DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters

Zhiyao Xie[1], Shiyu Li[2], Mingyuan Ma[2], Chen-Chia Chang[2], Jingyu Pan[2], Yiran Chen[2], and Jiang Hu[3]
Hong Kong University of Science and Technology[1] Duke University[2] Texas A&M University[3]
eezhiyao@ust.hk, {shiyu.li, mingyuan.ma, chenchia.chang, jingyu.pan, yiran.chen}@duke.edu, jianghu@tamu.edu

ABSTRACT

Accurate and efficient on-chip power modeling is crucial to runtime power, energy, and voltage management. Such power monitoring can be achieved by designing and integrating on-chip power meters (OPMs) into the target design. In this work, we propose a new method named DEEP to automatically develop extremely efficient OPM solutions for a given design. DEEP selects OPM inputs from all individual bits in RTL signals. Such bit-level selection provides an unprecedentedly large number of input candidates and supports lower hardware cost, compared with signal-level selection in prior works. In addition, DEEP proposes a powerful two-step OPM input selection method, and it supports reporting both total power and the power of major design components. Experiments on a commercial microprocessor demonstrate that DEEP's OPM solution achieves correlation $R > 0.97$ in per-cycle power prediction with an unprecedented low area overhead on hardware, i.e., $< 0.1\%$ of the microprocessor layout. This reduces the OPM hardware cost by $4-6\times$ compared with the state-of-the-art solution.

1 INTRODUCTION

Power efficiency has become one of the primary design objectives for modern compute systems, ranging from low-end embedded systems, mobile computing to high-end data centers. As such, accurate while efficient power estimation is not only essential for design-time hardware design decisions, but also vitally important for power, energy, and voltage management during circuit runtime [38].

In practice, runtime circuit management techniques raise different requirements on the on-chip power estimation. The dynamic voltage and frequency scaling (DVFS), for example, only requires coarse-grained temporal resolution in power-tracing, where each estimation can be the average power over microseconds in duration. In contrast, techniques for fast power management [19], voltage boosting [13], or voltage noise mitigation [7, 17, 37] require much more fine-grained temporal resolution, updating power estimations within 10s of clock cycles. Therefore, ideal on-chip power monitoring in modern compute systems needs to be accurate, efficient, and support fine-grained temporal resolution. In addition, it should also be easily extensible to novel designs with automation. Despite extensive prior explorations, such a perfect on-chip power estimator is largely unattained.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549427

Previous works have utilized event counters to model runtime power in real microprocessors [3, 4, 6, 12, 14, 31]. These power models are based on counter-values of specific micro-architectural events, like cache misses and the number of retired instructions, across thousands or millions of clock cycles. These events do not naturally support power tracing in more fine-grained temporal resolution. Moreover, the development of such models requires extensive designer's knowledge of the target design to define the hardware events, and cannot be easily transferred to novel designs.

In recent years, on-chip power meter (OPM) based on selected RTL signals [9, 23, 25, 41] has been designed to improve temporal resolution and enable automated development, at the expense of high hardware implementation cost. Most recently, a solution [38] makes great progress by reducing the OPM hardware cost to $< 1\%$ area overhead in commercial microprocessors. However, 1% overhead of cutting edge microprocessor design is non-negligible and there is still huge room for improvement.

In this work, we propose a new methodology named DEEP to automatically construct an extremely low-cost on-chip power meter (OPM) in any given circuit design[1]. DEEP supports a truly negligible implementation cost thus making it feasible to be implemented in almost any design. The major contributions in this work are summarized below.

- We propose DEEP, which automatically develops OPM with $4-6\times$ lower cost compared with the state-of-the-art solution. It achieves correlation $R > 0.97$ in power prediction with area overhead $< 0.1\%$ of the microprocessor layout. This model accuracy is measured based on accurate post-layout power simulations, and hardware cost is verified on the actual OPM implementation on the design layout.
- DEEP proposes to select OPM input from all individual bits in RTL signals. Compared with signal-level selection in prior works, such bit-level selection provides an unprecedentedly large number of candidates and leads to a significantly lower hardware cost.
- DEEP proposes an innovative two-stage method to efficiently select high-quality OPM inputs from a huge number of input candidates.
- Besides estimating the power of the whole design, DEEP extends the OPM to also report the power of selected design components without extra hardware cost or accuracy loss.

2 PREVIOUS WORKS

There have been many prior studies in runtime on-chip power modeling. Besides a few analog solutions [5], popular previous methods can be categorized into two major types, *counter*-based

[1]The power model development methodology will be open-sourced in https://anonymous.4open.science/r/DEEP-3E6B/.

Baseline Methods	Model Input Candidate V_M (Candidate Count M)	Input Selection Method	Power Estimation Level	Temporal Resolution	Claimed OPM Area Overhead
B1. MICRO'21 [38]	All RTL signals (178 K)	MCP	Design-level	Per-cycle	< 1%
B2. MICRO'19 [20]	All RTL signals (178 K)	K-means	Design-level	100s cycles	N/A
B3. DATE'18 [25]	Registers (67 K)	Lasso	Design-level	> 1K cycles	7%
B4. DATE'18 [41]	Module I/O signals (< 178K)	Increase by level	Component-level	100s cycles	4 – 10%
B5. ASPDAC'15 [39]	Registers (67 K)	No Selection	Design-level	Per-cycle	16%
DEEP (this work)	All bits of RTL signals (578 K)	Two-step Selection	Component-level	Per-cycle	**< 0.1%**

Table 1: Overview of representative works in proxy-based on-chip power estimation.

and *proxy*-based. Most counter-based solutions [3, 4, 6, 10, 11, 14–16, 18, 24, 27–30, 36] utilize already existing performance counters in industrial designs like microprocessors or digital signal processors (DSPs). Such counters can be treated as free and the associated area overhead is minimum. However, the development of such counter-based power models requires extensive designers' knowledge of the specific design to define related hardware events. It restricts the automation of these power modeling models. Moreover, these counter-monitored hardware events manifest multiple cycles after the causal trigger event. It restricts the temporal resolution of estimations to thousands to millions of cycles.

In comparison, proxy-based runtime power models can be more friendly to automation, applicable to multiple designs, and support more fine-grained temporal resolutions. Prior works [9, 23, 25, 38, 41] select the most power correlated signals, named proxies, as inputs of the power model. Most of them target the best trade-off among hardware implementation cost, accuracy, and temporal resolution.

Besides these runtime power estimators, some works [8, 20, 32, 39] utilize proxy-based models to accelerate design-time power simulation by emulating on FPGA or other platforms. In the strict sense, these works are originally proposed to benefit power simulation at design time instead of runtime. But these power models actually can be extensible to runtime power monitoring when implemented on-chip.

Table 1 summarizes representative proxy-based power estimation methods with their power model input candidates, temporal resolution, and claimed OPM hardware cost measured in area overhead. The most recent and state-of-the-art work [38] reduces overhead to < 1% of the design layout.

In this work, DEEP satisfies almost all desired properties of an 'ideal' OPM development method. It automatically develops accurate OPMs for per-cycle temporal resolution with a negligible < 0.1% hardware overhead. Moreover, it enables reporting the power of selected components/modules, which supports more flexible and component-level power, energy, and voltage management techniques.

3 METHODOLOGY

The power consumption of circuits consists of both dynamic and leakage components. Since the leakage power remains rather invariant to the switching activity and code execution, it generally does not affect the runtime circuit management. Therefore, like many previous works [20, 38], DEEP focuses on developing OPM to monitor the runtime dynamic power of a given design. Since dynamic power is linearly proportional to charging/discharging of gate/wire-capacitance, which is reflected in signal transitions/toggling, the toggling of signals is detected on-chip and used as power model inputs. As demonstrated in many prior works [20, 38, 39], based on togging activities, the dynamic power can be reasonably approximated by an efficient linear power model.

3.1 Power Modeling Framework Overview

We first introduce DEEP's framework in developing proxy-based runtime OPM, as shown in Figure 1. There are three major stages. First, given an arbitrary design RTL and corresponding testbenches, signal waveforms and ground-truth power values are generated through simulation. Second, a power model is automatically developed for the given design. Third, the power model is implemented on hardware as the OPM and integrated as part of the target design. To reduce OPM hardware cost, only the most power-correlated design signals, named power proxies, can be selected as power model inputs.

DEEP targets per-cycle power prediction, which provides the best temporal resolution and is a most challenging scenario [38]. Given a design with altogether M input variable candidates V_M, DEEP selects a subset $V_Q \subset V_M$ with $Q = |V_Q| \ll M$, as power proxies, and Q is the number of proxies. Then a linear power model can be easily trained with these Q proxies as model inputs. For such per-cycle power model, it is deployed to estimate the power at the i^{th} clock cycle,

$$p[i] = \sum_{j=1}^{Q} w_j \cdot x_j[i] \tag{1}$$

where $x_1[i], x_2[i], ..., x_Q[i] \in \{0, 1\}$ are input features indicating the per-cycle togglings or transitions of Q proxies in the i^{th} clock

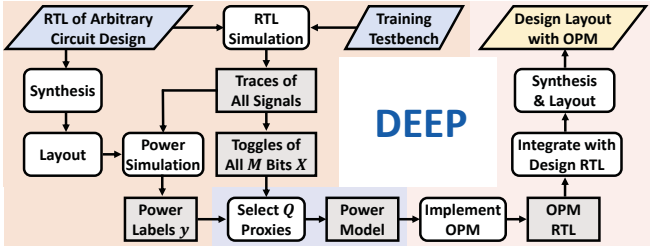


Figure 1: The DEEP OPM development framework.

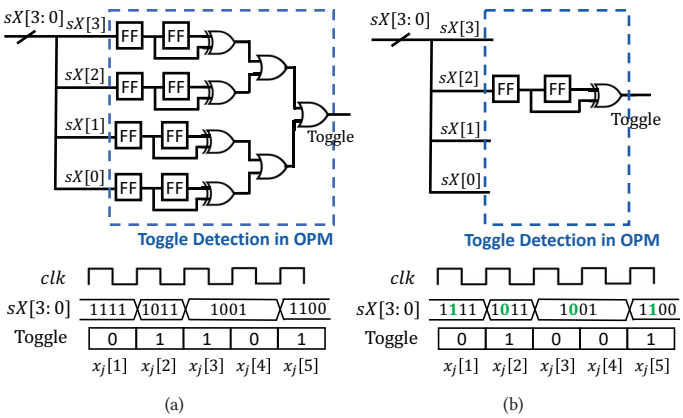


Figure 2: (a) Toggle detection of one bus signal $sX[3:0]$ as one model input in previous works [38]. (b) Toggle detection of only one individual bit $sX[2]$ as one model input in DEEP.

cycle, $w_1, w_2, ..., w_Q \in \mathbb{R}^+$ are trainable weights. Selecting power proxies V_Q from V_M with $Q \ll M$ can greatly reduce the hardware cost for runtime OPM. The choice of Q controls the trade-off between accuracy and efficiency.

3.2 Proxy Candidates in Power Model

Deciding the proxies V_Q as power model input is the key model development step since it determines the OPM accuracy and efficiency. This involves first deciding the candidates of model inputs V_M and then selecting V_Q from it.

In many previous works [20, 38, 41], inputs candidates V_M are all or part of available RTL signals. Once selected as proxies, the signal's toggling activities will be detected as model inputs. An example of toggle detection in recent prior work [38] is shown in Figure 2(a). To detect the toggling of a signal sX with a width of 4, every bit in sX is registered with one flip-flop and then monitored with a 1-bit toggle detector (one flip-flop and one XOR). As a result, its toggling activity is set to 1 if any bit in sX flips, otherwise it is 0. Assume sX is the j^{th} proxy, its toggling activities are model input $x_j[1], ..., x_j[N] \in \{0, 1\}$ in Equation 1. Besides this binary toggling value as model input, some other works [20, 41] adopt non-binary numbers for the toggling activity for each signal. This makes toggle detection and subsequent power calculation even more complex.

In the development framework, the OPM hardware cost is not known until OPM implementation finally finishes, making hardware-cost-aware model development difficult. Prior works [20, 38] thus use the proxy number Q as a metric to evaluate OPM hardware cost during model development. In this way, the proxy selection algorithm only needs to minimize Q for lower OPM cost. However, since they use RTL signals as candidates, *this metric is very misleading, because the hardware cost to detect different signals as proxies can be largely different.* As Figure 2(a) demonstrates, wide data bus signals, which are important proxies in practice, require much more circuitry to detect their toggling than single-bit signals. Therefore, the number of proxies Q does not accurately reflect OPM cost. In addition, using RTL signals as candidates forces selection algorithms either choose a wide bus signal or drop it, without other alternatives.

Algorithm 1 Power Proxy V_Q Selection Step 1

Input: Toggling activities $X \in \{0, 1\}^{N \times M}$. Power Label $y \in \mathbb{R}^N$. N is number of cycles, M is the number of all candidate variables.
Step 1: Pruning:

1: Initiate intermediate selection list $V_I = []$
2: Initiate a linear model with M weights w'_j. For the i^{th} cycle, power prediction is $o[i] = \sum_{j=1}^{M} w'_j \cdot X[i, j]$
3: Define MSE as error term $\mathcal{L} = ||o - y||_2$
4: Define MCP penalty term $\mathcal{P} = \sum_{j=1}^{M} P_{MCP}(w'_j)$
5: Training $w'_j = argmin(\mathcal{L} + \mathcal{P})$ with coordinate descent. Constrain $w'_j >= 0$ during training
6: **for** $j \in [1, M]$ **do**
7: add variable j to V_I if $w'_j \neq 0$

Output: The intermediate variable selection list V_I

In this work, DEEP proposes to select proxies at more the fine-grained bit-level instead of signal-level. In other words, DEEP allows the power model inputs V_M to be every individual bit instead of whole signals. As Figure 2(b) shows, it supports selecting individual bits like the $sX[2]$ from this bus signal. This difference leads to essentially different solutions. First, compared with the number of selected signals, the number of selected bits, which is Q in *this* work, can much more accurately reflect the real OPM hardware cost. This is because the cost to detect every bit is similar (two flip-flops and one XOR). Only in this case, minimizing Q leads to the most efficient OPM solutions. Second, by selecting individual bits instead of whole signals, for the same number of proxies Q, the hardware cost is greatly reduced, as indicated by the difference between Figure 2(a) and Figure 2(b). Third, this provides much more input candidates and flexible solutions. For the commercial microprocessor in our experiment, as shown in Table 1, there are 155 K RTL signals but altogether 578 K individual bits in the design. Therefore M = 578 K in the experiment. It indicates a larger solution space with potentially better OPM solutions, compared with using RTL signals [20, 38, 41] or only registers [25, 39] as V_M. This is further validated by results in Section 5.1.

3.3 Proxy Selection in Power Model Design

After candidates V_M are determined, DEEP proposes a two-step method to select proxies V_Q from V_M. In the first step, a powerful top-down pruning method is performed to narrow down the scope of variables from V_M to an intermediate input list V_I, where $I = V_I$ and $Q < I < M$. This pruning is a highly efficient method when exploring a huge number of candidates, but it does not directly lead to well-optimized final solution. This is further elaborated in Section 5.1. In the second step, a bottom-up selection method selects finalized V_Q from such intermediate result V_I.

The pruning-based first step is introduced in Algorithm 1. In this step, DEEP prunes all candidates with minimax concave penalty (MCP) [40] as defined below.

$$\text{For weight } w \in \mathbb{R},\ P_{MCP}(w) = \begin{cases} \lambda|w| - \frac{w^2}{2\gamma} & \text{if } |w| \leq \gamma\lambda \\ \frac{1}{2}\gamma\lambda^2 & \text{if } |w| > \gamma\lambda \end{cases}$$

MCP's superior performance over Lasso in OPM development has been proved in the latest prior work [38]. Compared with the popular Lasso penalty using L1 norm of weights, it protects large weights

Algorithm 2 Power Proxy V_Q Selection Step 2

Input: Toggling activities $X_I \in \{0,1\}^{N\times I}$ of selected bits V_I. Power Label $y \in \mathbb{R}^N$. N is number of cycles.

Step 2: Bottom-up Selection:

```
1:  function SELECTONEBESTVAR(V_Q)
2:      // Select one variable that adds most to accuracy
3:      Set R^2_best = 0, j_best = 0
4:      for signal j in [1, I] from V_I do
5:          Set temporary proxy list V_T = V_Q + [j]
6:          Select toggle activities of V_T, denote as X_I[V_T]
7:          Train a linear model M with X_I[V_T] and Y
8:          if Model M's accuracy R^2 > R^2_best then
9:              R^2_best = R^2;  j_best = j
10:     return j_best
11:
12: Initialize the proxy list V_Q = []
13: repeat Q times
14:     V_Q .append( SELECTONEBESTVAR(V_Q) )
15:     while V_Q still changes do
16:         for each proxy v in V_Q do
17:             // Check if better variable than v exists
18:             Remove v from V_Q
19:             V_Q .append( SELECTONEBESTVAR(V_Q) )
```

Output: The selected proxy list V_Q

under strong penalty strength. γ and λ are hyper-parameters deciding the penalty strength and the threshold of large weights. When the weight $w > \gamma\lambda$, the penalty of it $P_{MCP}(w)$ becomes a constant.

Algorithm 1 starts with a linear model with all M candidate variables as inputs and corresponding weights w'_j. After training with penalty $\mathcal{P}$ in the loss function, most weights will shrink to zero during training. Only variables with non-zero weights are added to the intermediate list V_I.

After the pruning, a bottom-up selection algorithm is performed to determine the finalized proxy list V_Q based on the intermediate selection V_I. It is based on the idea of best subset selection [21, 35], which targets to select a near-optimal subset of variables from V_I that yields the best performance. As Algorithm 2 shows, we first define a straightforward function named SELECTONEBESTVAR. It scans each of the variables in V_I and selects the one that adds the most to model accuracy, measured with the coefficient of determination R^2 [22].

In Algorithm 2, proxies V_Q are selected incrementally one-by-one. In each iteration, as line 14 shows, one signal that adds the most to model accuracy is selected and added to V_Q. This iteration does not stop here. Instead, it then goes through every already selected proxy v in V_Q and searches if any other variable would add more accuracy by replacing it. This is achieved by removing each v signal from V_Q, then search the whole candidate list V_I with SELECTONEBESTVAR in line 18 and 19. When there is no better signal than v, the SELECTONEBESTVAR will return the originally removed signal v, which is added back to the V_Q. This remove-replace process will continue until V_Q no longer changes (line 15).

The heuristic method in Algorithm 2 claims to find near-optimal subset solution efficiently [21, 35]. But compared with pruning in the first step, it is significantly slower when V_I is large and as

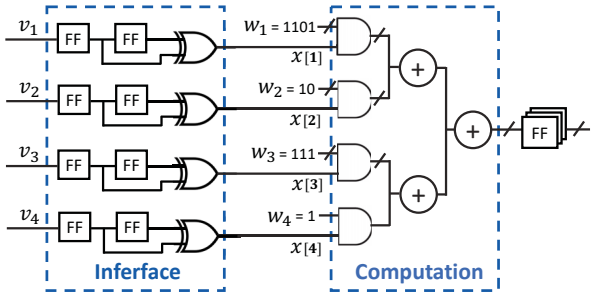


Figure 3: OPM hardware implementation with quantized weights. The outputs are per-cycle power values.

V_Q grows. It is prohibitively time-consuming if directly applied to the huge candidate list V_M. Therefore, we adopted the two-step selection, with the first step efficiently narrowing down a huge scope and the second step finalizing high-quality proxies.

3.4 Hardware Implementation

After proxies V_Q are determined, the final power model defined in Equation 1 is trained and implemented on hardware as OPM. Since we adopt individual bits as input variables, the OPM implementation is straightforward. As Figure 3 shows, it consists of two main parts. The interface part detects the toggling activities of proxies, and the computation part implements the linear power model by summing up all weights of toggling signals. For DEEP-OPM, the interface part takes up more area overhead than computation. The OPM output is per-cycle power estimation. Based on such per-cycle power, it is straightforward to further calculate multi-cycle average power if necessary.

To reduce the hardware cost of OPM, weight quantization is performed before implementing OPM on hardware. In DEEP, all weights are quantized to integers, which greatly reduces the hardware cost with limited accuracy loss. As Figure 3 shows, different weight values as integers correspond to different numbers of weight bits in the power computation part. These weights will be determined during OPM design time and will not be tuned after fabrication. At runtime, the OPM power estimations will be compared with preconfigured thresholds to initiate power or voltage management. To handle possible inter-chip variations, instead of adjusting model weights on a per-chip basis, post-silicon calibration of those preconfigured thresholds will be sufficient.

3.5 Component-Level Implementation

We have introduced the development of an OPM to estimate the total power of a design. In this work, DEEP can also estimate the power of selected major components in the design. To achieve this, the power model development method is applied to design components separately, resulting in multiple sub-OPMs. This means developing a sub-OPM for each component/module by restricting variable candidates V_M, toggling activities X, and power label y only to this component. An exception is for some memory components, where only I/O signals of the RAM macros are available on-chip. In this case, its candidates V_M can be selected from the whole design. Based on all these sub-OPMs, their outputs are summed up to report total the power of the design. Our experiment shows that this component-level DEEP does not lead to an extra OPM implementation cost.

#RTL Signal	#Register	#RTL Bit	#Standard Cell	#Macro
155 K	67 K	578 K	603 K	66

Table 2: Basic statistics of the microprocessor.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

Our experiment is performed on a 64-bit high-efficiency commercial microprocessor. It is two-way superscalar and supports SIMD and floating-point operations. In our experiment, this microprocessor is configured with one CPU core, 32 KB L1 instruction cache, 32 KB L1 data cache, and 1024 KB L2 cache. The L2 memory system includes the L2 cache pipeline and all logic required to maintain memory coherence.

We implement this design with an industrial 28nm technology node at 1 GHz. Design Compiler [1] and IC Compiler II [33] are adopted for logic synthesis and design layout, respectively. The RTL simulation is based on Synopsys VCS [2] and per-cycle power is simulated with PrimePower [34]. Different from some prior works [38] using inaccurate RTL-level power as the label, in this work, all power labels are simulated based on the post-routing layout solutions. The total area of the microprocessor layout is 4.05 mm^2, with standard cells occupying 0.74 mm^2 of the layout area.

Table 2 shows some basic statistics of this microprocessor implementation. There are around 155 thousand RTL signals and 67 thousand registers in the design. In comparison, when using all bits in RTL signals as candidates in this work, there are altogether 578 thousand bits as candidates V_M.

Figure 4 shows decomposed input candidate count and ground-truth power of major components/modules. According to this, to verify our component-level DEEP, we develop sub-OPMs for five selected major submodules in the microprocessor, including L1 cache with table lookaside buffer (TLB), L2 cache with the logic maintaining memory coherence, data processing unit (DPU) of core, instruction fetch unit (IFU) of core, and all other logic in the CPU core except DPU and IFU. They all contribute more than 10% of the total power consumption in implementation. Based on these sub-OPMs, the power of CPU core, CPU core+L1, and CPU core+L1+L2 (total power) are also calculated by the OPM. As mentioned, only I/O signals of RAM blocks in L1 and L2 caches are available in V_M, thus they only account for a very small portion of bits in Figure 4(a). Therefore, when developing sub-OPM for the L1 cache, its V_M is not limited to the L1 cache itself, but can be from the whole design. As for the L2 cache, as Figure 4(a) shows, there are many signals in the non-RAM logic. Thus like other components, the V_M of L2 is still within the L2 component.

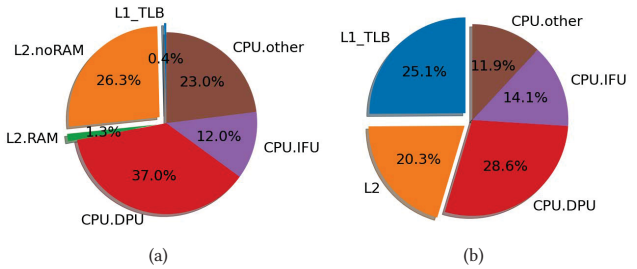


Figure 4: (a) Decomposition of all 578 K RTL bits as candidates V_M. (b) Decomposition of averaged power.

In this experiment, two strictly separated types of workloads are generated and simulated on the microprocessor to generate training (including validation) and testing data. Following the practice in [38], all testing workloads are designer-crafted representative power indicative workloads, including dhrystone, saxpy, cache_miss, maxpower, etc. Altogether the testing set consists of six designer-crafted workloads, with altogether $N_{test} = 26,000$ cycles. In comparison, workloads used in model training are first generated based on a random combination of various instructions, then selected to achieve good coverage of power consumption [38]. There are seven automatically generated workloads with $N_{train} = 17,000$ cycles in the training data.

The model development methods for both DEEP and all baselines are implemented with Python3. Some proxy selection baselines are based on scikit-learn [26]. The MCP algorithm with coordinate descent optimization is implemented from scratch. In the experiment, the two-step selection method in DEEP finishes in hours, with MCP converging in 100 epochs. The layout of microprocessor takes days to finish. Key hyperparameters are tuned based on the performance on validation data and are reported below. The γ in MCP is set to 5, and penalty strength α controls the size of V_I. We do not strictly control V_I size as we report many OPM solutions to demonstrate the accuracy and efficiency trade-off curve. As a rule of thumb according to our experiment, a reasonable range is $I \in [3\times Q, 30\times Q]$. The proxy size of DEEP $Q \in [100, 300]$ provides a good range of OPM accuracy and overhead.

4.2 Baseline Methods and Metrics

We compare DEEP with representative prior works on OPM design in Table 1. To measure OPM accuracy, we evaluate it with mean absolute error (MAE) percentage and Pearson correlation R between power label $y \in \mathbb{R}^N$ and prediction $p \in \mathbb{R}^N$. For the hardware cost of OPM solutions, we define two area overhead metrics AO_L and AO_C. AO_L equals the total gate area of all OPM components divided by the area of the whole layout region (4.05 mm^2). Considering the layout consists of many hard macros, to avoid underestimating the cost, AO_C divides the total gate area of OPM only by the area of all standard cells in the design (0.74 mm^2).

$$\text{MAE} = \frac{\sum_{i=1}^{N} |y[i] - p[i]|}{\sum_{i=1}^{N} y[i]}, \quad AO_L = \frac{\text{OPM gate area on layout}}{\text{The layout area}}$$

Almost all baselines are not open-sourced and we replicate them ourselves for comparisons. For baselines B2-B5 in Table 1, we focus on the replication of proxy selection methods to generate V_Q, which is the key part of OPM development. But we adjust some of their OPMs design with more efficient hardware implementation. For example, we eliminate all multipliers and perform weight quantization. Therefore the measured hardware cost in our experiment is much lower than they originally claimed in Table 1. But for the strongest baseline B1 [38], we strictly follow their originally proposed method for a fair comparison.

When comparing OPM solutions, the trade-off between accuracy vs. area overhead is the focus. We write an area overhead estimator to evaluate the gate count in an OPM and its area overhead AO_L based on proxies and weights in its power model. To ensure its correctness, this area estimator is calibrated with accurate area values measured by synthesizing multiple OPM implementations.

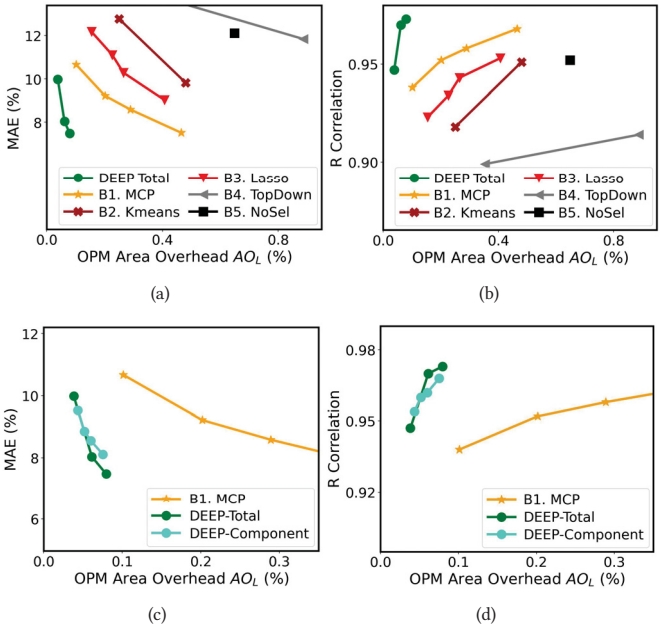


Figure 5: OPM hardware cost (AO_L) vs. per-cycle power prediction accuracy. Comparison with baselines B1 to B5. (a)(c) Accuracy measured in MAE percentage. (b)(d) Accuracy measured in R correlation.

In later sections, we will also validate the estimated area overhead with ground-truth area measurements on OPM-integrated layouts.

4.3 Performance Comparison

We first compare DEEP with all baseline methods in Figure 5, which shows OPM accuracy vs. the hardware cost in area overhead AO_L. Figure 5(a)(c) present accuracy in MAE percentage and (b)(d) show accuracy in R correlation. The top two subfigures present comparisons with all baselines B1-B5, with AO_L range in [0, 1%]. The bottom two subfigures further zoom in the comparison between DEEP with the strongest baselines B1 with AO_L range in [0, 0.3%].

In Figure 5(a)(b), we first observe that the most recent baseline B1 [38] indeed achieves a significantly superior accuracy-efficiency trade-off than the other four baseline solutions B2-B5. For baseline B5, since it uses all registers as the initial model input without explicit selection, there is no trade-off in the plot. We view B1 as the state-of-the-art solution and focus on comparisons with it in Figure 5(c)(d).

In Figure 5(c)(d), both versions of DEEP are shown, one only reporting the total power while the other reporting both total power and components' power. Their overall performances are very close.

In Figure 5(c)(d), DEEP significantly outperforms the B1 baseline. For the DEEP-Component solution achieving MAE = 9.5% and correlation R = 0.954, the area overhead AO_L is only around 0.04% of the whole layout area. This equals 0.24% of the total standard cell area. Compared with B1 at the same accuracy, the area overhead is reduced by 4×. For a more accurate DEEP solution with MAE = 7.5% and correlation R = 0.973, the AO_L is 0.08% of the layout, which equals 0.42% of total standard cell area. In comparison, the OPM area overhead in B1 at this accuracy is 6× of the DEEP.

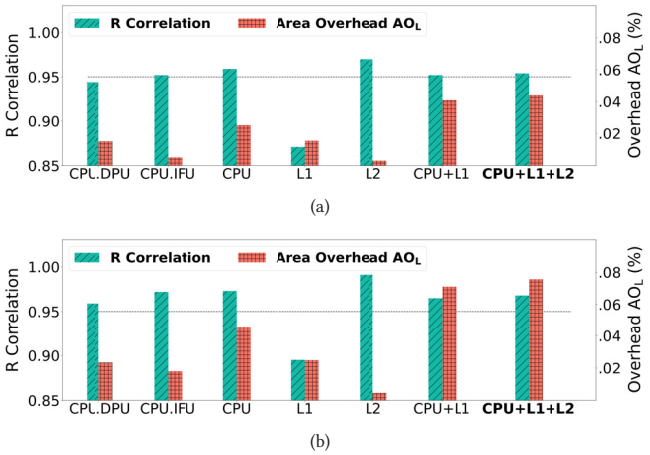


Figure 6: Component-level DEEP-OPM accuracy. (a) DEEP-OPM with total overhead AO_L = 0.04%. (b) DEEP-OPM with total overhead AO_L = 0.08%.

To further study the component-level DEEP solution, Figure 6 reports the accuracy and area overhead at different component levels. Figure 6(a) reports a DEEP-Component OPM with total AO_L = 0.04% and (b) reports a more accurate yet costly solution with total AO_L = 0.08%. As mentioned, the whole OPM consists of five major sub-OPMs, monitoring L1, L2, CPU.DPU, CPU.IFU, and other parts of CPU core. The total area overhead AO_L is roughly the summation of these five sub-OPMs. The addition of their outputs further provides the power of the CPU core, CPU core+L1, and CPU core+L1+L2 (total power). For each component, we measure the correlation between the ground-truth simulated power and power estimation of this part of circuit.

In Figure 6, the horizontal black dashed line indicates R correlation equals to 0.95. The overall accuracy of most components is high. For OPM with AO_L = 0.04%, the correlation $R > 0.94$ for all components except the single L1 cache. Similarly, for OPM with AO_L = 0.08%, the correlation $R >= 0.96$ for all components except L1. The inferior accuracy in L1 is caused by the limited available signals and bits in the L1 cache, as indicated in Figure 4(a). Although this challenge is already handled by setting V_M of L1 to be from the whole design, there are still fewer variables that correlate well with L1 power. The R = 0.9 for L1 in OPM with AO_L = 0.08%.

4.4 Hardware Solution

We verify DEEP's OPM solution on hardware implementation. The DEEP-component OPM with estimated overhead AO_L = 0.04% and R = 0.954 mentioned in both Figure 5 and Figure 6(a) is implemented. We integrate this OPM design into the microprocessor and generate the whole layout, as shown in Figure 7. In this layout, all cells in the OPM are colored in red. The large macros are L2 data RAMs and smaller macros are for L1 cache, TLB, and tag RAMs. Then we measure the ground-truth OPM area overhead on this post-layout microprocessor. The actual overhead turns out to be indeed AO_L = 0.04%, which validates that our area estimator is correct and accurately calibrated.

In comparison, a microprocessor layout with baseline B1's OPM is shown in Figure 8. Note that the macro locations are not fixed and are automatically placed by IC Compiler II, leading to a slightly

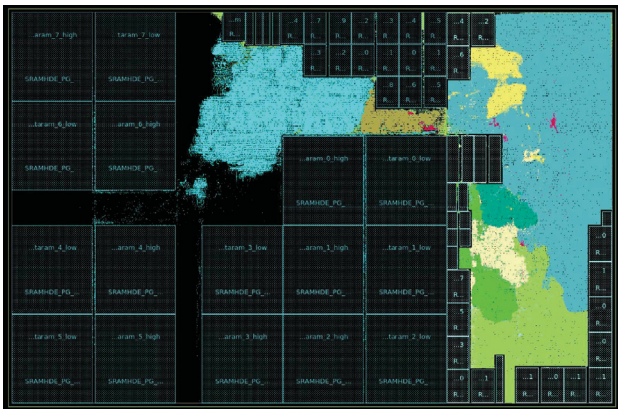


Figure 7: Microprocessor layout with DEEP component-level OPM integrated. The red region is OPM. Area overhead AO_L = 0.04% when measured on this layout. The MAE = 9.5% and R = 0.954.

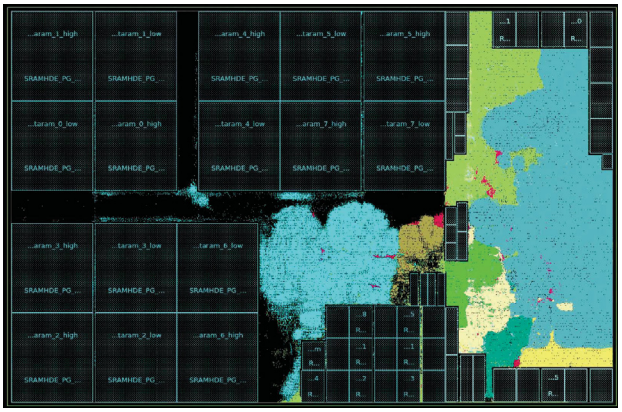


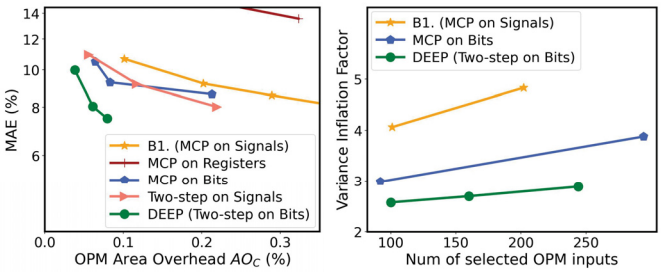
Figure 8: Microprocessor layout with state-of-the-art baseline B1 [38]-OPM. The red regions is OPM. Area overhead AO_L = 0.16%. The MAE = 9.5% and R = 0.951. Its overhead is 4× of the DEEP solution.

different floorplan solution. The actually measured AO_L on the layout in Figure 8 is 0.16%. Its accuracy is very close to the DEEP OPM shown in Figure 7 but with 4× area overhead. This measurement on layout implementation is consistent with the observation in Figure 5. When visually comparing the two layout solutions, the red OPM region in Figure 8 is also obviously larger than Figure 7.

5 DISCUSSION

5.1 Result Analysis

To better understand the unprecedented high efficiency of this DEEP algorithm, we decompose the contribution of two major new policies in DEEP through ablation studies in Figure 9(a). Compared with baseline B1, DEEP adopts a two-step proxy selection method on all individual bits. The contribution of using bits instead of signals as V_M is reflected by the superior performance of the blue curve 'MCP on Bits' over B1 baseline in Figure 9(a). In addition, we also measure only using registers as V_M in 'MCP on Registers', as adopted by baselines B3 and B5 in Table 1. The performance



(a) (b)

Figure 9: (a) Analysis of model development policies in DEEP. (b) The correlation among proxies V_Q measured by variance inflation factor (VIF).

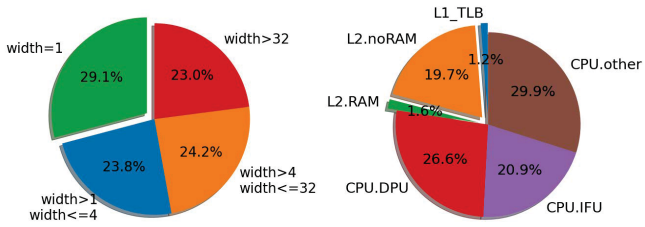
turns out to be very bad. This observation is consistent with the trend of adopting all RTL signals rather than only registers as input candidates in recent works [20, 38].

We also measure the accuracy of applying the two-step selection method on all signals as candidates, as the pink curve 'Two-step on Signals' in Figure 9(a) shows. Its superior performance over baseline B1 shows the contribution of adopting the two-step proxy selection over simple MCP.

5.2 Proxy Analysis

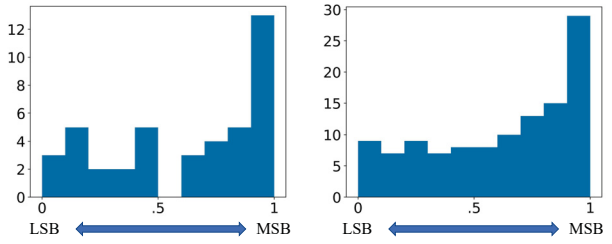
Besides ablation study on the accuracy, we also look into selected proxies V_Q, which directly determines OPM quality. Figure 9(b) calculates the averaged variance inflation factor (VIFs) of all proxies. The VIF reflects the collinearity or correlation among selected proxies. A high VIF generally implies less independence among selected proxies and high variance in the model, which can be improved. In Figure 9(b), DEEP shows better VIF than both baseline B1 and MCP selection on all bits as candidates. This implies that the two-step selection algorithm in DEEP tends not to select correlated candidates as proxies simultaneously, partially explaining the better performance of DEEP OPM.

Figure 10 further analyzes the source of proxies V_Q. There are 244 bits selected as inputs in DEEP-OPM with AO_L = 0.08%. Figure 10(a) inspects all original RTL signals from which these input bits are selected. Only 30% input bits are from per-bit signals and all other 70% input bits are selected from buses. Figure 10(b) presents the source of these selected input bits in terms of components.



(a) (b)

Figure 10: Analysis of 244 proxies V_Q in a DEEP OPM with AO_L = 0.08%. (a) Width of original RTL signals from which these bits proxies are selected. (b) Components where these bit proxies are selected.



(a) (b)

Figure 11: Histogram of proxy bit position in bus signals: MSB tends to be selected. (a) DEEP-OPM with AO_L = 0.04%. (b) DEEP-OPM with AO_L = 0.08%.

This distribution is similar to the distribution of candidates V_M in Figure 4(a).

Another interesting analysis is which bit in a bus signal tends to be selected as proxies. Figure 11 analyzes the selected bits from bus signals with width larger than 4. It inspects the 'bit position' of selected bits in V_Q. For a bus signal named $BS[\omega-1:0]$ with width ω, assuming one bit $BS[k]$ in this bus is selected as a proxy, we define its bit position to be $k/(\omega-1)$, which ranges from 0 to 1, with 0 representing the least significant bit (LSB) and 1 representing the most significant bit (MSB). As shown in two histograms in Figure 11, MSBs tend to be selected as inputs. This trend is very reasonable since toggles in MSB tend to capture more arithmetic activities due to carrying.

5.3 Weight Analysis

Besides proxies, the number of bits of weights after quantization also provides insights into understanding the OPM hardware cost. Table 3 reports the distribution of weight bits after quantization. Please do not confuse these weight bits with the width of RTL signals. For the OPM with overhead equal to 0.08%, the maximum weight bits is 6. In this OPM, more than 80% of weights only take less or equal to 3 bits. It indicates the importance of weight quantization.

DEEP-OPM	W bits	1	2	3	4	5	6
(AO_L = 0.08%)	Count	6	77	120	28	8	5

Table 3: Post-quantization weight bits distribution.

6 CONCLUSION

In this work, we present an extremely efficient runtime OPM development method named DEEP. It reduces the hardware cost of OPM to < 0.01% area overhead, making per-cycle on-chip power estimation affordable in almost any design. We believe this provides an 'ideal' runtime power estimator that supports all desired properties of OPM, including accuracy, low hardware cost, good temporal resolution, automated development, and reporting component-level power values.

7 ACKNOWLEDGEMENT

This work is partially supported by NSF-2106828, NSF-2106725, and SRC GRC-CADT 3103.001/3104.001. We thank Shidhartha Das and Xiaoqing Xu for helping us with their expertise on power modeling.

REFERENCES

[1] 2021. Design Compiler® RTL Synthesis. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html.
[2] 2021. VCS® functional verification solution. https://www.synopsys.com/verification/simulation/vcs.html.
[3] Frank Bellosa. 2000. The benefits of event: driven energy accounting in power-sensitive systems. In *ACM SIGOPS European Workshop (EW)*.
[4] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. 2010. Decomposable and responsive power models for multicore processors using performance counters. In *ACM ICS*.
[5] Srikar Bhagavatula and Byunghoo Jung. 2013. A power sensor with 80ns response time for power management in microprocessors. In *CICC*.
[6] W Lloyd Bircher and Lizy K John. 2007. Complete system power estimation: A trickle-down approach based on performance events. In *IEEE ISPASS*.
[7] Keith A Bowman, Sarthak Raina, J Todd Bridges, Daniel J Yingling, Hoan H Nguyen, Brad R Appel, Yesh N Kolla, Jihoon Jeong, Francois I Atallah, and David W Hansquine. 2016. A 16 nm All-Digital Auto-Calibrating Adaptive Clock Distribution for Supply Voltage Droop Tolerance Across a Wide Operating Range. *IEEE JSSC* (2016).
[8] Joel Coburn, Srivaths Ravi, and Anand Raghunathan. 2005. Power emulation: a new paradigm for power estimation. In *DAC*.
[9] Luca Cremona, William Fornaciari, and Davide Zoni. 2020. Automatic identification and hardware implementation of a resource-constrained power model for embedded systems. *Elsevier Sustainable Computing: Informatics and Systems* (2020).
[10] C Gilberto and M Margaret. 2005. Power prediction for intel xscale processors using performance monitoring unit events. In *ISLPED*.
[11] Bhavishya Goel, Sally A McKee, Roberto Gioiosa, Karan Singh, Major Bhadauria, and Marco Cesati. 2010. Portable, scalable, per-core power estimation for intelligent resource management. In *International Conference on Green Computing (IGCC)*.
[12] Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben Asher, and Avi Mendelson. 2016. Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems. *TACO* (2016).
[13] Chang-Hong Hsu, Yunqi Zhang, Michael A Laurenzano, David Meisner, Thomas Wenisch, Jason Mars, Lingjia Tang, and Ronald G Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *HPCA*.
[14] Wei Huang, Charles Lefurgy, William Kuk, Alper Buyuktosunoglu, Michael Floyd, Karthick Rajamani, Malcolm Allen-Ware, and Bishop Brock. 2012. Accurate fine-grained processor power proxies. In *MICRO*.
[15] Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*.
[16] R. Joseph and M. Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *ISLPED*.
[17] Vijay Kiran Kalyanam, Eric Mahurin, Keith Bowman, and Jacob Abraham. 2020. A Proactive Voltage-Droop-Mitigation System in a 7nm Hexagon™ Processor. In *VLSI*.
[18] Vijay Kiran Kalyanam, Peter G Sassone, and Jacob A Abraham. 2017. Power prediction of embedded scalar and vector processor: Challenges and solutions. In *ISQED*.
[19] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *MICRO*.
[20] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection. In *MICRO*.
[21] Claude E McHenry. 1978. Computation of a best subset in multivariate analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* (1978).
[22] Nico JD Nagelkerke et al. 1991. A note on a general definition of the coefficient of determination. *Biometrika* (1991).
[23] Mohamad Najem, Pascal Benoit, Mohamad El Ahmad, Gilles Sassatelli, and Lionel Torres. 2017. A design-time method for building cost-effective run-time power monitoring. *IEEE TCAD* (2017).
[24] Fabian Oboril, Jos Ewert, and Mehdi B Tahoori. 2015. High-resolution online power monitoring for modern microprocessors. In *DATE*.
[25] Daniele Jahier Pagliari, Valentino Peluso, Yukai Chen, Andrea Calimera, Enrico Macii, and Massimo Poncino. 2018. All-digital embedded meters for on-line power estimation. In *DATE*.
[26] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine Learning in Python. *JMLR* (2011).
[27] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Power-performance modeling on asymmetric multi-cores. In *CASES*.
[28] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. 2013. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs (TCAS-II)* (2013).

[29] Mark Sagi, Nguyen Anh Vu Doan, Martin Rapp, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf. 2020. A Lightweight Nonlinear Methodology to Accurately Model Multicore Processor Power. *IEEE TCAD* (2020).
[30] Karan Singh, Major Bhadauria, and Sally A McKee. 2009. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News* (2009).
[31] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L Greathouse, and Zhiying Wang. 2014. PPEP: Online performance, power, and energy prediction framework and DVFS space exploration. In *MICRO*.
[32] Dam Sunwoo, Gene Y Wu, Nikhil A Patil, and Derek Chiou. 2010. PrEsto: An FPGA-accelerated power estimation methodology for complex systems. In *FPL*.
[33] Synopsys. 2021. IC Compiler II for Physical Implementation. https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html.
[34] Synopsys. 2021. PrimePower RTL to Signoff Power Analysis. https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html.
[35] NCSS: Statistical System. 2021. NCSS User's Guide III: 310. Variable Selection for Multivariate Regression. https://www.ncss.com/software/ncss/regression-analysis-in-ncss/#Subset.
[36] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-Hashimi, and Geoff V Merrett. 2016. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE TCAD* (2016).
[37] T Webel, PM Lobo, T Strach, PB Parashurama, S Purushotham, R Bertran, and A Buyuktosunoglu. 2020. Proactive power management in IBM z15. *IBM Journal of Research and Development* (2020).
[38] 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*.
[39] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai, and Tin-Fook Ngai. 2015. Early stage real-time SoC power estimation using RTL instrumentation. In *ASPDAC*.
[40] Cun-Hui Zhang. 2010. Nearly unbiased variable selection under minimax concave penalty. *The Annals of statistics* (2010).
[41] Davide Zoni, Luca Cremona, and William Fornaciari. 2018. Powerprobe: Run-time power modeling through automatic RTL instrumentation. In *DATE*.