalent load capacitance at the gate output, V_{DD} is the supply voltage, i.e., the voltage at which the load capacitance charges, and f_{clk} is the operating clock frequency.

$$P_{dynamic} \approx \alpha \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \tag{4}$$

In contrast, the static component P_{static} of power consumption can be traced back to the power dissipated by leakage currents, which also flow when the device is inactive. There are multiple

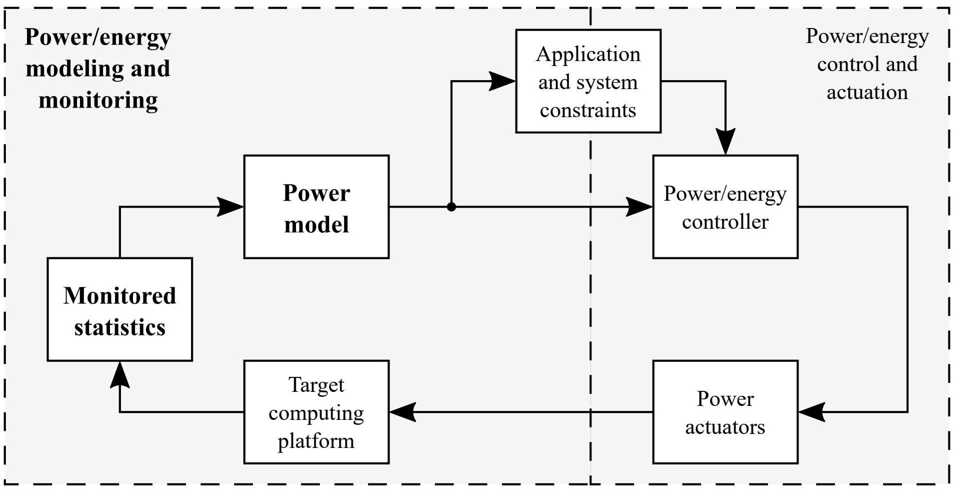

Power/energy modeling and monitoring
Application and system constraints
Power/energy control and actuation
Power model
Power/energy controller
Monitored statistics
Target computing platform
Power actuators

Fig. 1. Block diagram for a generic run-time power/energy management framework. This survey focuses on the blocks written in **bold**.

leakage sources, but the static power component due to the subthreshold leakage current is the most significant one, while the contribution of the others is negligible in devices that employ modern CMOS process technologies. The subthreshold leakage current derives from the transistor threshold voltage, which keeps decreasing as the node process technology advances and the transistor size shrinks. The transistor threshold voltage is reduced by design to offset the slower transistor dynamics as the supply voltage is scaled down, determining an exponential increase in the subthreshold leakage current. Static power P_{static} is approximately expressed as shown in Equation (5), where β and γ are technology-dependent constants, V_{dd} is the supply voltage, V_{th} is the threshold voltage, and V_T is the thermal voltage.

$$P_{static} \approx \beta \cdot V_{dd} \cdot e^{-\frac{V_{th}}{\gamma \cdot V_T}} \tag{5}$$

2.2 Run-time energy and power management frameworks

A generic power/energy run-time management framework is composed of a set of core blocks that interact with each other in a closed-loop fashion, as depicted in Figure 1. At a coarse grain, it can be split into two main functionalities, namely *i)* power/energy modeling and monitoring and *ii)* power/energy control and actuation.

This survey will focus on the power/energy modeling and monitoring part. In particular, a power model is fed a set of select monitored statistics measured or estimated from the target computing platform. The power model and the set of monitored statistics are generally defined at design time to fit the target computing platform and provide accurate estimates at the desired temporal resolution while requiring acceptable overheads of area, performance, and power due to the monitoring infrastructure. Power modeling and power monitoring are introduced in Section 2.3, while comprehensive overviews and taxonomies are provided in Section 4 and Section 5, respectively.

On the control and actuation side, the power consumption estimates produced at run-time by the power model, together with the constraints of the running applications of the overall system, are input to the power/energy controller, which produces control signals to maintain power/energy consumption close to the desired set-points, according to the implemented policies. Finally, the power actuators act on the target platform according to the inputs given by the power/energy controller, closing the power/energy management framework loop. Such actuators usually act on two main factors of dynamic power consumption, i.e., the supply voltage V_{DD} and the operating clock frequency f_{clk}. Actuation techniques exploited in run-time power management frameworks range from more complex dynamic voltage and frequency scaling (DVFS) [15, 44, 88, 100, 123] and dynamic frequency scaling (DFS) [64] techniques to simpler ones such as power gating [42, 46, 99, 113] and clock gating [26, 58, 117].

2.3 Power modeling and monitoring basics

Power modeling encompasses the design of models that can adequately estimate the power consumption of a target computing platform. A power model can generally be defined as a mathematical function that correlates the power consumption with features, measured or estimated at a certain level of abstraction, of the underlying hardware architecture. A power model can be employed mainly in two ways, i.e., as an input to either design-time power analysis or run-time power monitoring. Depending on the specific target application of a power model, it must satisfy different requirements and therefore be designed accordingly.

Design-time power analysis makes an offline use of a power model to explore the design space of the target computing platform, allowing to evaluate their energy, power, performance, and area quality metrics at an early design stage. *McPAT* [59], *DSENT* [106], *ORION* [49], and *Strober* [51] are examples of frameworks for design-time power analysis. The offline nature of power analysis carried out at an early design stage makes the performance of the underlying power model a non-critical factor. Such analysis adds no overhead in terms of performance, area, and power to the original hardware platform since it is carried out as part of the hardware design process. Notably, these frameworks often work at the architectural level and therefore employ a generic power model for multiple computing platforms, thus resulting in a low estimation accuracy [93, 118].

On the contrary, *run-time power monitoring* delivers an infrastructure to compute online estimates of the power consumption of the target computing platform. Employing them requires carefully considering the overhead in terms of performance, mainly for software implementations, and area and power, in particular for hardware implementations. In both scenarios, the run-time power monitors must be implemented in an ad-hoc manner, providing better accuracy than offline power analysis, at the cost of time spent designing the monitoring infrastructure.

3 RUN-TIME POWER MONITORS

This section presents the classification of the state-of-the-art run-time power monitors targeting edge devices, with the goal of providing the reader with a comprehensive overview of the available state-of-the-art contributions. In particular, Section 3.1 presents the dimensions considered for the classification, while Section 3.2 discusses the actual classification of the contributions surveyed within this manuscript. Notably, a detailed discussion of the run-time power modeling and monitoring aspects for each state-of-the-art contribution is provided in Section 4 and Section 5, respectively, while Section 6 presents the key findings as well as the future research directions.

3.1 Taxonomy dimensions

Each state-of-the-art proposal is classified and discussed according to four independent dimensions, i.e., target platforms, model families, power model statistics, and monitoring implementation, that are defined in the rest of this subsection.

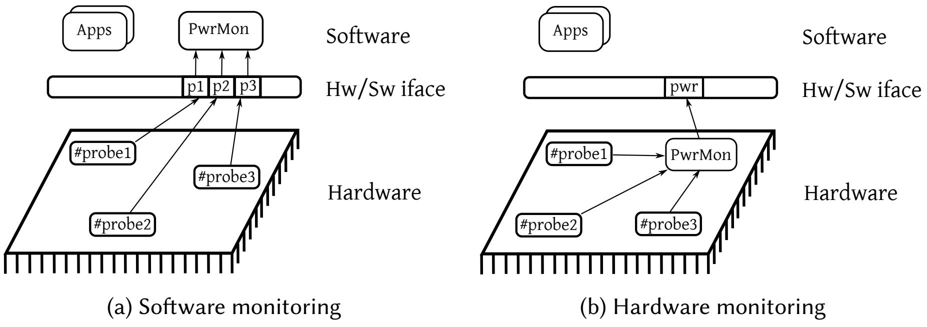


Fig. 2. Generic hardware and software implementations of run-time power monitors

3.1.1 Target platforms. The survey considers run-time power monitors that target the computing elements of edge devices, distinguishing between CPUs, GPUs, and hardware accelerators. *General-purpose central processing units* (**CPU** in Table 1) may be either single- or multi-core, possibly implementing multi-threading and heterogeneous architectures. *Graphics processing units* (**GPU**) accelerate highly parallel tasks and are employed in high-end embedded systems. *Hardware accelerators* (**Accel.**) provide dedicated hardware support to computationally-expensive functionalities and might be either hand-written or obtained as the output of the high-level synthesis (HLS) procedure [72].

3.1.2 Model families. In general, power models are either regression or machine learning models. *Regression models* (**Regression** in Table 1) are commonly used in run-time power monitoring scenarios due to their simple and low-overhead implementation. Regression may be either linear, polynomial or other more complex variants. In such regression models, power statistics are the independent variables while power consumption is the dependent one. *Machine learning models* (**Learning** in Table 1) implement the power model as a neural network, a decision tree, or a random forest.

3.1.3 Model statistics. Feeding the power model with optimal input statistics is of paramount importance for the sake of accuracy and effectiveness in modeling the power consumption of the target platform when considering real-world scenarios. Such input statistics can be split into two coarse-grained categories, i.e., performance events and switching activity. *Performance events* (**Perf. events** in Table 1) are commonly provided by general-purpose programmable platforms such as CPUs and GPUs through so-called performance monitoring counters (PMCs). The events collected by these counters, such as the number of instructions, branches taken, and cache misses, were initially exposed with the goal of monitoring performance in CPUs and GPUs, while their good correlation with the power consumption extended their use to power monitoring scenarios. On the other hand, the switching activity of each signal contributes to the dynamic power consumption of hardware platforms realized in CMOS technology, thus making the *toggling activity* (**Toggl. activity** in Table 1) of specific signals an alternate candidate input statistic for run-time power models. Notably, some state-of-the-art solutions also make use of additional statistics (**Others** in Table 1) such as voltage, frequency, and temperature measurements as inputs to the proposed run-time power models.

Table 1. Overview of the state-of-the-art contributions discussed in this survey and their classification according to the modeling and monitoring dimensions. Legend: ✓yes, – no; **Target** target platforms (see Section 3.1.1); **Family** model families (see Section 3.1.2); **Statistics** model statistics (see Section 3.1.3); **Implem.** monitoring implementation (see Section 3.1.4).

				Modeling					**Monitoring**	
	Target			**Family**		**Statistics**			**Implem.**	
Ref.	**CPU**	**GPU**	**Accel.**	**Regression**	**Learning**	**Perf. events**	**Toggl. activity**	**Others**	**Softw.**	**Hardw.**
[9]	✓	–	–	Linear regr.	–	✓	–	–	✓	–
[48]	✓	–	–	Linear regr.	–	✓	–	–	✓	–
[43]	✓	–	–	Linear regr.	–	✓	–	–	✓	–
[94]	✓	–	–	Linear regr.	–	✓	–	–	✓	–
[90]	✓	–	–	Linear regr.	–	✓	–	–	✓	–
[111]	✓	–	–	Linear regr.	–	✓	–	Voltage, freq., CPI	✓	–
[77]	✓	–	–	Linear regr.	–	✓	–	Voltage, freq., temp., state	None	
[76]	✓	–	–	Linear regr.	–	✓	–	Freq.	None	
[75]	✓	–	–	Linear regr.	–	✓	–	Freq.	None	
[74]	✓	–	–	Linear regr.	–	✓	–	–	None	
[86]	✓	–	–	Linear regr.	–	–	✓	–	–	✓
[71]	✓	–	–	Linear regr., MARS	Neural netw.	–	✓	–	–	✓
[126]	✓	–	–	Linear regr.	–	–	✓	–	–	✓
[125]	✓	–	–	Linear regr.	–	–	✓	–	–	✓
[120]	✓	–	–	Linear regr.	–	–	✓	–	–	✓
[119]	✓	–	–	Linear regr.	–	–	✓	–	–	✓
[70]	–	✓	–	Linear regr.	–	✓	–	–	✓	–
[20]	–	✓	–	–	Random for.	✓	–	–	Virtual	
[102]	–	✓	–	–	Neural netw.	✓	–	–	✓	–
[66]	–	✓	–	–	Neural netw.	✓	–	–	✓	–
[78]	–	✓	–	Linear regr.	–	✓	–	Voltage, freq., temp.	✓	–
[47]	–	–	✓	Linear regr.	–	–	✓	–	–	✓
[60]	–	–	✓	–	Decis. tree	–	✓	–	–	✓
[83]	✓	–	✓	Linear regr.	–	–	✓	–	✓	✓
[52]	✓	–	✓	Linear regr.	–	–	✓	–	–	✓
[23]	✓	–	✓	Linear regr.	–	–	✓	–	–	✓
[127]	✓	–	✓	Linear regr.	–	–	✓	–	–	✓

3.1.4 Monitoring implementation. Power statistics related to the switching activity can be monitored either at the software or hardware level, as shown in Figure 2. A *software power monitoring* (**Softw.** in Table 1) implementation provides a flexible solution that can be applied in an after-market perspective, since making use of the performance monitoring counters (PMCs) already available in an existing design requires no hardware changes while the computation of the power estimate is carried out at the software level. Software-implemented run-time power monitors, depicted in Figure 2a, leverage the architectural statistics exposed through the PMCs, where a carefully selected subset of such PMCs is periodically read out to compute the power estimates. In contrast, a *hardware power monitoring* (**Hardw.**) solution requires to instantiate ad-hoc additional digital hardware elements that operate in a totally isolated manner from the main computing pipeline. As shown in Figure 2b, hardware-implemented run-time power monitors leverage the microarchitectural statistics directly exposed by the hardware platform. In particular, a hardware-implemented power monitor is an additional hardware component that periodically reads out the switching activity from a selected subset of physical wires to compute the power estimates. The output of the hardware-implemented power monitor can be exposed at the software level through a dedicated performance counter.

3.2 Classification of the state of the art

Table 1 classifies the surveyed state of the art according to the four taxonomy dimensions discussed in Section 3.1. A large fraction of the proposed solutions target general-purpose CPUs, while few are aimed at GPUs and hardware accelerators (see **Target** in Table 1). The simplicity of linear regressors makes them the de-facto standard power modeling design choice regardless of the monitored target component, although some proposals also explored learning-based solutions such as neural-network models (see **Family** in Table 1). From the model statistics viewpoint, performance events represent the most exploited source of information to compute the run-time power estimates for both CPUs and GPUs, while circuit-level switching activity is the most used information to feed run-time power monitors targeting hardware accelerators. Notably, there are no proposals in the literature that combine both performance events and toggling activity as the statistics of a run-time power model (see **Statistics** in Table 1). Finally, the majority of run-time power monitors targeting CPUs and GPUs are software-implemented due to the impossibility of modifying the hardware of commercial computing units, while, on the contrary, run-time power monitors targeting hardware accelerators are mainly implemented at the hardware level (see **Implem.** in Table 1).

An extensive discussion of the quality, performance, overheads, and implementation details of each state-of-the-art contribution related to the run-time power modeling and monitoring aspects is provided in Section 4 and Section 5, respectively.

4 POWER MODELING

The run-time power model is at the core of the run-time power monitoring infrastructure since it defines the mathematical function implemented in the power monitor. In particular, the run-time power model represents the identified mathematical function that delivers a periodic estimate of the power consumption of the target computing platform starting from the values of a carefully selected set of statistics that are periodically sampled. Such statistics might be any measurable activity in the target platform satisfying two properties. First, each statistic displays a high correlation with the power consumption of the underlying hardware architecture. Second, the set of selected statistics collectively explains the power consumption.

This section is organized into two parts, with the goal of presenting a taxonomy of the state of the art concerning the run-time power models for edge computing platforms, organized according to three macro-dimensions and evaluated according to three quality metrics. Section 4.1 introduces the considered quality metrics, defined as the properties that allow comparing the quality of different power models. Section 4.2 discusses the proposals available in the literature, starting from the quality metrics and the dimensions of the taxonomy defined in Section 4.1 and Section 3.1, respectively.

4.1 Quality metrics

For each state-of-the-art contribution, this survey considers three quality metrics, i.e., model complexity, accuracy, and training and assessment strategies. Such metrics guide the analysis and evaluation of the literature and allow a direct comparison between the different considered state-of-the-art solutions.

The designers of a run-time power monitor must carefully select the properties of the underlying power model in order to satisfy the requirements and constraints of the overall system and obtain the desired qualities related to power monitoring.

4.1.1 Model complexity. The complexity of a selected power model determines the complexity of the corresponding monitoring infrastructure and, therefore, the resulting implementation overheads. In this survey, we employ the number of variables employed as inputs of a power model as a proxy for its complexity (**Complexity** in Table 2).

4.1.2 Model accuracy. Accuracy is the degree of closeness of direct or indirect measurements of a quantity to its actual value. It describes systematic errors and measures statistical bias. In this survey, we measure the accuracy of a power model (**Accuracy** in Table 2) as its average estimation error, i.e., the mean relative error for the estimation of the dynamic power consumption.

4.1.3 Training and assessment strategies. Identifying (**Benchmarks** – **Training** in Table 2) and assessing (**Benchmarks** – **Assessment**) the run-time power model require executing benchmarks on the computing platform under analysis. These benchmarks may be either *general-purpose* ones or *micro-benchmarks*. *General-purpose benchmarks* (**Gen.**) are meant to mimic realistic applications, thus exhibiting complex software architectures and long execution time. They are commonly used to compare the performance, power consumption, and efficiency of different computing platforms. *Micro-benchmarks* (**Micr.**) are small kernels meant to timely and efficiently stress specific components of the computing platform. They are commonly used to test and debug specific portions of a computing platform.

On the contrary, using random stimuli to generate the switching activity patterns and the corresponding power traces is not instead a viable solution for the training and assessment of a power model for two main reasons. On the one hand, generating unconstrained random stimuli can provoke the execution of illegal machine instructions that will never occur during normal operating conditions. On the other hand, covering the entire space of the inputs and internal states using a random stimuli generation requires a large amount of time.

4.2 Detailed overview of the state of the art

Table 2 summarizes all the considered run-time power modeling proposals from the state of the art, classifying them according to the target platform, model family, and input statistics dimensions. For each proposal, complexity and accuracy numbers as well as training and assessment strategies are also reported. The state-of-the-art works are ordered primarily according to their target platforms, i.e., CPUs, GPUs, and hardware accelerators, and secondarily according to the employed power statistics, i.e., performance events and toggling activity. In the rest of this subsection, we discuss in more detail such state-of-the-art solutions, in the same order as in Table 2, analyzing their quality metrics and discussing their strengths and weaknesses.

Table 2. Detailed taxonomy of the run-time power modeling state-of-the-art. Legend: ✓yes, – no, N/A unavailable data; **Complexity** model complexity (see Section 4.1.1); **Accuracy** model accuracy (see Section 4.1.2), average estimation error; **Benchmarks** – **Training/Assessment** training and assessment strategies (see Section 4.1.3), general-purpose benchmarks (**Gen.**), micro-benchmarks (**Micr.**).

			Benchmarks				
			Training		**Assessment**		
Ref.	**Complexity**	**Accuracy**	**Gen.**	**Micr.**	**Gen.**	**Micr.**	**Section**
[9]	4	N/A	–	✓	–	✓	Sec. 4.2.1
[48]	9	≤ 8% Compaq ≤ 15% Intel	✓	–	✓	–	
[43]	15	avg. 3W, ≤ 5.8%	–	✓	✓	–	
[94]	3-5	3.0-4.0% Nehalem 3.9-6.1% Atom	✓	–	✓	–	
[90]	6	2.8% Cortex-A15 3.9% A7 → A15	✓	–	✓	–	
[111]	6 + 3 (V, F, C)	3.8% Cortex-A7 2.8% Cortex-A15	✓	–	✓	–	
[77]	10 + 4 (V, F, T, C)	5% Cortex-A7 8% Cortex-A15	✓	–	✓	–	
[76]	≤ 4 Cortex-A7 ≤ 6 Cortex-A15	≤ 3% single-thread ≤ 7.5% multi-thread	✓	–	✓	–	
[75]	1-12	1.46% (1 perf. ev.) 1.72% (12 perf. ev.)	✓	–	✓	–	
[74]	6	≤ 5%	✓	✓	✓	✓	
[86]	2-8	2%	✓	–	✓	–	Sec. 4.2.2
[71]	1-8 (Linear regr.) 1-6 (MARS) 1-6 (Neural netw.)	3.6% (Linear regr.) 2.9% (MARS) 4.0% (Neural netw.)	✓ ✓ ✓	– – –	✓ ✓ ✓	– – –	
[126]	6	≤ 9%	✓	–	✓	–	
[125]	N/A	≤ 1.8%	–	✓	✓	–	
[120]	100-400	6%	–	✓	–	✓	
[119]	100-300	3%	N/A	N/A	N/A	N/A	
[70]	13	4.7%	✓	–	✓	–	Sec. 4.2.3
[20]	22	7.8%	✓	–	✓	–	
[102]	12	2.1%	✓	–	✓	–	
[66]	12 + 14 (CPU + GPU)	4.5%	✓	–	✓	–	
[78]	4 + 3 (V, F, T)	5%	✓	–	✓	–	
[47]	1655-2050	≤ 5%	–	–	–	–	Sec. 4.2.4
[60]	10-20	4.4%	–	–	–	–	
[83]	1-25	≤ 3%	–	–	–	–	Sec. 4.2.5
[52]	113	≤ 7%	✓	✓	✓	✓	
[23]	1-29	≤ 5%	–	–	–	–	
[127]	N/A	≤ 2.7%	–	–	–	–	

4.2.1 CPUs - Performance events-based power modeling. The first class of state-of-the-art power modeling solutions targets general-purpose CPUs that expose performance monitoring counters (PMCs). While the initial works and most of the subsequent ones focused on linear run-time power models, the state-of-the-art also explored performance-events-based models with higher complexity.

[9] introduced the concept of estimating the energy consumption of a CPU by exploiting information gathered by PMCs about the activity of the CPU hardware units, e.g., ALU, FPU, cache, and memory. In particular, performance events such as the number of integer and control-flow operations, floating-point operations, and level 2 cache misses were shown to have a linear correlation with the energy consumption of the target platform. The experimental evaluation was carried out on the *x86 Intel Pentium II* general-purpose CPU [97] and *AMD ELAN* microcontroller, and it made use of synthetic workloads to train and assess the power model.

[48] proposed a method to estimate the power consumption of separate components of a CPU from its performance events. Such components include the instruction and data caches, the register file, the integer and floating-point ALUs, the buses, and the branch predictor. It also discussed which PMCs count events are actually relevant for the power consumption and how to estimate event counts for power-relevant events not well supported by the available PMCs. Indeed, since typical performance counters do not capture all the power relevant events, the proposed methodology approximated utilization factors through heuristics depending on the machine structure and on the available PMCs. The experimental evaluation considered the *Intel Pentium Pro* [10, 85] and *Compaq Alpha 21164* [7] CPUs and the training and assessment of the linear power models were carried out by employing general-purpose benchmarks from *SPEC CPU95* [103]. Experimental results highlighted average estimation errors up to 8% and 15% for the *Alpha* and *Pentium* processors, respectively.

In [43], the authors estimated the power consumption for each of 22 components of a CPU through linear regression models taking performance events as inputs. The total power consumption of the overall processor could then be computed as the sum of the power consumptions of the 22 components plus the idle power consumption of the CPU. This work targeted the *x86 Intel Pentium 4* CPUs, employing micro-benchmarks during the identification phase and then validating the obtained power model on the execution of benchmarks from the *SPEC CPU2000* suite [39] and of three Linux desktop applications. Concerning the *SPEC CPU2000* benchmarks, the identified power model showed a average estimation error of 3W and a maximum error of 5.8W.

[94] demonstrated the possibility to effectively estimate the dynamic power consumption of both *Intel* high-performance *Nehalem* [101] and low-power *Atom* [37] CPUs by using only three performance events, i.e., the number of fetched instructions, level 1 cache hits, and dispatch stalls. The performance events were fed to a linear power model, which was experimentally validated against the simulated *Wattch* [14] power model. Both training and assessment made use of general-purpose benchmarks from the *SPEC CPU2000* [39], *MiBench* [35], and *MediaBench* [56] suites. Results showed 4% and 6.1% average estimation errors on the *Nehalem* and *Atom* CPUs, which decreased to 3% and 3.9%, respectively, by employing 5 counters. Moreover, the authors showed that, for small differences in architecture type, the expression obtained for one architecture can be used to estimate power on the other, with a small increase, around 3%, of the estimation error.

[90] proposed a run-time power model that, targeting *ARM big.LITTLE* heterogeneous multi-core CPUs, predicted the power consumption of an application on a target core given its execution profile on the current core. For the smaller *Cortex-A7* cores of the target *big.LITTLE* multi-core, the authors claimed it was not necessary to devise a power model, due to the power consumption ranging between 1.4W and 1.5W across all training benchmarks, therefore showing a negligible variability. On the contrary, the larger *Cortex-A15* showed a power consumption varying between

4.5W and 5.1W, suggesting instead the need to identify a run-time power model. The authors identified therefore a linear regression model that takes as the inputs the amounts of integer and floating-point instructions, the number of instructions per cycle, and the access rates to the L1 data, L2, and main memory levels of the memory hierarchy. Moreover, by employing an inter-core prediction model for miss events, the statistics fetched during the execution on a *Cortex-A7* core can be exploited to estimate the power that would be instead consumed by executing on a *Cortex-A15* core. The authors made use of the *SD-VBS* [110], *SPEC CPU2000* [39], and *SPEC CPU2006* [40] benchmark suites, splitting them into two sets devoted to training and assessment, respectively. On the larger core, the model identified with an average estimation error of 1.2% on the training set showed a 2.6% average estimation error on the test set. The inter-core power model, predicting *Cortex-A15* power from *Cortex-A7* statistics, provided instead a 3.9% average estimation error on the test benchmarks.

[111] proposed a methodology to model power consumption with the PMCs available on *ARM big.LITTLE* multi-core processors. In particular, the authors targeted a SoC that featured *ARM Cortex-A7* and *-A15* CPU cores. This work highlighted the importance of carefully selecting a small subset of PMCs to be used within the power model, due to the need to reduce their number so that they can be monitored simultaneously as well as to avoid redundancy due to correlated events. A good PMC selection corresponds to events that are highly correlated with power but at the same time are not highly correlated with each other. The experimental evaluation employed benchmarks from the *MiBench* [35], *MediaBench* [56], and *Longbottom* [62] suites and the results produced average power estimation errors of 3.8% and 2.8% for *Cortex-A7* and *-A15* CPUs, respectively.

[77] proposed an approach for developing run-time power models that exploited a combination of physical predictors measuring frequency, voltage, and temperature, performance events, and information about the CPU state. The methodology is validated on an *ARM big.LITTLE* heterogeneous architecture. The authors also claimed that tuning the model for every CPU frequency level is much more accurate than having a unified model for all clock frequencies. The accuracy of the developed models was compared with the state-of-the-art, showing average prediction errors of 8% and 5% on *Cortex-A15* and *Cortex-A7*, respectively.

[76] extended the methodology described in [77] with more comprehensive event space exploration and statistical techniques to develop more accurate power models for *ARM big.LITTLE* SoCs compared to previous works [90, 111]. CPU state information was notably not employed as a power statistic, differently from [77], due to the large overhead introduced by its usage. The implemented models, which can be split into single- and multi-thread ones and into intra- and inter-core ones, were evaluated by targeting the *ODROID-XU3* development board, executing *cBench* benchmarks for single-thread models and benchmarks from the *PARSEC* [12] suite for multi-thread ones. Intra-core single-thread and multi-thread models reported average estimation errors below 3% and 7.5%, respectively. Notably, the multi-thread models took as their inputs sets of performance events that were completely different from those employed in the single-thread ones.

[75] described an approach to obtain accurate power models on embedded computing platforms which do not expose PMCs. Targeting a *Gaisler LEON3* CPU [3], the authors demonstrated the usage of a soft-core replica implemented on FPGA and enhanced with PMCs to obtain execution statistics and then correlate them with the power consumption measured on the original computing platform. Training made use of the *BEEBS* [84] open-source benchmark suite, specifically designed to assess the performance and energy consumption of embedded-system targets, while the assessment was carried out on a closed-source computer vision algorithm used in space satellite imaging. Experimental results highlighted a 1.46% accuracy error for a model employing the number of store instructions as the lone performance event and a 1.72% one for a model taking 12 events as

its inputs. Moreover, the ability of both models to follow program phases made them suitable for run-time power profiling at the design stage.

[74] modified the *Thumbulator* instruction set simulator to expose performance events in order to model the power consumption of microcontrollers that do not provide access to PMCs. The methodology targeted the *ARM Cortex-M0* microcontroller, and the training and assessment made use of the BEEBS [84] benchmark suite and of 154 micro-benchmarks, extracted from a edge computing application, implementing the various CNN layers with different hyperparameters and optimisations. In particular, the authors identified six performance events, namely executed instructions excluding multiplications, multiplication instructions, taken branches, RAM reads, RAM writes, and flash reads, as run-time power statistics. The resulting models, tailored to different clock frequencies and with knowledge of the activation of prefetch buffers and CPU wait states, provided an average prediction error below 5%.

4.2.2 CPUs - Toggling activity-based power modeling. [86] was one of the first works to exploit the toggling activity of control signals to model the dynamic power consumption. The authors presented a methodology to add hardware counters attached to the control path of the CPU, enabling run-time power monitoring without employing performance events exposed by PMCs, which might also be unavailable on some processors. Indeed, the toggling activity of a subset of identified control signals from the target CPU was shown to have a linear relation with the power consumption. The sum of the measured toggling activity counters, multiplied by coefficients determined within the model identification phase, was therefore shown to be a good estimate of the CPU power consumption. The model training and assessment employed applications from the *MiBench* [35] benchmark suite, and the estimates were compared to the measures simulated by *Synopsys Primepower* [107]. The proposed methodology, applied to an existing processor [87], resulted in average power and energy estimation errors of 2% and 1.5%, respectively.

[71] also proposed a methodology to add run-time monitoring of dynamic power consumption in SoCs starting from their RTL description. After generating the dataset of power consumption, signals, and toggling activity from simulation and selecting the optimal set of signals to monitor, and thus employ their toggling activity as the independent variables in the power model, the authors explored the usage of three power models and compared their accuracy as well as the overheads of the corresponding monitoring infrastructure. In particular, the authors evaluated linear regression, multivariate adaptive regression splines (MARS) [27], and neural network models. In their experimental evaluation, they targeted a *SecretBlaze* processor [8] implemented on a *Xilinx Spartan-6* FPGA [2]. The results highlighted the linear model as the best-performing one, when the input is the toggling activity of three signals, with a 3.6% average estimation error. On the contrary, the MARS model with three toggling activity counters showed a 2.9% average estimation error, however with a higher complexity which resulted in higher overheads when employing the power model at run time. Finally, the neural network model achieved a 4% average error, not improving accuracy over both the other models while also requiring a more complex software computation than the linear model.

[126] similarly proposed a dynamic power model, based on toggling activity, designed to be instrumented into an existing RTL design in order to add run-time power monitoring capabilities. To implement such monitoring infrastructure, the described methodology first collected from a simulation the architectural statistics and the related time-based power traces. This information has been used to develop a power model by employing linear regression. Obtaining such linear power model meant identifying the signals which contribute the most to dynamic power consumption and their corresponding weights. The design of the power model considered four figures of merit, i.e., not only the accuracy of the power consumption estimation but also the performance, power,

and area overheads of the implemented monitoring infrastructure. The proposed methodology, while theoretically applicable to any RTL design, was only evaluated by applying it to the *mor1kx* CPU [81]. Applications from the *WCET* [34] benchmark suite were employed for the training and assessment of the power model, which produced an average estimation error of 9%, with a standard deviation of 2%.

[125] extended the work in [126] to target complex hardware multi-threaded architectures that support single instruction, multiple data (SIMD) parallel processing, such as multi/manycore accelerators and GPUs. The proposed methodology identified power models by solely considering signals that represent either a primary input or output of an RTL module within the design hierarchy. According to the authors, such choice avoided modeling complex non-linear relationship between the power consumption and the internal logic of a hardware module and significantly reduces the computational time required by the model identification. The methodology distinguished between two ways to measure the switching activity of multi-bit signals, with the authors remarking that Hamming-weight counters, which count the number of varying bits for each variation in the signal, are well-suited to measure the switching activity of data signals, while single-toggle counters, that count any change in the target signal, are better tailored to control signals. The methodology was validated on *nu+* [28], a hardware multi-threaded SIMD processor. Notably, and differently from [126], power model identification exploited a set of micro-benchmarks tailored to the target architecture to stress specific parts of the target platform. The assessment phase employed instead the same *WCET* benchmark suite [34], with the experimental results showing a 1.8% maximum estimation error.

The *APOLLO* framework proposed in [120] introduced a novel methodology to identify the dynamic power consumption model, which is obtained by applying minimax concave penalty (MCP) regression [124]. The authors targeted an *Arm Cortex-A77* core for their experimental evaluation, which highlighted a per-cycle accurate estimation. Both training and assessment are carried out by executing micro-benchmarks. The training set is composed of 300 micro-benchmarks generated automatically through a genetic algorithm (GA)-based framework [36], while the validation set is instead made of 12 micro-benchmarks written by CPU designers to represent different use cases, such as low or high power consumption and CPU throttling. The model obtained in the experimental evaluation employed a number of input variables ranging from 100 and 400, producing an average accuracy error of 6% when executing the micro-benchmarks from the validation set.

[119] proposed the *DEEP* methodology to extend the work in [120]. Rather than operating on multi-bit signals as a single variable, *DEEP* proposes to select model variables at bit level rather than signal level, i.e., signals selected by the linear-regression power model can be any individual bits rather than whole signals. Such solution provides more flexibility in the design space that can be explored, since the number of bits is possibly much larger than the number of signals, which can in general be multi-bit ones. *DEEP* proposes a two-step selection method to identify the variables of the power model. A top-down pruning step based on MCP reduces the number of variables and it is followed by a bottom-up selection algorithm based on best subset selection[67] that produces the set of bits chosen as the model inputs. The authors targeted a 64-bit, single-core CPU for the experimental evaluation of the proposed methodology. Models taking as inputs a number of variables ranging from 100 to 300 produced an accuracy around 3%.

4.2.3 GPUs - Performance events-based power modeling. Power modeling approaches followed by state-of-the-art literature also use PMCs in GPUs similarly to those adopted for general-purpose CPUs.

[70] proposed to estimate the power consumption deriving from the execution of a GPU kernel through a linear regression model, where the independent variables are the performance events

collected by the PMCs. Notably, the proposed methodology estimated power consumption at the granularity of kernel calls since the PMCs of the GPU can be accessed only after the completion of each computed kernel. The methodology was demonstrated on *Nvidia CUDA* GPUs [79], training and evaluating the model by using 49 publicly available GPGPU kernels from the *CUDA SDK* and the *Rodinia* benchmark suite [19]. The experimental results highlighted an average estimation error of 4.7%.

[20] applied instead a tree-based random forest method to the run-time power modeling of *Nvidia CUDA* GPUs [79]. The experimental evaluation targeted the *Nvidia GTX280* graphics card, with 52 GPU application kernels from *CUDA SDK*, *Rodinia* [19], and *Parboil* [105] used for training and assessment of the power model. The experimental results showed that the random forest method produced an average estimation error of 7.77%, improving over state-of-the-art regression tree and multiple linear regression methods, both producing 11.7% errors.

The authors of [102] highlighted the shortcomings of employing linear regression models to estimate the power consumption of GPUs, proposing instead the adoption of neural networks (NNs). Among the advantages of modeling GPU power consumption with NNs, they cited their capability to capture non-linear dependencies, their flexibility and adaptability, and their attitude to learn various computation and memory access patterns which makes them suitable to model such a complex architectures. The experimental analysis targeted the *NVIDIA Tesla C2075* and *M2090* GPUs [115] and employed GPU kernels from *CUDA SDK*, *GEM* [32], and the *SHOC* [24] benchmark suite, with the results showing an average power estimation error of 2.1%.

[66] applied deep learning techniques to obtain a power model for a heterogeneous mobile SoC executing parallel applications. The resulting power model was a neural network trained using CPU and GPU PMCs along with actual power measurements by employing a set of OpenGL and OpenCL benchmarks representative of graphics and computing workloads, also used for its evaluation. In particular, the authors targeted a *Intel Z3560* SoC with a *Imagination PowerVR G6430* GPU. The proposed model is a fully connected neural network with four layers, that takes as inputs 14 PMCs from the GPU and 12 from the CPU and provides a power consumption estimate as the output. The model was implemented using the *Keras* framework [22] with *Tensorflow* [1] as its backend. Notably, the GPU utilization PMC is considered as a measure of data coverage of the training dataset. Benchmarks were executed several times with different input sizes under different load conditions until reaching a good coverage of GPU utilization values. The experimental results showed an average estimation error of 4.47% compared to real power measurements, with the authors claiming 3.3× and 2× smaller average estimation errors than state-of-the-art linear regression and NN models.

[78] proposed a power model for the *Nvidia Tegra X1* SoC, that features four *ARM Cortex-A57* high-performance cores coupled with a *Nvidia Maxwell*-based 256-core GPU. The authors distinguished between local events, affecting power consumption in limited regions of the GPU, e.g., the amounts of executed instructions and memory accesses, and global states, affecting global power consumption, e.g., operating frequency, voltage, and temperature. This work aimed to identify a single model that worked for any combination of voltage, frequency, and temperature conditions, making use of only four GPU performance counters. Training was carried out by executing applications from *Rodinia* [19], while the quality of the identified power model was assessed on benchmarks from *CUDA SDK*. Experimental results showed an average accuracy error of 5% for the proposed model, making the quality of its power estimation comparable to the one given by employing different models tailored each to a specific operating clock frequency.

4.2.4 *Accelerators - Toggling activity-based power modeling.*

State-of-the-art run-time power models for hardware accelerators usually exploit the toggling activity of a subset of their signals. Indeed, hardware accelerators do not usually expose any kind of information about performance events, whether they are designed by hand or generated through high-level synthesis tools.

[47] exploited singular value decomposition (SVD) [31, 53], which is commonly used in machine learning to reduce the number of data dimensions for principal component analysis [112], to identify the correlation between the toggling activity of registers and the power consumption. In particular, SVD highlights which are the most critical registers and how much they contribute to the total power consumption, which is then estimated by a linear model. The power model can then be automatically instrumented to the RTL implementation and synthesized into an FPGA platform, where it is employed for run-time power monitoring. The proposed methodology, which targets generic RTL designs, was validated by considering three FPGA implementations of crypto cores and audio/video encoders/decoders. The obtained power model predicted run-time power consumption with an average estimation error below 5%, compared to a commercial power estimation tool.

[60] proposed a dynamic power consumption model based on decision-tree regression, rather than the traditional linear regression. The methodology targeted hardware accelerators implemented on FPGA, with the goal of instrumenting the identified power model within the RTL design. The proposed methodology is composed of three main steps. First, the most significant signals, i.e., nets in the Verilog HDL netlist exported after place-and-route, are identified through a timing simulation with random-generated input vectors, also filtering the redundant signals. Then, power traces are derived via post-implementation timing simulation. Finally, a decision tree model is identified, tuning its hyper-parameters depending on the desired complexity. The authors claimed that the decision tree is well suited to an area-efficient hardware implementation, since the corresponding series of comparators can be decomposed into a series of if-then-else rules that map effectively to hardware. The experimental results showed an average power estimation error of 4.36%.

4.2.5 CPUs and accelerators - Toggling activity-based power modeling. As the last class of the proposed power modeling taxonomy, we discuss solutions targeting generic RTL designs, i.e., general-purpose CPUs and dedicated hardware accelerators, by applying the same methodology, which exploits toggling activity as the input statistics to the identified run-time power model.

[83] proposed regression models based on toggling activity for both static and dynamic power while instrumenting the RTL level of generic hardware designs, i.e., covering both general-purpose CPUs and hardware accelerators. Concerning dynamic power, the switching activity of an identified subset of registers is exploited to compute the power consumption estimate according to a linear model, whose coefficients are also identified within an automated analysis flow. In addition, the model is retrained after the place-and-route phase that added the corresponding power monitoring infrastructure, allowing to account for variations in power consumption due to its insertion in the RTL design. The methodology was applied to the 32-bit RISC-V [114] *RI5CY* processor [109] and to two hardware accelerators from *OpenCores* that implemented the AES cryptographic algorithm and a FIR filter. The implemented models showed an average estimation error smaller than 3%.

[52] presented a run-time power modeling methodology applicable to any RTL design that automatically identifies the signals that contribute the most to dynamic power consumption through clustering. It leverages the intuition that signals displaying similar toggle patterns will also have a similar contribution to power consumption, hence the modeling error can be reduced by having all the signals from a cluster share the same coefficient in the power model. After clustering signals that showed similar switching activities in the VCD dumps obtained for the execution of benchmark applications from the training set, and thus identifying a smaller number of signals, the methodology produces a linear-regression model with high-order, i.e., >1 order, terms. The RTL simulation required to collect run-time power traces within the model training phase is further sped up through FPGA acceleration by exploiting the open-source *Strober* framework [51]. Both

micro-benchmarks and general-purpose benchmarks are used for training and assessment purposes. The authors demonstrated their methodology by targeting a heterogeneous processor composed of a Rocket in-order core [6] and a Hwacha vector accelerator [57]. Results are shown for a number of model signals ranging from 40 to 120 and a window size ranging from 80 to 380 clock cycles, and a model with 113 signals and a 340-cycle window size is shown to have the best accuracy.

[23] proposed a linear model based on toggling activity and targeting generic RTL designs, but with a focus on satisfying the resource constraints when implementing the corresponding monitoring infrastructure. The solution considers the impact of the area overhead due to the power monitoring and it allows to implement a model with the desired accuracy, capable to fit a certain budget allocated for its resources, i.e., considering constraints to the amount of FPGA resources that can be devoted to the actual RTL implementation of the power monitor. The methodology was evaluated on HLS-generated [89] hardware accelerators corresponding to kernels from the *WCET* benchmark suite [34] and on a 32-bit RISC-V single-core CPU [96]. Resource-constrained instances of the power model managed to maintain the average estimation error below 5%.

[127] introduced a methodology to design SCA-resistant power monitors, ensuring that the switching activity of the signals used to compute the power estimates is not a function of the cryptographic keys, plaintexts, and ciphertexts processed within the target computing platform. Such security aspect must be tackled at the model level, i.e., during the identification of the run-time power model. SCA-resistant power monitors targeting a general-purpose computing platform showed an average estimation error in the 1.2%-2.4% range, depending on the temporal resolution, while the ones targeting a HLS-generated hardware accelerator showed a 0.89%-2.7% error range.

5 POWER MONITORING

Run-time power monitors implement the identified power models to deliver online estimates for the power consumption of the target computing platforms. Their implementation can be constrained with respect to *i)* the minimum values for the quality metrics, i.e., temporal resolution and accuracy, and *ii)* the imposed constraints, i.e., area, power, and performance overheads. Regardless of the software or hardware implementation of the power monitor, the activity counters represent the critical components of the entire run-time power monitoring infrastructure. The activity counters are in charge of sampling the activities to feed the power model. In software-implemented power monitors, the performance counter infrastructure defines and implements the activity counters to sample the required architectural statistics, such as the number of executed ALU instructions or cache accesses. In contrast, hardware-implemented power monitors define their own activity counters.

The rest of this section is organized into two parts, with the goal of presenting a taxonomy of the state of the art of run-time power monitors for edge computing platforms that considers two classification dimensions and four quality metrics. Section 5.1 presents first the considered quality metrics, which are the properties that allow comparing the effectiveness of different power monitors. Section 5.2 discusses then the contributions from the literature, starting from the quality metrics and the dimensions of the proposed taxonomy that were defined in Section 5.1 and Section 3.1, respectively.

5.1 Quality metrics

This survey analyses the considered state-of-the-art run-time power monitoring solutions according to four quality metrics. They are the temporal resolution and the performance, area, and power overheads. Such metrics guide the analysis of the state of the art and allow evaluating and comparing the solutions proposed by the literature.

Table 3. Detailed taxonomy of the run-time power monitoring state-of-the-art. Legend: – no, N/A unavailable data; **Temp res.** temporal resolution (see Section 5.1.1), sampling period in seconds or clock cycles, (optional) operating frequency of target; **Overheads** – **Perf./Area/Power** performance, area, and power overheads (see Section 5.1.2); [71]: ^{l} linear, ^{n} NN, and ^{m} MARS models, ^{e} energy overhead; [83]: ^{r} RI5CY, ^{a} AES, ^{f} FIR, ^{s} SW-based, ^{h} HW-based power estimation; [127]: ^{u} unprotected, ^{p} protected.

Ref.	**Temp. res.**	**Overheads**			**Section**
		Perf.	**Area**	**Power**	
[9]	N/A	N/A	–	–	Sec. 5.2.1
[48]	1ms	N/A	–	–	
[43]	440ms	N/A	–	–	
[94]	100K cc	N/A	–	–	
[90]	500ms @1GHz	$< 0.5\%$	–	–	
[111]	1ms-1s @2GHz	N/A	–	$\leq$ 60mW	
[86]	10K cc	0.2%	4.9%	3%	Sec. 5.2.2
[71]	0.1-1ms @25MHz	$\geq 0.3\%^{l}$ $\geq 2\%^{n}$ $\geq 3\%^{m}$	$1.5\text{-}4\%^{l}$ $1.5\text{-}3\%^{n}$ $1.5\text{-}3\%^{m}$	$0.8\text{-}2\%^{l,e}$ $\geq 2\%^{n,e}$ $\geq 3\%^{m,e}$	
[126]	2us @50MHz	–	6.9%	4.7%	
[125]	2us @50MHz	–	9.9% LUT 3.9% FF	12mW	
[120]	1cc	–	0.2%	0.9%	
[119]	1cc	–	0.04-0.08%	N/A	
[70]	Kernel	N/A	–	–	Sec. 5.2.3
[20]	N/A	N/A	N/A	N/A	
[102]	60Hz	N/A	–	$\leq$ 5W	
[66]	Kernel	N/A	–	–	
[78]	500ms	N/A	–	–	
[47]	N/A	–	15.8-25.1% 3600-5700 LUT	N/A	Sec. 5.2.4
[60]	3us @100MHz	–	$\leq$ 162 LUT $\leq$ 508 FF $\leq$ 2.5 BRAM $\leq$ 9 DSP	$\leq$ 6mW	
[83]	0.1-1ms @800MHz	$\leq 1.4\%^{s}$ $0\%^{h}$	$1.4\%^{r,s}$ $7.1\%^{r,h}$ $0.5\%^{a,s}$ $1.8\%^{a,h}$ $9.7\%^{f,s}$ $31.7\%^{f,h}$	$1.8\%^{r,s}$ $5.8\%^{r,h}$ $0.5\%^{a,s}$ $1.6\%^{a,h}$ $12.4\%^{f,s}$ $22.1\%^{f,h}$	Sec. 5.2.5
[52]	80-380 cc	–	N/A	N/A	
[23]	0.02-0.5ms	–	Constrained	Prop. to area	
[127]	0.1-0.5ms	–	$\leq 7\%^{u}$ $\leq 6\%^{p}$	$\leq 5.5\%^{u}$ $\leq 5\%^{p}$	

The design of a power monitoring infrastructure might optimize different quality metrics depending on the requirements and constraints of the overall system. In general, the optimization of a quality metric might worsen another one. The designers must indeed carefully consider the trade-offs between the contrasting quality metrics, and, in particular, might aim to minimize the values of each of the four considered quality metrics.

5.1.1 Temporal resolution. Sampling period, corresponding to the difference in time between two consecutive estimates. In this survey, we measure the temporal resolution of a power monitor (**Temp. res.** in Table 3) as either the absolute time in seconds or the number of clock cycles occurring between two samples. For some contributions, the operating frequency of the target platform is also reported in Table 3 for reference.

5.1.2 Performance, area, and power overheads. The monitoring infrastructure may present overheads in terms of loss of performance (**Overheads** – **Perf.** in Table 3), due to the use of computing resources, and increased area (**Overheads** – **Area**) and power consumption (**Overheads** – **Power**), originated by the additional hardware logic instantiated to implement the power monitor.

5.2 Detailed overview of the state of the art

The run-time power monitoring contributions were previously categorized according to their target platform and hardware or software implementation in Table 1, and their power modeling classification was further discussed in Section 3. Table 3 summarizes all the considered run-time power monitoring state-of-the-art investigations, highlighting quantitative results in terms of temporal resolution as well as performance, area, and power overheads. The state-of-the-art works are ordered primarily according to their target platforms, i.e., CPUs, GPUs, and hardware accelerators, and secondarily according to whether the monitoring infrastructure is implemented at the software or hardware level. The solutions for power monitoring can be grouped into four main classes according to the target platform and the software or hardware implementation of the monitoring infrastructure. In particular, we can identify software-based monitoring solutions targeting general-purpose CPUs and GPUs and hardware-based ones targeting CPUs and hardware accelerators. Due to the overlapping of hardware-based solutions that target generic RTL designs, i.e., both CPUs and accelerators, such state-of-the-art solutions are treated as a fifth separate taxonomy class. In the rest of this subsection, we discuss state-of-the-art solutions in more detail, in the same order as in Table 3, analyzing their quality metrics, strengths, and weaknesses.

5.2.1 CPUs - Software-based power monitoring. The first state-of-the-art solutions for run-time power monitoring targeted general-purpose CPUs and made use of performance monitoring counters. Indeed, such counters were already available on target platforms, albeit being designed for other purposes, i.e., monitoring the performance of the system by counting significant events such as cache misses, memory accesses, and ALU operations.

[9] first introduced a run-time methodology that employed the PMCs available on the target CPU to track events related to power consumption. The number and type of these events is periodically stored and updated, allowing to estimate energy and power consumption both at thread and system level. The methodology was evaluated on a *Intel Pentium II* general-purpose CPU [97] and an *AMD ELAN* microcontroller. The authors remarked a performance overhead caused by saving and restoring counter values in the context switching routine.

[48] implemented a similar solution on the *Compaq Alpha 21264* [7] and *Intel Pentium Pro* [10, 85] CPUs. The authors pointed out the limitation due to the maximum number of performance events that can be measured simultaneously, requiring therefore to rotate through the available PMCs, examining concurrently a limited subset of counters that can be accessed at the same time.

[43] described a methodology that combined measurements of the total CPU power consumption with PMC-based per-component power estimates. The resulting tool offers total power measurements at run time for *Intel Pentium 4* [41, 104] single-core CPUs and also provides a breakdown of power consumption for the 22 main components of the CPU. The authors employed 15 PMCs with 4 rotations, minimizing the counter switches required to measure all the needed metrics. Notably, an actual run-time monitoring implementation would mandate for the collection of 4 PMCs at most, due to limitations of the *Pentium 4* CPU.

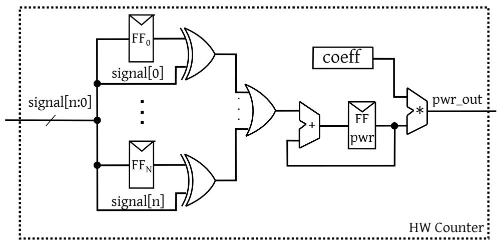
[94] assessed the possibility to identify a universal subset of PMCs that can be used to accurately estimate the power consumption of any computing architecture, reducing the number of considered PMCs in order to minimize the effects of the limitations of the actual number of concurrently monitored PMCs and the effective availability of PMCs. The experimental analysis was provided on the *Intel* low-power *Atom* [37] and high-performance *Nehalem* [101] processors, highlighting an estimation error below 5%. The authors also remarked the effects of the window size on the power estimation error. Indeed, while a small interval might cause a more significant accuracy loss, large intervals may lead to more accurate estimates at the cost however of slowing down the feedback control dynamics. An interval length of $100,000$ clock cycles was shown to be a reasonable solution for both considered CPUs.

In [90], the authors targeted an *ARM big.LITTLE* chip consisting of *ARM Cortex-A7* and *-A15* cores. The proposed power monitoring infrastructure collected the PMC values by exploiting the ARM Streamline *gator* kernel module and daemon [4]. The *gator* driver is a dynamic kernel module that interrupts the core at periodic intervals to collect the performance counters. The implemented power monitoring solution is characterized by a minimal performance overhead due to the *gator* daemon in the background. According to the authors, the average CPU utilization of the *gator* daemon was less than 0.5%.

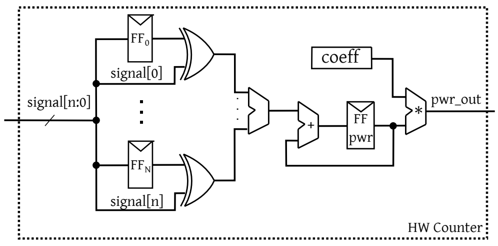
[111] targeted an *Exynos 5422* SoC, that implemented an *ARM big.LITTLE* architecture and featured four *ARM Cortex-A7* low-power cores and four *Cortex-A15* high-performance cores. The implemented power model required the collection of six performance counters and of the clock cycles counter. The authors described the employed data acquisition setup on real-world, commercial platform, namely the *ODROID-XU3* development board. However, such experimental setup is not aimed to provide run-time power monitoring. The sampling rate of PMC collection ranges between 1Hz and 1KHz, while the *Cortex-A15* in particular can reach a clock frequency up to 2GHz. The power overhead given by the data collection platform is up to 60mW at a 1KHz sampling rate on a *Cortex-A15* core.

5.2.2 CPUs - Hardware-based power monitoring. Hardware-based monitoring solutions emerged later than software ones due to the need to build a custom infrastructure that accounts for the toggling activity of select signals within the target platform. Such hardware-based monitoring techniques were first applied to general-purpose CPUs.

[86] presented a methodology to modify a processor so that it can estimate its own power consumption at run time, building a power monitoring infrastructure composed of hardware counters attached to the control signals of the target architecture. The area and power overheads given by applying the proposed methodology to an existing processor [87] showed values of 4.9% and 3%, compared to the original target processor, respectively, with a sampling period of $10,000$ clock cycles. Notably, the computation of the power and energy consumption estimates is performed at the software level by the target processor, producing a 0.2% performance overhead. The small impact of performing such computation within the target processor justifies the choice to not implement additional dedicated logic, which would instead add further cost in terms of area and power consumption, according to the authors.



(a) Single-toggle counter



(b) Hamming-weight counter

Fig. 3. Architecture of two activity counters measuring the switching activity of a multi-bit signal [126].

[71] proposed a methodology to design a power monitoring infrastructure considering all single-bit signals derived from the RTL description of the target platform, instead of focusing only on the control signals as in [86]. The monitoring infrastructure collects the toggling activity of an identified subset of single-bit signals by instantiating dedicated hardware counters. The authors show applied the methodology to a *SecretBlaze* processor [8] implemented on a *Xilinx Spartan-6* FPGA [2]. Notably, the computation of the power consumption estimate, starting from the toggling activity measured by the hardware counters, is carried out at the software level, i.e., it is demanded to the CPU. The sampling period varies between 100us and 1ms, while the CPU clock frequency is set to 25MHz. The size of counters, which depends on the sampling period and on the clock frequency, ranges between 12 and 15 bits. Each counter occupies eight *Spartan-6* slice registers and LUTs, resulting in an area overhead of 0.5% FFs and 0.3% LUTs compared to the target SoC. The overall area overhead ranges from 1.5%, for 3 counters, to 4%, for 8 counters. Performance overheads grow with the number of counters. The linear model has the lowest overhead, while the maximum overhead of MARS is critically large. The maximum overhead of the linear and NN models is lower than 10% for sampling periods greater than 200 and 300 us, respectively. This overhead reduces significantly by limiting the number of signals in the models and by increasing the sampling period, decreasing down to 0.3% for linear and 2% for NN models with three signals at 1 ms. The minimum MARS performance overhead is instead about 3% at 1 ms. On the energy side, the overhead when implementing a linear model is the lowest one, ranging between 4.2% and 11% at 100 us and between 0.8% and 2% at 1 ms. It is instead greater than 12% at 100 us and 2% at 1 ms for NN and 20% at 100 us and 3% at 1 ms for MARS.

[126] presented a run-time power monitoring methodology that automatically extracts and implements a power model from the RTL description of the target architecture. The use of ad-hoc hardware that continuously updates the power estimate minimizes both the performance and the power overheads. The proposed methodology was validated on *mor1kx* [81], a 32-bit, single-issue, in-order CPU that implements the *OpenRISC 1000* ISA [55]. The CPU was implemented on a *Xilinx Artix-7* FPGA [91] at 50MHz. Results showed an average prediction error within 9% and area and power overheads limited to 6.9% and 4.7%, respectively, by employing 6 toggle counters, each of which occupies 1.1% area and consumes 0.8% power with respect to the overall CPU. The measured statistics, i.e., the toggle counts for the primary inputs and outputs of each module in the hierarchy, were periodically sampled every 100 clock cycles, resulting in a temporal resolution of 2us. Despite the claimed generic applicability of the methodology to any RTL design, the experimental evaluation did not consider dedicated hardware accelerators.

[125] extended the work in [126] to target complex hardware multi-threaded processors. Toggling activity counters implement two different architectures, depending on whether they measure single-toggle or Hamming-weight counts. Figure 3 details the architecture of such two classes of counters monitoring a generic multi-bit wire. A single-toggle counter, whose architecture is depicted in Figure 3a, counts whether there is at least one change in the signal on the monitored physical wire. It takes the multi-bit input *signal* and combines a XOR-tree network with an OR gate to output 1 or 0 depending on whether at least one bit of the signal toggled, and the produced value is added to the cumulative switching activity stored in *FF pwr*. A Hamming-weight counter, whose architecture is depicted in Figure 3b, counts instead the number of bits in the signal that switched their values. It takes the multi-bit input *signal* and combines a XOR-tree network with an adder to output the number of switching bits, adding the latter value to the cumulative switching activity stored in *FF pwr*. Concerning area overhead, the number of flip-flops for Hamming-weight counters increases with the size of the monitored signal and the number of clock cycles in the time window, while in the case of single-toggle counters it depends only on the time window, since for each clock cycle they perform at most a unitary increment. Hamming-weight counters also show a higher power overhead, due to their higher switching activity. The methodology was validated on *nu+* [28], a hardware multi-threaded SIMD processor, which was implemented on a *Xilinx Artix-7* FPGA [91]. Temporal resolution was set to 2us at a 50 MHz clock frequency, thus corresponding to 100 clock cycles. The RTL implementation showed an area overhead of 9.95% LUTs and 3.87% FFs and a power overhead of 12.17 mW.

The *APOLLO* framework proposed in [120] also automatically instruments run-time power monitoring capabilities within an existing RTL design. The monitoring infrastructure operates by performing the accumulation of the weights assigned to the monitored signals which toggled their state during the current clock cycle. Such conditional accumulation is implemented as a set of adders and AND gates and no multipliers. The weights are fixed-point values with the same width, and they are accumulated, conditionally depending upon the toggling activity of the monitored signals, into a single counter that holds the estimation for the average power consumption over a sampling window. The experimental evaluation, targeting an *Arm Cortex-A77* core, explored a design space where the number of monitored signals ranged between 100 and 400 and the number of bits for the fixed-point representation of the model weights ranged from 5 to 13. The area overhead stayed below 0.5% across the whole design space, reaching values below 0.1% with the minimum number of monitored signals and smallest fixed-point representation. In particular, the best combination of accuracy and overheads was identified by the authors to only require an area overhead of 0.2% and a power overhead of 0.9%.

The *DEEP* framework [119] is an extension of the work in [120], and the architecture of the monitoring infrastructure is similar to the one automatically instrumented by the *APOLLO* framework. To reduce the resource utilization of the run-time monitor, weight quantization is applied at design time. All the weights are quantized to integers and can be encoded by a different number of bits. The values of the weights are fixed and determined at design time, simplifying the architecture of the power monitor. In addition to produce the estimation of the power consumption of the whole monitored design, the monitoring infrastructure can also be modified to produce and expose estimates at the component level. The sum of such component-level estimates will then correspond to the total power consumption. The experimental evaluation, targeting a 64-bit, single-core CPU, produced results that highlighted an area overhead ranging from 0.04% to 0.08%

5.2.3 GPUs - Software-based power monitoring. Software-based power monitoring was also applied to GPUs, that expose performance counters in a similar way to general-purpose CPUs. An example is the software interface provided by the *CUDA* framework, which exposes the statistics collected from the PMCs of *Nvidia* GPUs.

[70] employed the *CUDA* performance counters exposed by *Nvidia* GPUs to estimate the power consumption of GPU kernels. The proposed solution is limited by the need to monitor 13 counters, all of which are employed by the identified power estimation model, while the GPU only allows 4 counters to be monitored simultaneously. All counter values can be collected by running the same kernel multiple times with the same input data, that is a possibility only for offline power estimation, while in an online scenario it is mandatory to minimize the number of monitored counters. The authors did not provide any analysis of the performance overhead.

The authors of [20] collected run-time characteristics from the execution of a set of sample kernels on the *GPGPU-Sim* GPU simulator, a cycle-level simulator that simulates an instruction set which closely resembles the real ISA of Nvidia GPUs.

[102] targeted a *Nvidia CUDA* GPU, collecting the selected performance events to be fed to a run-time power model through CUPTI[80]. A dedicated thread was devoted to sampling a selected set of GPU performance counters at a sampling interval at a sampling frequency of 60Hz, i.e., 60 times per second. Even the small power overhead due to the run-time profiling thread was considered within the power model, and was less than 5W even on the higher-end *Nvidia Tesla C2075* and *M2090* GPUs.

The authors of [66] applied their proposed power modeling methodology targeting mobile GPUs to monitor the power consumption of the *Intel Z3560* heterogeneous mobile SoC, which features an *Imagination PowerVR* GPU. The GPU performance counters were collected through the *Imagination PVRScope* API, while the CPU counters were obtained using the Linux *perf* API [25].

[78] targeted a *Nvidia Tegra X1* SoC, implementing the collection of performance monitoring counters from the *Nvidia Maxwell* GPU to be fed to a run-time power model. The power consumption model required four PMCs, that is the maximum limit by the GPU architecture. Sampling was performed through a CPU thread that collected the counters once every 0.5 seconds, i.e., at a 2Hz sampling frequency.

5.2.4 Accelerators - Hardware-based power monitoring. Other works in the state of the art of power monitoring targeted instead dedicated hardware accelerators, either designed by hand or generated through high-level synthesis. Estimating the power consumption of accelerators requires by definition to recur to alternative proxies to PMCs, which are not available on such targets.

[47] presented a framework that allowed to automatically instrument the power model at the RTL level by collecting the toggling activity of an identified subset of registers, thus effectively enabling run-time power monitoring. The additional logic instrumented by the proposed framework must compute the sum of the products between the toggling activity and the model coefficients, with the result corresponding to the estimated dynamic power consumption. The model coefficients are quantized to fixed-point values to simplify their hardware implementation. The authors evaluate the area overhead on a H.264/AVC decoder implemented on a *Xilinx Virtex-6* FPGA [2]. The additional power monitoring logic requires 3603 LUTs, compared to the 22807 LUTs required by the H.264/AVC decoder, with an area overhead of 15.8%, if a 1-stage adder tree is employed. In such conditions, the decoder runs at a 41.6MHz clock frequency while the clock frequency for the power monitoring infrastructure is set at 73.3MHz, which is sufficient for run-time estimation, according to the authors. Moreover, the adoption of a 2-stage adder tree would require instead 5735 LUTs, with an area overhead of 25.1% and increasing the clock frequency to 114.2MHz.

The hardware power monitoring infrastructure proposed in [60] targeted HLS-generated accelerators instantiated on FPGAs [73] by implementing a decision-tree-based power model, with the goals of reducing the overheads of traditional linear-model implementations and providing a sampling interval as small as tens of clock cycles. The FPGA-based power monitoring architecture is composed of two main components. The toggling activity counters may exploit either LUT or DSP resources, according to which are most widely available, depending on the target chip and on the resources already occupied by the implemented RTL design. The decision tree regression engine is designed instead as a memory-based architecture, which exploits the BRAM resources available on the FPGA to store the memory structure of the decision tree. The experimental analysis employed C-based benchmarks from *CHStone* [38], *PolyBench/C* [121], and *Machsuite* [92], deriving their Verilog description through the *Vivado HLS* framework [73] and implementing them on a *Xilinx Virtex-7* FPGA [91]. The power monitoring infrastructure was shown to require up to 162 LUTs, 508 FFs, 2.5 36kb BRAMs, and 9 DSPs. The power overhead was up to 6 mW and the temporal resolution of the power estimation was 3 us.

5.2.5 CPUs and accelerators - Hardware-based power monitoring. More recent works explored the possibility to monitor the power consumption of generic RTL designs, encompassing general-purpose CPUs and dedicated hardware accelerators, by applying the same methodology, which exploits the usage of hardware counters to monitor the toggling activity of RTL signals.

[83] implemented power monitoring for run-time dynamic and static power estimation, accounting for the toggling rate of an identified subset of flip-flops. The methodology is proposed in two variants, a software-based one suited for SoCs and CPUs that produces smaller area and power overheads and a hardware-based one that can be tailored to fit any RTL design. The hardware-based meter requires additional area and produces additional power consumption due to the need to also perform in hardware the fixed-point multiply-and-accumulate operations, which are instead carried out by a CPU or a dedicated microcontroller in the software version, that are required to compute the power estimate. The counters for the toggling rate of the identified flip-flops, as well as the power computation logic in the hardware-based case, are automatically added to the original RTL design. The methodology was evaluated on a *RI5CY* RISC-V core [109], on a FIR filter, and on a AES cryptographic accelerator. On the RISC core, the power meters have an area overhead of 1.4% for the software-based version and 7.1% for the hardware-based one, with an average estimation error smaller that 3%. Concerning performance overhead, the authors note that the software-based computation of the power estimate takes around 1.4us, that corresponds to up to 1.4% for a sampling interval that ranging between 100us and 1ms.

The methodology proposed in [52] automatically instruments the power model into the target platform by exploiting a custom pass added to the FIRRTL compiler [45]. Each monitored signal is fed to a corresponding component that computes the Hamming distance between the values at the previous and current clock cycles. In addition, each of those signals is assigned to a counter within a register file instantiated in the top module of the target platform. The unit computing the Hamming distance of each monitored signal is connected to the corresponding counter in the register file, and such counter is updated at each clock cycle. The register file is exposed to the software running on the target platform. Within the experimental evaluation, targeting a heterogeneous processor composed of a Rocket in-order core [6] and a Hwacha vector accelerator [57], the power monitor is implemented to collect the toggling activity from 40-120 signals in sampling windows of 80-380 clock cycles. No information is provided by the authors for what concerns area and power overheads due to instantiating the run-time power monitoring infrastructure within the target platform.

[23] proposed a framework that leveraged the *Yosys* open-source synthesizer [116] to automatically implement a resource-constrained power monitor in any computing platform for which the hardware RTL description is available. Indeed, while the other state-of-the-art works mostly focused on maximizing the accuracy and temporal resolution or minimizing the implementation overheads, this work aimed to provide the best accuracy and resolution while satisfying a strict area constraint

pre-defined by the system designer. The methodology was applied both to HLS-generated hardware accelerators [89], which implemented eight kernels from the *WCET* benchmark suite [34], and to a 32-bit RISC-V single-core CPU [96]. At a 20us temporal resolution, the proposed methodology produced an area overhead reduction comprised between 37% and 81% compared to state-of-the-art solutions that did not consider area constraints, while maintaining the average accuracy loss below 5%. In a scenario that is more common with the rest of the state of the art, i.e., with a temporal resolution in the range of hundreds of microseconds, the average accuracy loss remained below 1% with a similar area overhead.

More recent works targeted the security aspects of power monitoring. Due to the threat posed by side-channel analysis (SCA) attacks carried out by analyzing the power estimates generated by hardware-based power monitors, [127] introduced a methodology that extends the work in [23] to design SCA-resistant power monitors. The proposed framework ensures that the switching activity of the signals used to compute the power estimates is not a function of the cryptographic keys, plaintexts, and ciphertexts processed within the target computing platform. SCA-resistant power monitors targeting a general-purpose computing platform showed an average estimation error in the 1.2%-2.4% range, depending on the temporal resolution, compared to 0.9%-2.3% for the corresponding unprotected version, while the ones targeting a HLS-generated hardware accelerator showed 0.9%-2.7% and 0.5%-1.5% error ranges, respectively. The area and power are up to 6% and 5%, respectively, for the protected monitors, while the corresponding unprotected ones show area and power overheads up to 7% and 5.5%.

6 SUMMARY AND FUTURE DIRECTIONS

This survey reviewed previous research on run-time power monitors for edge devices, with the final goal of steering the reader towards the identification of the power monitor that best fits the needs of their application scenario and the constraints of their target computing platform. Starting from the presentation of the run-time modeling and monitoring design problems, the survey defines a novel taxonomy for run-time power monitors where each contribution is classified according to a set of carefully selected quantitative and qualitative metrics.

The review of the state of the art highlighted five key findings:

- Linear regression represents the most used statistical approach to model the relationship between dynamic power consumption and the switching activity of the circuit [86]. Compared to more complex solutions that implement models based on learning techniques such as neural networks [66, 102], random forests [20], decision trees [60], and deep learning techniques, the reported results demonstrated that linear regression offers lower complexity and overheads while ensuring accuracy error for the corresponding power monitors within 5% across a wide range of computing platforms [71].
- Run-time power monitors of commercial CPUs and GPUs are commonly implemented at the software level and leverage coarse-grained performance events such as cache misses and ALU, floating-point, and load-store instructions [43, 48, 77, 90, 94, 111]. Such design choices are motivated by *i)* the impossibility of changing the hardware of such platforms and *ii)* the availability of a performance counter infrastructure to collect the required performance events to feed the power model [9].
- The run-time power monitors for application-specific accelerators are implemented in hardware leveraging the linear regression approach on the circuit-level switching activity of selected signals [47, 60, 83]. At the same time, as the momentum of the RISC-V ISA is fueling a trend in the design of custom embedded and edge CPUs [6, 16, 29, 30, 122], similar monitoring

techniques are commonly applied also to such platforms, driven by their simplicity compared to commercial processors and the availability of their RTL design sources [71, 119, 125].

- Hardware-based monitors can consider either selected single bits of multi-bit signals [120] or entire multi-bit signals [119], and activity counters for multi-bit signals can notably be divided into single-toggle and Hamming-weight counters, as previously shown in Figure 3 and discussed in Section 5.2.2. The design of the toggling activity counters is crucial to the area and power overheads and accuracy of a hardware-based monitor. The simpler architecture of a single-toggle one consumes less power due to a reduced switching activity, since it records at most one toggle regardless of the number of toggling bits [125]. In contrast, a Hamming-weight counter, which outputs a statistic that encodes more information than a single-toggle one, i.e., the number of bits that toggled within the monitored multi-bit signal, has a more complex architecture, including an adder instead of the XOR used in the single-toggle counter [119].
- State-of-the-art proposals have traditionally targeted primarily the optimization of the run-time power monitor's accuracy and temporal resolution metrics. However, recent contributions have addressed the area overhead. For example, [23] proposed an area-constrained methodology to automatically instrument run-time power monitors in generic RTL hardware components, while [119] demonstrated the possibility to aggressively reduce the area overhead by selecting model variables at bit level rather than signal level, i.e., signals selected by the power model can be any individual bits rather than whole signals.

Having carefully investigated the current state of the art related to run-time power monitors for edge devices, we identify the following key research directions.

Optimization of non-functional metrics. While research in the field started considering the area overhead as a meaningful metric to optimize within the systematic design of the run-time power monitor [23, 119], the optimization of the power overhead one is still largely neglected. Notably, the run-time power monitor can easily become a primary source of power consumption, especially for those monitors leveraging the circuit-level switching activity, since the power consumption due to the power monitor is directly related to the switching activity and the bit width of the monitored circuit-level signals [47]. From a different, but related, perspective, the security of a run-time power monitor represents an additional non-functional metric to consider in the design methodology [127]. Side-channel attacks are implementation attacks that allow retrieving the secret key of an executed cryptographic primitive by correlating the partially known data being encrypted, i.e., the plaintext, with the power consumption of the computing platform performing such encryption [63, 108]. In this scenario, the power monitor can provide low-noise power estimates making it easier for malicious actors to carry out a side-channel attack [11]. To this end, further research must be devoted to designing secure run-time power monitors that carefully balance the security requirements, the overheads, and the quality of the power estimates.

Implementation of hierarchical run-time power monitors. Traditionally, the design of the run-time power monitor has targeted a single computing platform that was seen as a monolithic device. However, the ever-increasing complexity of current computing platforms may require novel solutions to the design of run-time power monitors. For example, complex heterogeneous systems can be seen as the composition of a set of hardware accelerators and a set of general-purpose processors [33]. Conversely, large FPGAs can offer enough space to allocate multiple hardware accelerators, each of whom may be subject to different constraints or requirements, imposing the design of ad-hoc power monitors for each part of the system. Whereas state-of-the-art power monitors can already operate at component-level granularity to compose the dynamic power consumption of a single design such as a CPU [43, 119], we encourage research into designing future run-time power monitors that can provide effective estimates on those highly heterogeneous platforms with a multitude of different CPU cores and accelerators.

ACKNOWLEDGMENTS

This work was supported by the EU Horizon 2020 *TEXTAROSSA* project (Grant No. 956831).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 265–283. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] Peter Alfke. 2009. Xilinx Virtex-6 and Spartan-6 FPGA families. In *2009 IEEE Hot Chips 21 Symposium (HCS)*. 1–20. https://doi.org/10.1109/HOTCHIPS.2009.7478379

[3] Jan Andersson, Magnus Hjorth, Fredrik Johansson, and Sandi Habinc. 2017. LEON Processor Devices for Space Missions: First 20 Years of LEON in Space. In *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. 136–141. https://doi.org/10.1109/SMC-IT.2017.31

[4] Arm. 2014. Gator daemon, driver and related tools. https://github.com/ARM-software/gator. (2014). [Online; accessed 27-May-2022].

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2010. A View of Cloud Computing. *Commun. ACM* 53, 4 (apr 2010), 50–58. https://doi.org/10.1145/1721654.1721672

[6] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator*. Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[7] P. Bannon and J. Keller. 1995. Internal architecture of Alpha 21164 microprocessor. In *Digest of Papers. COMPCON'95. Technologies for the Information Superhighway*. 79–87. https://doi.org/10.1109/CMPCON.1995.512368

[8] Lyonel Barthe, Luis Vitorio Cargnini, Pascal Benoit, and Lionel Torres. 2011. The SecretBlaze: A Configurable and Cost-Effective Open-Source Soft-Core Processor. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 310–313. https://doi.org/10.1109/IPDPS.2011.154

[9] Frank Bellosa. 2000. The Benefits of Event: Driven Energy Accounting in Power-Sensitive Systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System (EW 9)*. Association for Computing Machinery, New York, NY, USA, 37–42. https://doi.org/10.1145/566726.566736

[10] D. Bhandarkar and J. Ding. 1997. Performance characterization of the Pentium Pro processor. In *Proceedings Third International Symposium on High-Performance Computer Architecture*. 288–297. https://doi.org/10.1109/HPCA.1997.569689

[11] Sarani Bhattacharya and Debdeep Mukhopadhyay. 2018. Utilizing Performance Counters for Compromising Public Key Ciphers. *ACM Trans. Priv. Secur.* 21, 1, Article 5 (jan 2018), 31 pages. https://doi.org/10.1145/3156015

[12] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT '08)*. Association for Computing Machinery, New York, NY, USA, 72–81. https://doi.org/10.1145/1454115.1454128

[13] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. 2016. Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods. *ACM Comput. Surv.* 49, 3, Article 41 (Sept. 2016), 27 pages. https://doi.org/10.1145/2962131

[14] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. *SIGARCH Comput. Archit. News* 28, 2 (May 2000), 83–94. https://doi.org/10.1145/342001.339657

[15] Thomas D. Burd and Robert W. Brodersen. 2000. Design Issues for Dynamic Voltage Scaling. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED '00)*. Association for Computing Machinery, New York, NY, USA, 9–14. https://doi.org/10.1145/344166.344181

[16] Christopher Celio, Pi-Feng Chiu, Krste Asanović, Borivoje Nikolić, and David Patterson. 2019. BROOM: An Open-Source Out-of-Order Processor With Resilient Low-Voltage Operation in 28-nm CMOS. *IEEE Micro* 39, 2 (2019), 52–60. https://doi.org/10.1109/MM.2019.2897782

[17] A.P. Chandrakasan and R.W. Brodersen. 1995. Minimizing power consumption in digital CMOS circuits. *Proc. IEEE* 83, 4 (1995), 498–523. https://doi.org/10.1109/5.371964

[18] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. 1992. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits* 27, 4 (April 1992), 473–484. https://doi.org/10.1109/4.126534

[19] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[20] J. Chen, Bin Li, Ying Zhang, L. Peng, and J. Peir. 2011. Statistical GPU power analysis using tree-based methods. In *2011 International Green Computing Conference and Workshops*. 1–6. https://doi.org/10.1109/IGCC.2011.6008582

[21] Jiasi Chen and Xukan Ran. 2019. Deep Learning With Edge Computing: A Review. *Proc. IEEE* 107, 8 (2019), 1655–1674. https://doi.org/10.1109/JPROC.2019.2921977

[22] François Chollet. 2015. Keras. https://keras.io/. (2015). [Online; accessed 27-May-2022].

[23] Luca Cremona, William Fornaciari, and Davide Zoni. 2020. Automatic identification and hardware implementation of a resource-constrained power model for embedded systems. *Sustainable Computing: Informatics and Systems* (2020), 100467. https://doi.org/10.1016/j.suscom.2020.100467

[24] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-3)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/1735688.1735702

[25] Arnaldo Carvalho De Melo. 2010. The new Linux 'perf' tools. In *Slides from Linux Kongress*, Vol. 18. 1–42.

[26] Monica Donno, Alessandro Ivaldi, Luca Benini, and Enrico Macii. 2003. Clock-Tree Power Optimization Based on RTL Clock-Gating. In *Proceedings of the 40th Annual Design Automation Conference (DAC '03)*. Association for Computing Machinery, New York, NY, USA, 622–627. https://doi.org/10.1145/775832.775989

[27] Jerome H. Friedman. 1991. Multivariate Adaptive Regression Splines. *The Annals of Statistics* 19, 1 (1991), 1 – 67. https://doi.org/10.1214/aos/1176347963

[28] Mirko Gagliardi, Edoardo Fusella, and Alessandro Cilardo. 2018. Improving Deep Learning with a customizable GPU-like FPGA-based accelerator. In *2018 14th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. 273–276. https://doi.org/10.1109/PRIME.2018.8430335

[29] Neel Gala, Arjun Menon, Rahul Bodduna, G. S. Madhusudan, and V. Kamakoti. 2016. SHAKTI Processors: An Open-Source Hardware Initiative. In *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*. 7–8. https://doi.org/10.1109/VLSID.2016.130

[30] Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K. Gürkaynak, and Luca Benini. 2017. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 10 (2017), 2700–2713. https://doi.org/10.1109/TVLSI.2017.2654506

[31] G. H. Golub and C. Reinsch. 1971. *Singular Value Decomposition and Least Squares Solutions*. Springer Berlin Heidelberg, Berlin, Heidelberg, 134–151. https://doi.org/10.1007/978-3-662-39778-7_10

[32] John C. Gordon, Andrew T. Fenley, and Alexey Onufriev. 2008. An analytical approach to computing biomolecular electrostatic potential. II. Validation and applications. *The Journal of Chemical Physics* 129, 7 (2008), 075102. https://doi.org/10.1063/1.2956499 arXiv:https://doi.org/10.1063/1.2956499

[33] Jan Gray. 2016. GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 17–20. https://doi.org/10.1109/FCCM.2016.12

[34] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. 2010. The Mälardalen WCET Benchmarks: Past, Present And Future. In *10th International Workshop on Worst-Case Execution Time Analysis, WCET 2010, July 6, 2010, Brussels, Belgium (OASICS)*, Björn Lisper (Ed.), Vol. 15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 136–146. https://doi.org/10.4230/OASIcs.WCET.2010.136

[35] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*. 3–14. https://doi.org/10.1109/WWC.2001.990739

[36] Zacharias Hadjilambrou, Shidhartha Das, Paul N Whatmough, David Bull, and Yiannakis Sazeides. 2019. GeST: An Automatic Framework For Generating CPU Stress-Tests. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 1–10. https://doi.org/10.1109/ISPASS.2019.00009

[37] Tom R Halfhill. 2008. Intel's tiny atom. *Microprocessor Report* 22, 4 (2008), 1.

[38] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii. 2008. CHStone: A benchmark program suite for practical C-based high-level synthesis. In *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1192–1195. https://doi.org/10.1109/ISCAS.2008.4541637

[39] J.L. Henning. 2000. SPEC CPU2000: measuring CPU performance in the New Millennium. *Computer* 33, 7 (2000), 28–35. https://doi.org/10.1109/2.869367

[40] John L. Henning. 2006. SPEC CPU2006 Benchmark Descriptions. *SIGARCH Comput. Archit. News* 34, 4 (sep 2006), 1–17. https://doi.org/10.1145/1186736.1186737

[41] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Desktop Platforms Group, and Intel Corp. 2001. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal* 1 (2001), 2001.

[42] Zhigang Hu, Alper Buyuktosunoglu, Viji Srinivasan, Victor Zyuban, Hans Jacobson, and Pradip Bose. 2004. Microarchitectural Techniques for Power Gating of Execution Units. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED '04)*. Association for Computing Machinery, New York, NY, USA, 32–37. https://doi.org/10.1145/1013235.1013249

[43] C. Isci and M. Martonosi. 2003. Runtime power monitoring in high-end processors: methodology and empirical data. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*. 93–104. https://doi.org/10.1109/MICRO.2003.1253186

[44] Tohru Ishihara and Hiroto Yasuura. 1998. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED '98)*. Association for Computing Machinery, New York, NY, USA, 197–202. https://doi.org/10.1145/280756.280894

[45] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach. 2017. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 209–216. https://doi.org/10.1109/ICCAD.2017.8203780

[46] Hailin Jiang, M. Marek-Sadowska, and S.R. Nassif. 2005. Benefits and costs of power-gating technique. In *2005 International Conference on Computer Design*. 559–566. https://doi.org/10.1109/ICCD.2005.34

[47] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai, and Tin-Fook Ngai. 2015. Early stage real-time SoC power estimation using RTL instrumentation. In *The 20th Asia and South Pacific Design Automation Conference*. 779–784. https://doi.org/10.1109/ASPDAC.2015.7059105

[48] R. Joseph and M. Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design (IEEE Cat. No.01TH8581)*. 135–140. https://doi.org/10.1109/LPE.2001.945389

[49] A. B. Kahng, B. Lin, and S. Nath. 2015. ORION3.0: A Comprehensive NoC Router Estimation Tool. *IEEE Embedded Systems Letters* 7, 2 (June 2015), 41–45. https://doi.org/10.1109/LES.2015.2402197

[50] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. 2019. Edge computing: A survey. *Future Generation Computer Systems* 97 (2019), 219–235. https://doi.org/10.1016/j.future.2019.02.050

[51] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanovicc. 2016. Strober: Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 128–139. https://doi.org/10.1109/ISCA.2016.21

[52] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 1050–1062. https://doi.org/10.1145/3352460.3358322

[53] V. Klema and A. Laub. 1980. The singular value decomposition: Its computation and some applications. *IEEE Trans. Automat. Control* 25, 2 (1980), 164–176. https://doi.org/10.1109/TAC.1980.1102314

[54] Fanxin Kong and Xue Liu. 2014. A Survey on Green-Energy-Aware Power Management for Datacenters. *ACM Comput. Surv.* 47, 2, Article 30 (Nov. 2014), 38 pages. https://doi.org/10.1145/2642708

[55] Damjan Lampret, Chen-Min Chen, Marko Mlinar, Johan Rydberg, Matan Ziv-Av, Chris Ziomkowski, Greg McGary, Bob Gardner, Rohit Mathur, and Maria Bolado. 2003. Openrisc 1000 architecture manual. *Description of assembler mnemonics and other for OR1200* (2003).

[56] Chunho Lee, M. Potkonjak, and W.H. Mangione-Smith. 1997. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of 30th Annual International Symposium on Microarchitecture*. 330–335. https://doi.org/10.1109/MICRO.1997.645830

[57] Yunsup Lee, Andrew Waterman, Rimas Avizienis, Henry Cook, Chen Sun, Vladimir Stojanović, and Krste Asanović. 2014. A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators. In *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*. 199–202. https://doi.org/10.1109/ESSCIRC.2014.6942056

[58] Hai Li, S. Bhunia, Y. Chen, T.N. Vijaykumar, and K. Roy. 2003. Deterministic clock gating for microprocessor power reduction. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003.*

Proceedings. 113–122. https://doi.org/10.1109/HPCA.2003.1183529
[59] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. 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).* 469–480.
[60] Zhe Lin, Wei Zhang, and Sinha Sharad. 2017. Decision tree based hardware power monitoring for run time dynamic power management in FPGA. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL).* 1–8. https://doi.org/10.23919/FPL.2017.8056832
[61] Dake Liu and C. Svensson. 1994. Power consumption estimation in CMOS VLSI chips. *IEEE Journal of Solid-State Circuits* 29, 6 (1994), 663–670. https://doi.org/10.1109/4.293111
[62] Roy Longbottom. 2016. Roy Longbottom's PC Benchmark Collection. http://www.roylongbottom.org.uk/. (2016). [Online; accessed 27-May-2022].
[63] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. 2021. A Survey of Microarchitectural Side-Channel Vulnerabilities, Attacks, and Defenses in Cryptography. *ACM Comput. Surv.* 54, 6, Article 122 (jul 2021), 37 pages. https://doi.org/10.1145/3456629
[64] Yung-Hsiang Lu, L. Benini, and G. De Micheli. 2002. Dynamic frequency scaling with buffer insertion for mixed workloads. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21, 11 (2002), 1284–1305. https://doi.org/10.1109/TCAD.2002.804087
[65] Sulav Malla and Ken Christensen. 2019. A Survey on Power Management Techniques for Oversubscription of Multi-Tenant Data Centers. *ACM Comput. Surv.* 52, 1, Article 1 (Feb. 2019), 31 pages. https://doi.org/10.1145/3291049
[66] N. Mammeri, M. Neu, S. Lal, and B. Juurlink. 2019. Performance Counters based Power Modeling of Mobile GPUs using Deep Learning. In *2019 International Conference on High Performance Computing Simulation (HPCS).* 193–200. https://doi.org/10.1109/HPCS48598.2019.9188139
[67] C. E. McHenry. 1978. Computation of a Best Subset in Multivariate Analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 27, 3 (1978), 291–296. https://doi.org/10.2307/2347164 arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.2307/2347164
[68] Peter Mell, Tim Grance, et al. 2011. The NIST definition of cloud computing. (2011).
[69] Christoph Möbius, Waltenegus Dargie, and Alexander Schill. 2014. Power Consumption Estimation Models for Processors, Virtual Machines, and Servers. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2014), 1600–1614. https://doi.org/10.1109/TPDS.2013.183
[70] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. 2010. Statistical power modeling of GPU kernels using performance counters. In *International Conference on Green Computing.* 115–122. https://doi.org/10.1109/GREENCOMP.2010.5598315
[71] M. Najem, P. Benoit, M. El Ahmad, G. Sassatelli, and L. Torres. 2017. A Design-Time Method for Building Cost-Effective Run-Time Power Monitoring. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 7 (July 2017), 1153–1166. https://doi.org/10.1109/TCAD.2016.2614347
[72] Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, Jason Anderson, and Koen Bertels. 2016. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (2016), 1591–1604. https://doi.org/10.1109/TCAD.2015.2513673
[73] Stephen Neuendorffer and Fernando Martinez-Vallina. 2013. Building Zynq® Accelerators with Vivado® High Level Synthesis. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '13).* Association for Computing Machinery, New York, NY, USA, 1–2. https://doi.org/10.1145/2435264.2435266
[74] Kris Nikov, Kyriakos Georgiou, Zbigniew Chamski, Kerstin Eder, and Jose Nunez-Yanez. 2022. Accurate Energy Modelling on the Cortex-M0 Processor for Profiling and Static Analysis. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS).* 1–4. https://doi.org/10.1109/ICECS202256217.2022.9971086
[75] Kris Nikov, Marcos Martínez, Simon Wegener, Jose Nunez-Yanez, Zbigniew Chamski, Kyriakos Georgiou, and Kerstin Eder. 2022. Robust and Accurate Fine-Grain Power Models for Embedded Systems With No On-Chip PMU. *IEEE Embedded Systems Letters* 14, 3 (2022), 147–150. https://doi.org/10.1109/LES.2022.3147308
[76] Krastin Nikov and Jose Nunez-Yanez. 2020. Intra and inter-core power modelling for single-ISA heterogeneous processors. *International Journal of Embedded Systems* 12, 3 (2020), 324–340. https://doi.org/10.1504/IJES.2020.107046 arXiv:https://www.inderscienceonline.com/doi/pdf/10.1504/IJES.2020.107046
[77] K. Nikov, J. L. Nunez-Yanez, and M. Horsnell. 2015. Evaluation of Hybrid Run-Time Power Models for the ARM Big.LITTLE Architecture. In *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing.* 205–210. https://doi.org/10.1109/EUC.2015.32
[78] Jose Nunez-Yanez, Kris Nikov, Kerstin Eder, and Mohammad Hosseinabady. 2020. Run-Time Power Modelling in Embedded GPUs with Dynamic Voltage and Frequency Scaling. In *Proceedings of the 11th Workshop on Parallel Programming and Run-Time Management Techniques for Many-Core Architectures / 9th Workshop on Design Tools*

and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM'2020). Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3381427.3381429

[79] Nvidia. 2022. CUDA Zone. https://developer.nvidia.com/cuda-zone. (2022). [Online; accessed 27-May-2022].

[80] Nvidia. 2022. The API reference guide for CUPTI, the CUDA Profiling Tools Interface. https://docs.nvidia.com/cuda/cupti/index.html. (2022). [Online; accessed 27-May-2022].

[81] OpenRISC. 2022. mor1kx - an OpenRISC processor IP core. https://github.com/openrisc/mor1kx. (2022). [Online; accessed 27-May-2022].

[82] Kenneth O'brien, Ilia Pietri, Ravi Reddy, Alexey Lastovetsky, and Rizos Sakellariou. 2017. A Survey of Power and Energy Predictive Models in HPC Systems and Applications. *ACM Comput. Surv.* 50, 3, Article 37 (jun 2017), 38 pages. https://doi.org/10.1145/3078811

[83] D. J. Pagliari, V. Peluso, Y. Chen, A. Calimera, E. Macii, and M. Poncino. 2018. All-digital embedded meters for on-line power estimation. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE).* 737–742. https://doi.org/10.23919/DATE.2018.8342105

[84] James Pallister, Simon Hollis, and Jeremy Bennett. 2013. BEEBS: Open benchmarks for energy measurements on embedded platforms. *arXiv preprint arXiv:1308.5174* (2013).

[85] D.B. Papworth. 1996. Tuning the Pentium Pro microarchitecture. *IEEE Micro* 16, 2 (1996), 8–15. https://doi.org/10.1109/40.491458

[86] J. Peddersen and S. Parameswaran. 2007. CLIPPER: Counter-based Low Impact Processor Power Estimation at Run-time. In *2007 Asia and South Pacific Design Automation Conference.* 890–895. https://doi.org/10.1109/ASPDAC.2007.358102

[87] Jorgen Peddersen, Seng Lin Shee, Andhi Janapsatya, and Sri Parameswaran. 2005. Rapid Embedded Hardware/Software System Generation. In *Proceedings of the 18th International Conference on VLSI Design Held Jointly with 4th International Conference on Embedded Systems Design (VLSID '05).* IEEE Computer Society, USA, 111–116. https://doi.org/10.1109/ICVD.2005.145

[88] T. Pering, T. Burd, and R. Brodersen. 1998. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No.98TH8379).* 76–81. https://doi.org/10.1145/280756.280790

[89] C. Pilato and F. Ferrandi. 2013. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In *2013 23rd International Conference on Field programmable Logic and Applications.* 1–4. https://doi.org/10.1109/FPL.2013.6645550

[90] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin. 2013. Power-performance modeling on asymmetric multi-cores. In *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES).* 1–10. https://doi.org/10.1109/CASES.2013.6662519

[91] Brent Przybus. 2010. Xilinx Redefines Power, Performance, and Design Productivity with Three New 28 nm FPGA Families: Virtex-7, Kintex-7, and Artix-7 Devices. *Xilinx White Paper* (2010).

[92] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks. 2014. MachSuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC).* 110–119. https://doi.org/10.1109/IISWC.2014.6983050

[93] 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 *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO '14).* Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages. https://doi.org/10.1145/2555486.2555491

[94] R. Rodrigues, A. Annamalai, I. Koren, and S. Kundu. 2013. A Study on the Use of Performance Counters to Estimate Power in Microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs* 60, 12 (Dec 2013), 882–886. https://doi.org/10.1109/TCSII.2013.2285966

[95] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (2017), 30–39. https://doi.org/10.1109/MC.2017.9

[96] Giovanni Scotti and Davide Zoni. 2019. A Fresh View on the Microarchitectural Design of FPGA-Based RISC CPUs in the IoT Era. *Journal of Low Power Electronics and Applications* 9, 1 (2019). https://doi.org/10.3390/jlpea9010009

[97] Tom Shanley. 1998. *Pentium Pro and Pentium II system architecture.* Addison-Wesley Professional.

[98] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[99] Youngsoo Shin, Jun Seomun, Kyu-Myung Choi, and Takayasu Sakurai. 2010. Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-Cell VLSI Designs. *ACM Trans. Des. Autom. Electron. Syst.* 15, 4, Article 28 (oct 2010), 37 pages. https://doi.org/10.1145/1835420.1835421

[100] Tajana Simunic, Luca Benini, Andrea Acquaviva, Peter Glynn, and Giovanni De Micheli. 2001. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proceedings of the 38th Annual Design Automation Conference (DAC '01).* Association for Computing Machinery, New York, NY, USA, 524–529. https://doi.org/10.1145/378239.379016

[101] Ronak Singhal. 2008. Inside Intel® Core microarchitecture (Nehalem). In *2008 IEEE Hot Chips 20 Symposium (HCS)*. 1–25. https://doi.org/10.1109/HOTCHIPS.2008.7476555

[102] S. Song, C. Su, B. Rountree, and K. W. Cameron. 2013. A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 673–686. https://doi.org/10.1109/IPDPS.2013.73

[103] SPEC. 1995. SPEC CPU95. https://www.spec.org/cpu95/. (1995). [Online; accessed 27-May-2022].

[104] B. Sprunt. 2002. Pentium 4 performance-monitoring features. *IEEE Micro* 22, 4 (2002), 72–82. https://doi.org/10.1109/MM.2002.1028478

[105] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. 2012. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing* 127 (2012), 29.

[106] C. Sun, C. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. Peh, and V. Stojanovic. 2012. 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*. 201–210. https://doi.org/10.1109/NOCS.2012.31

[107] Synopsys. 2022. Synopsys PrimePower. https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html. (2022). [Online; accessed 27-May-2022].

[108] Jakub Szefer. 2019. Survey of microarchitectural side and covert channels, attacks, and defenses. *Journal of Hardware and Systems Security* 3, 3 (2019), 219–234.

[109] Andreas Traber, Florian Zaruba, Sven Stucki, Antonio Pullini, Germain Haugou, Eric Flamand, Frank K Gurkaynak, and Luca Benini. 2016. PULPino: A small single-core RISC-V SoC. In *3rd RISCV Workshop*.

[110] Sravanthi Kota Venkata, Ikkjin Ahn, Donghwan Jeon, Anshuman Gupta, Christopher Louie, Saturnino Garcia, Serge Belongie, and Michael Bedford Taylor. 2009. SD-VBS: The San Diego Vision Benchmark Suite. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 55–64. https://doi.org/10.1109/IISWC.2009.5306794

[111] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett. 2017. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 1 (Jan 2017), 106–119. https://doi.org/10.1109/TCAD.2016.2562920

[112] Michael E. Wall, Andreas Rechtsteiner, and Luis M. Rocha. 2003. *Singular Value Decomposition and Principal Component Analysis*. Springer US, Boston, MA, 91–109. https://doi.org/10.1007/0-306-47815-3_5

[113] Po-Han Wang, Chia-Lin Yang, Yen-Ming Chen, and Yu-Jung Cheng. 2011. Power Gating Strategies on GPUs. *ACM Trans. Archit. Code Optim.* 8, 3, Article 13 (oct 2011), 25 pages. https://doi.org/10.1145/2019608.2019612

[114] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. 2016. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1*. Technical Report UCB/EECS-2016-118. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html

[115] Craig M. Wittenbrink, Emmett Kilgariff, and Arjun Prabhu. 2011. Fermi GF100 GPU Architecture. *IEEE Micro* 31, 2 (2011), 50–59. https://doi.org/10.1109/MM.2011.24

[116] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*.

[117] Qing Wu, M. Pedram, and Xunwei Wu. 2000. Clock-gating and its application to low power design of sequential circuits. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 47, 3 (2000), 415–420. https://doi.org/10.1109/81.841927

[118] S. L. Xi, H. Jacobson, P. Bose, G. Y. Wei, and D. Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 577–589. https://doi.org/10.1109/HPCA.2015.7056064

[119] 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 *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22)*. Association for Computing Machinery, New York, NY, USA, Article 76, 9 pages. https://doi.org/10.1145/3508352.3549427

[120] 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: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3466752.3480064

[121] Tomofumi Yuki. 2014. Understanding PolyBench/C 3.2 Kernels. In *International Workshop on Polyhedral Compilation Techniques (IMPACT)*. 1–5.

[122] Florian Zaruba and Luca Benini. 2019. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 11 (2019), 2629–2640. https://doi.org/10.1109/TVLSI.2019.2926114

[123] Bo Zhai, David Blaauw, Dennis Sylvester, and Krisztian Flautner. 2004. Theoretical and Practical Limits of Dynamic Voltage Scaling. In *Proceedings of the 41st Annual Design Automation Conference (DAC '04)*. Association for Computing Machinery, New York, NY, USA, 868–873. https://doi.org/10.1145/996566.996798

[124] Cun-Hui Zhang. 2010. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics* 38, 2 (2010), 894 – 942. https://doi.org/10.1214/09-AOS729

[125] Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, and William Fornaciari. 2018. PowerTap: All-digital power meter modeling for run-time power monitoring. *Microprocessors and Microsystems* 63 (2018), 128–139. https://doi.org/10.1016/j.micpro.2018.07.007

[126] D. Zoni, L. Cremona, and W. Fornaciari. 2018. PowerProbe: Run-time power modeling through automatic RTL instrumentation. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 743–748. https://doi.org/10.23919/DATE.2018.8342106

[127] Davide Zoni, Luca Cremona, and William Fornaciari. 2022. Design of Side-Channel-Resistant Power Monitors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 5 (2022), 1249–1263. https://doi.org/10.1109/TCAD.2021.3088781