EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering

A regression-based power modelling approach for accurate dynamic power estimation in digital circuits

by

D. Monticelli

MSC THESIS

Assessment committee	
Chair:	Dr.Ir., S., Stuijk
Member 1:	Dr.Ir., J., Schellekens
Member 2:	Dr., M., Gomony
Advisory member 1:	S., De

Graduation	
Program:	Electrical Engineering
Capacity group:	Electronic Systems
Supervisor:	Dr., M., Gomony
Date of defense:	January 16, 2023
Student ID:	1610228
Study load (ECTS):	45
External supervisor(s):	S., De V.K.Y., Baapanapalli

A regression-based power modelling approach for accurate dynamic power estimation in digital circuits

Dario Monticelli
Student number: 1610228

***Abstract*—The increasing transistor density in integrated circuits (ICs) and the trend for low-power designs bring unforeseen challenges in estimating fundamental quality metrics such as power consumption in digital integrated circuits. This is due to the dynamic power figure being strongly dependent on the executed workload, which makes early-stage power estimation inaccurate with respect to final on-chip measurements. Therefore, there is a need for an approach that includes activity information to accurately evaluate different design solutions.**

This work addresses such a problem by proposing a learning-based approach for power modelling that takes into account the power contribution derived from the executed workload. Such power modelling is performed at the functional component level, in order to enable a modular approach for an activity-aware design space exploration (DSE). The power modelling is divided into two steps: a characterization-phase step and a run-phase step. During the first step, the power model is derived by collecting power data for various component's hardware characteristics and activity information. In the second step, power can be instantly inferred providing the selected component's configuration and the corresponding workload.

The approach is validated on a register transfer level (RTL) implementation of a finite impulse response (FIR) filter. At the top level, counting only the components' contribution, the measured maximum error is 2.11% and 1.59% for the 8 bit and 16 bit versions, respectively. This approach is automated and can be embedded in today's electronic design automation (EDA) flow.

***Index Terms*—power estimation, digital IC, EDA, machine learning.**

I. Introduction

The increasing transistor density in digital ICs and the stringent performance, power, and area metrics (PPA), challenge today's design trade-offs. In particular, power (shorthand for average power in this paper) has become the predominant constraint in advanced technology nodes since operating frequencies and data rates have kept increasing. This resulted in a quick rising in power density trend and the end of Dennard scaling [15], highlighting the critical importance of accurate power estimation in modern ICs. Therefore, to address this power wall problem, there is a need for accurate early-stage power estimation to quickly evaluate different design choices.

Although power factor is a crucial metric, the current design complexity makes difficult the development of an accurate power model at an early design stage. In fact, this requires knowledge of switching activity and the parasitic capacitances in every transistor and physical wire in the design. Therefore, there is a need to employ power models that capture such essential low-level aspects while keeping the complexity of the model to a minimum.

In conventional power estimation approaches performed by EDA tools, the switching activity is retrieved by propagating the workload through the design's *netlist*. A netlist is an interconnection of standard cells generated after the *synthesis* step, which is the process that maps the circuit's functionality to the provided standard cells. Although propagating the workload results in the most accurate way to estimate the real activity, these steps need to be repeated for each design configuration, resulting in an excessively time-consuming approach. On the other hand, other's power estimation tools provide faster estimations by computing netlist's activity in different ways at the cost of accuracy, as elaborated in section III.

This work deals with this speed vs accuracy tradeoff by proposing a learning-based approach to model RTL components targeting stream-processing applications, where no control logic is involved. The goal is to leverage the switching activity information retrieved from the workload to perform accurate average power estimation. Such an approach intends to bridge the gap between the post-synthesis level and RTL, by providing the power estimation accuracy of the former and execution speed of the latter. The approach is divided into a design phase and a run-phase. The first involves the components' characterization, where power models are derived from netlist simulation for custom-tailored benchmarks. Once such characterization is performed for the employed components, the run-phase simulates the design functional description to retrieve the data activity of each pre-characterized component. This way, the accuracy gap between RTL and post-synthesis description is bridged. Such an approach can be coherently integrated with today's EDA design flow leveraging current standard high-level synthesis (HLS) tools.

The contributions are summarized as follows:

- Propose a new approach for dynamic power characterization that leverages a component's input and output activity to train a machine learning (ML) regression algorithm (section IV);
- A customized stimuli generation that provides training data representing a large spectrum of workloads (subsection A3 of section IV);
- A study on which supervised regression model between decision-tree, linear-based, and support vector algorithms

perform the best to infer power from average toggle activity information (section V);
- Perform a design space exploration (DSE) on a FIR filter as a use case scenario to quantify the approach accuracy to post-synthesis power numbers (section VI).

II. Related work

Given the complexity and importance of performing an accurate power estimation that can be later reflected on real measurements, extensive research has been focused on this topic targeting logic designs. Such works can be divided into probabilistic, statistical and simulation-based.

Probabilistic methods [11] are based on models that compute average nets' transition probabilities to estimate power. These privileges fast estimation at the cost of accuracy since no real net toggling activity is employed. Statistical methods run random input stimuli at the primary inputs and propagate them until the analysis converges to satisfactory results [6]. Although signal propagation is employed, resulting in more realistic nets' activity than the probabilistic approach, the injected random vectors do not represent true workloads, resulting in less accuracy. Simulation methods estimate power directly from true netlist activity. Although the higher estimation accuracy compared to the other two approaches, it is also the most time-consuming approach since the workloads need to be propagated through the whole netlist. Also, there is the need for low-level hardware implementation details, which consequently requires manual effort overhead.

In the literature, numerous works propose simulation-based inspired approaches that reduce the simulation time while trying to keep the power accuracy high [12]. The general solution implemented from such works is to track a subset of selected signals as activity sources instead of the whole netlist. These selected signals are called proxies, and work as activity monitors to reduce the RTL simulation runtime. The way such selection is made, and how power is computed from such signals activity defines the shortcomings that define the simulation speed versus accuracy trade-off. This section discusses the approaches present in literature that estimate power consumption given proxies activity. These approaches can be divided into analytical and regression-based.

A. Analytical approaches

Attempts to profile microarchitectural CPU blocks' came first with Wattch [3] and then with McPAT [10], which also adds an area and timing estimations. For network-on-chip (NoC) structures, Orion [18], and the more recent DSENT [16] are employed. Such tools use an analytical approach to model components' physical characteristics with equations to estimate power, resulting in a top-down approach. Coupled with performance simulators [1], these tools use performance counters of the modelled structures as proxies to return power numbers. Although the fast runtime of such high-level simulations, these models lead to inaccurate results and significant estimation errors due to their generic description and lack of nets activity information [19]. Moreover, such tools do not capture today's trend to design domain-specific accelerators to keep improving system-on-chip (SoC) performance. The effort needed in expanding such complex models, often implemented in C++ libraries, can be as cumbersome as handwriting RTL, nullifying the benefits.

B. Regression approach

This approach entails a general methodology to use proxies' activity to retrieve the total power consumption using regression algorithms. This can be applied at various granularities with different tradeoffs: the coarser the granularity of the proxy, the more accurate the model will be, at the cost of having a less versatile model; the finer the granularity, the more general the model will be, at the expense of less accuracy.
Early work of Benini et al. [2] characterize combinational and register macros by interpolating the toggling activity at the macros' interface with a linear model. This approach is not versatile since the model is fixed to its component. Moreover, a linear regression does not capture the highly non-linear behaviour of these processes. The same drawbacks from linear models can be found in [17], where the simulation is accelerated with an FPGA board.

Advanced ML regression algorithms are used in more recent work for both design time and on-chip power meters. In Lee et al. [9], gradient boosting and decision tree returned the best results. In Yukai et al. [13] L1 regularization technique with the Lasso penalty is used to prune the features that will be used as proxies to train the model. In APOLLO [20], the minimax concave penalty (MCP) is used as a feature pruning technique instead of Lasso. Then, it adds an L2 regularization with ridge regression with a weak penalty to relax its coefficients and train the weights with the proxies' activity. In Kumar et al. [8], a cycle-accurate model is needed to extract the features and compute power through pre-trained microarchitectural CPU blocks. Although these ML-based methodologies can accurately develop power models, the drawback they have in common is that the computed model is fixed to the design under estimation, requiring a new characterization for each design variation, and therefore preventing quick DSEs.

Deep learning (DL) approaches are exploited in [22] and [21]. The first trains a convolutional neural network (CNN) with RTL simulation. This results in cycle-accurate power estimation, but still, each new hardware needs further time-consuming training. The second implements a graph neural network (GNN) to improve transferability amongst designs. Since NNs involve a multi-level structure, compared to regression models, these require larger datasets and training computations. The performance gains may not justify the efforts, resulting in overkilling the problem.

This paper is motivated by a simulation-based approach. It contributes to the literature by proposing the component's average activity as a proxy from which power is computed, as discussed in section IV. Figure 1 summarizes the discussed works mapping them over a simulation speed versus power accuracy tradeoff.

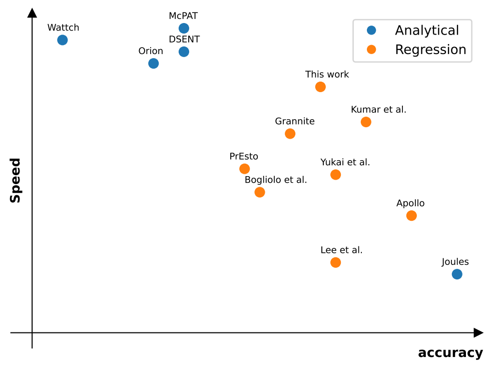

Fig. 1. Summary of related work organized in a speed vs accuracy trade-off space

III. Background

Power consumption is a fundamental metric used in the IC sector that accounts for the quality of a design, as well as delay and area. Power is generally divided into static and dynamic. The former accounts primarily for leakage current due to mainly technology-dependent parameters such as process, cell height, and threshold voltage V_{th}; the latter accounts for the parasitics switching charge at every node of the circuit. In the EDA flow, dynamic power is conventionally divided further into internal power, which models power consumption inside the standard cells and is provided by the vendors' *liberty* file, and switching power, which is retrieved from netlist nodes charge. In this paper, we focus on modelling dynamic power since static power is highly technology dependent and activity independent; hence it can be estimated by other means. Total dynamic power P is generally modelled as follows [14]:

$$P = \left(\sum_{i=1}^{N_{gates}} \alpha_i C_{L,i} \right) f_{ck} Vdd^2 = C_{eff} \times f_{ck} \times Vdd^2$$

Supply voltage Vdd and operating clock frequency f_{ck} are global/local variables that can be assumed as known and static. This is reasonable since the tool deals with design time estimation at early design stages (and not runtime). α represents the *activity factor* of the switching node and can be deterministically known only by simulating the workload. Details about the load and short-circuit capacitance of the cells (C_L) are stored in the technology files (*.lib*). C_{eff} represents the effective switched capacitance of the whole circuit.

Once the circuit is mapped to a technology file, such netlist can be simulated, and the activity of all nodes can be annotated in various formats depending on the level of information needed (*.tcf*, *.saif*, *.vcd*) for later power estimation. Then, propagating such activity into the netlist, power can be estimated with *post-synthesis* accuracy.

To better understand the importance of stimuli propagation, Table I compares the power estimation results for three components under different power estimation modes. *stim* power numbers are achieved from a full-gate-level simulation. The activity is then propagated through the netlist, and power is computed by matching the stimuli detected on the standard cells with the power models provided by the technology file. *VL* stands for vectorless, a power analysis method provided by EDA tools where a constant average switching activity factor is retrieved from the specified clock frequency and design constraints. As per Table I, when a stimulus is provided, the power result varies accordingly. This is not followed by the vectorless mode, which does not vary and stands between the two extremes. The resulting percentage errors between *stim* and *VL* are high, and there is therefore room for improvement. This shows how crucial it is to propagate activity information for accurate dynamic power estimation.

TABLE I
Average power difference with stimuli and vectorless mode

Component	activity	stim [μW]	VL [μW]	error
adder	low	3.74	9.59	-156.18%
	high	17.12		44.00%
multiplier	low	12.93	45.79	-254.19%
	high	78.95		42.00%
register	low	9.41	12.65	-34.41%
	high	16.06		21.26%

IV. High level framework

Today's hardware complexity prevents early stages estimation from being accurate since power is strongly dependent on physical details, and these get more precise only as the design gets closer to tape out. This work proposes a component-based power characterization for structural RTL designs. Structural designs are hierarchically defined by sub-components interconnections, without logic in between.

The approach is divided into a one-time component's characterization phase and a run-phase, as depicted in Figure 2. During the characterization phase, a set of component's features that work as proxies is parametrized and, for each feature's configuration, the component is fed with a set of stimuli. Each stimulus excites the component's input interface with a set of vectors. Each vector toggles with regard to the previous vector a fixed number of bits in random positions, simulating a large spectrum of workloads. The number of toggling bits is defined by the chosen average toggling activity. Then, both the component's features and the component's average toggle activity are related to the average power consumption. In this way, power is inferred from the component's interface activity, which substitutes the propagation step of the conventional power estimation flow. However, such an approach can be used only when it is possible to assume stream processing applications, where the power figure is determined by the toggling activity of the nets. The same approach cannot work for control paths, where what decides power consumption is the pattern of the signals rather than their activity.

The approach is presented in a framework with a high-level synthesis (HLS) design flow that translates the algorithmic hardware description to a structural one. This work's contributions consist of the design time engineering decisions (yellow box), the blue boxes during the run-phase, and the scripts that automatize each input/output step.

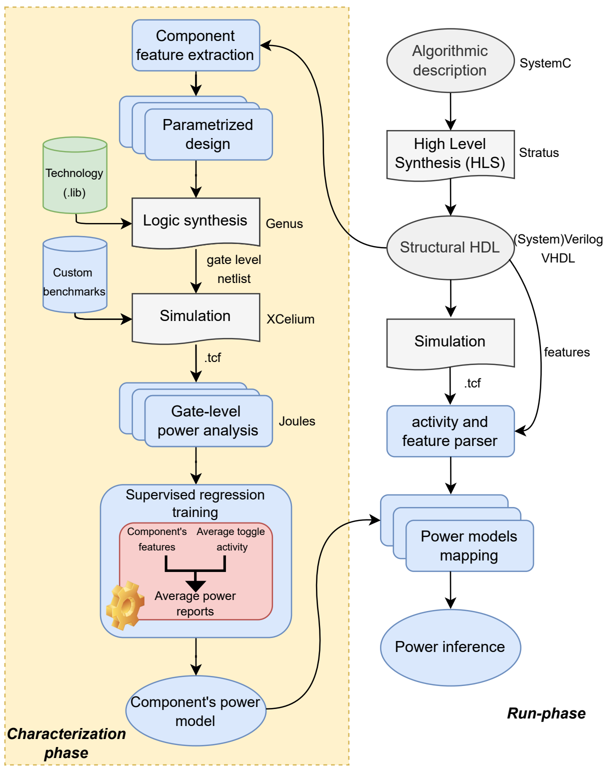

Fig. 2. Approach flow chart

A. Characterization-phase

The characterization step is needed to build a power model out of the chosen component's features that affect power. The goal is to have a large set of training data that relate such features choice to their power impact. Once trained, this model can be deployed and used in the run-phase for inference.

1) Feature extraction & parametrized design: Component's hardware features are extracted and parametrized in the RTL component description. This is done using domain-specific knowledge from common sources of power for the component under characterization. Besides data activity, examples of these are data bitwidth, technology process, pipeline stages, and any other parametrizable feature.

2) Synthesis & simulation: The parametrized designs are synthesized for each hardware configuration. In Section VI, Cadence Genus is used for the technology mapping using a 40nm node, returning the corresponding gate-level netlists. Also, power optimizations such as clock-gating and technology-dependent variables specified in the synthesis script can be parametrized during this step and used as features. In this way, one can perform DSE over such configurations as well. The generated netlists are simulated using Cadence XCelium using custom benchmarks, as explained in the next section.

3) Custom benchmarks: How data is generated is a fundamental step for learning based approaches. To relate activity to power consumption, each simulation runs a fixed input average toggling activity for the given feature configuration. The average toggling activity defines how many bits toggle each cycle with respect to the previous cycle. Also, the position of the toggling bit is random; this is referred to as random interleaving. In this work, average input activity α is defined as follows:

$$\alpha = \frac{\sum_{i=0}^{BW-1} bitline_i}{BW \times ck_{count}}$$

Where $bitline_i$ is the number of toggles at the $i-th$ bitline, BW is the number of bitlines and ck_{count} is the number of clock cycles, which can be computed as simulation time over clock period: $\frac{T_{sim}}{T_{ck}}$.

For example, assuming an input interface of 4 bitlines and fixing the average input activity to 25%, it results that 1 random bit out of 4 will toggle for each cycle. However, toggling the most significant bit (MSB) may result in different power consumption than toggling a bit in another position, say the least significant bit (LSB), since signal propagation inside the component depends on its kernel. Multiple simulations with random starting states are run for the same % activity to address this approach limitation. Also, for the 4 BW example, the smallest per-cycle % toggling activity is 25%. To address this other limitation, the available % steps are averaged for a long simulation time until the desired averaged input activity is achieved. For example, to achieve a 30% input toggling activity for the 4 BW case where the smallest step is 25%, the script will generate a set of stimuli that toggle each cycle, either 25% or 50%, so that the average after the simulation will be 30%. The same reasoning applies to any bitwidth and any average toggling activity. In this way, the benchmark generator script can sweep from 1% to 100% average toggle activity.

4) Power computation & model extraction: Given the synthesized gate-level netlists and their corresponding switching activity for each input average toggle rate, Cadence Jules is invoked to compute the average power. The power reports are parsed and the data is collected in a database alongside the component's features and the calculated average input/output activity. Such information is used to train the ML regressor model available in the scikit-learn Python package [4]. In section V the following regressors' performances are tuned, analyzed and compared: linear, lasso, extreme gradient boosting (XGB), random forest (RF), support vector regression (SVR), and decision tree (DT).

B. Run-phase

To incorporate the proposed method in the current EDA framework, Cadence Stratus can be used as a high-level synthesizer to convert behavioural into structural HDL for component feature extraction. HLS automates the conversion of untimed sequential software descriptions into optimized cycle-accurate hardware RTL models. Coupled with this work, the structural description can be parsed to derive the components feature. Then, after computing the average toggle activity of the provided workload, the power figure can be inferred. This way, the run-phase execution time is drastically reduced by removing the extended synthesis and simulation time, enabling faster design space explorations.

V. Machine learning algorithms

Machine learning techniques are implemented in combination with EDA tools at every level of the digital design flow [7]. This work deals with a supervised regression learning problem for power inference. This means that an estimator is tuned by interpolating a structured dataset that links the features to the estimated values. As described in Section IV, in this work the selected features are the ones extracted from a component, while the value to estimate is the correspondent impact on average power. The structured dataset is built during the characterization phase and it is splitted into a training and a testing one. The training dataset combines feature values and input/output toggle rates with the corresponding power consumption and is used to train the regression model. The testing dataset is used to evaluate the goodness of the trained model. The training data needs to be representative of the testing one to ensure an accurate prediction. Each model has *hyper-parameters* to be configured, which define the structure and penalty of the regressor. Hyperparameters are adjustable settings that control the learning process of a machine learning model. These parameters are set before training begins, determining the performance of the model on unseen data. To evaluate the different hyper-parameters, a grid search is performed for each model used to ensure the best fitness when constructing the estimator. Grid search consists in trying the different combinations of hyper-parameters that return the best *cross-validation* score. Cross-validation means performing the model's training and testing using different portions of the dataset each time. A cross-validation score close to 1 secures the model from data overfitting. Then, the mean average percentage error (MAPE) metric is computed on the testing set to measure the goodness of the generated model. The model that scores the best MAPE is employed to infer continuous output power values for the given input attributes.

In this section, the following six regression algorithms are analyzed [4]:

Linear: it uses a linear relationship between the independent variable and the dependent one(s) to fit the observed data. In this work, the weights are assigned to the features to minimize the residual squares sum penalty present fitting the linear relationship. Its simplicity results in a short training time but a low prediction score if the dataset presents non-linearities.

Lasso: it is also a linear regressor, with the difference of adding a penalty factor reducing the features' weights. It results in a sparse model when some features contribute less to the model than others. It is also called *regularization L1*, which is intended to reduce the model's sensitivity from single data observations, generally preventing overfitting of the data.

Support vector regression (SVR): it tries to find the best way to divide the margins of the hyperplane built on the features, so that the error between the predicted and actual values is minimized. A hyperplane is a subspace of one dimension less of the space it refers to. In this experiment, the input average toggle activity and the component's bitwidth are used as feature, resulting in a one-dimension hyperplane for the margin division.

Decision tree (DT): it is a model that iteratively divides the features space into smaller regions and learns simple decision rules inferred from the data features. Its effectiveness derives from the ability of the algorithm to iteratively divide the dataset into two parts such that the error function is minimized.

Random forest (RF): It aggregates many decision trees trained independently (*bagging*), where each tree is built from a subset of data entry and features chosen randomly. Then, the prediction is inferred by the combination of the predictions of each tree. When a prediction is performed from an aggregation of models like in the RF case, it is called an ensemble method.

Extreme gradient boosting (XGB): it is another ensemble method based on decision tree algorithm. It differentiates itself from RF by training and optimizing a set of decision trees built sequentially rather than independently. The ensemble estimator is built by splitting the nodes of the decision tree from the error (residual squares in this implementation) of the previous one. This technique is referred as *boosting*. Each new tree (so called *weak models*) predicts the output scaled by a learning rate factor. The ensemble of weak models combined produces the estimator.

Three components are characterized and evaluated with the approach defined in Section IV: a adder, multiplier and a 2:1 multiplexer. The features are the operand's bitwidth and the input average toggling activity. The power for each component is estimated using a 20% of the data as validation set, and the rest as training. Using a 9-folds cross-validation, the models are trained and validated on 9 different random portions of the dataset. Figure 3 plots for each component and for each regressor employed the measured $1-MAPE$. Hence, each bar of the chart lumps the average estimation error of the specified regressor against the validation part of the dataset. For each component, the linear and lasso methods both perform the poorest. This can be attributed to the fact that both use a linear relationship that poorly captures the non-linear dependency of power on the extracted features. Also, the regularization introduced by the lasso penalty do not suit an application with few features like the current one, resulting in the same MAPE as the linear one. In contrast, the decision-tree based algorithms perform all close to one. However, the two ensemble methods perform slightly better than the single decision tree model. This is due to the ensemble models combining the predictions of multiple individual models, rather than having only a single model like in the DT. Between the two ensemble models, XGB performs slightly better than the RF. This is attributed to the boosting method performing better in a dataset where there are few patterns in the data to be capture, since XGB trains weak models sequentially improving the error figure of the previous one. In contrast, the bagging method is more suitable for dataset where different patterns need to be captured, and hence taking advantage of individual models trained independently. In this use case where only two features are used, XGB performs better than RF. RF could benefit in case of many features whose variation impact the power figure differently. Although SVG uses a completely different approach than decision tree, the MAPE is comparable to the XGB for the adder and multiplexer. However it is not for the multiplier.

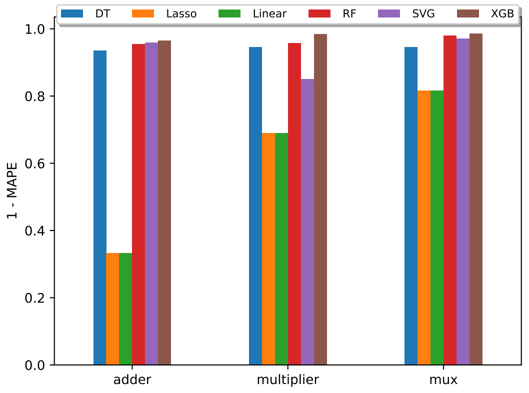

Fig. 3. MAPE regressors value for each component

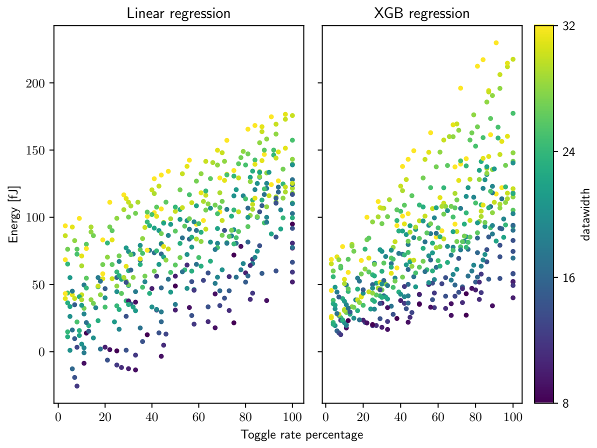

Fig. 4. Linear vs XGB predictor

We can conclude that the ensemble tree regressor with gradient boosting technique (XGB) returns the least MAPE error for the adder, multiplier and multipliexer, which is of 3.50%, 1.60%, and 1.39% respectively. This can also be evinced from Figure 4, where the scattered plot shows the inferred multiplexer energy required for different toggle rates and datawidth, using the linear and XGB model. This visually describes the ability of the tree-based estimator to approximate better the highly non-linear dependency between the energy required and the selected features. In contrast, the linear estimator poorly fits the trend, resulting in negative energy consumption for low toggle rates. Therefore, XGB qualifies to be employed for the experimental validation of Section VI.

VI. Validation

In this section, the proposed approach is validated against a structural description of a FIR filter as depicted in Figure 5.

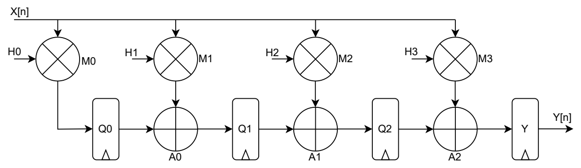

Fig. 5. Transposed FIR filter components' decomposition

A FIR filter in the digital domain is a block that performs convolution. In fact, the output signal $Y[n]$ is a weighted sum of the most recent samples of the input signal $X[i]$, multiplied by the coefficients $H_{1,\dots,4}$. Each symbol in figure 5 corresponds to an instance of a component, in particular: $\otimes$ stands for multiplier, $\oplus$ stands for adder, and the rectangle stands for register.

In this work, the synthesis environment employs a TSMC 40nm technology process with fixed cells height, standard threshold voltage V_{th}, PVT corner fixed to the typical case, and wireload modelling. The real values are irrelevant as long as they are consistent for the whole validation. *.lef* file (that includes layer information, via, and routing design rules) is used for better post-synthesis ground truth power numbers. Throughout the experiments, the synthesis' settings are kept the same, this is because the experiments are focused on extracting component feature rather than technology features. Moreover, for this validation the FIR filter sampling rate is fixed to 100MHz. Keeping the clock frequency fixed prevents the synthesis tool from employing gates with different driving strength, that would affect the dynamic consumption. This simplifies the current experiment by not tracking the gate selection during synthesis.

The FIR filter is made of the interconnection of a set of subcomponents. Each component is characterized with the flow depicted in Figure 2 to provide a power model that enables power-aware design space exploration over the parameterized features. For this validation, the parameterized feature extracted is the bitwidth. This means that each component is synthesized into a netlist for different bitwidths, and each netlist is simulated with the custom benchmarks described in Section IV. The multitude of activities produced (*.tcf*) is collected to compute the input and output average toggle activity of each design choice. Then, netlists and activities are fed into the Joules power engine, and the average power numbers are collected and associated in the database to the correspondent features. The XGB's hyper-parameters are now tuned on the provided database with the gridsearch approach. The hyper-parameter configuration that returns the best cross-validation score is the one used for the model's training. In this way, power models are dumped for the three components of the FIR filter, as shown in Figure 5.

For the run-phase, the FIR filter RTL structural design can be simulated with any kind of workload at the top level

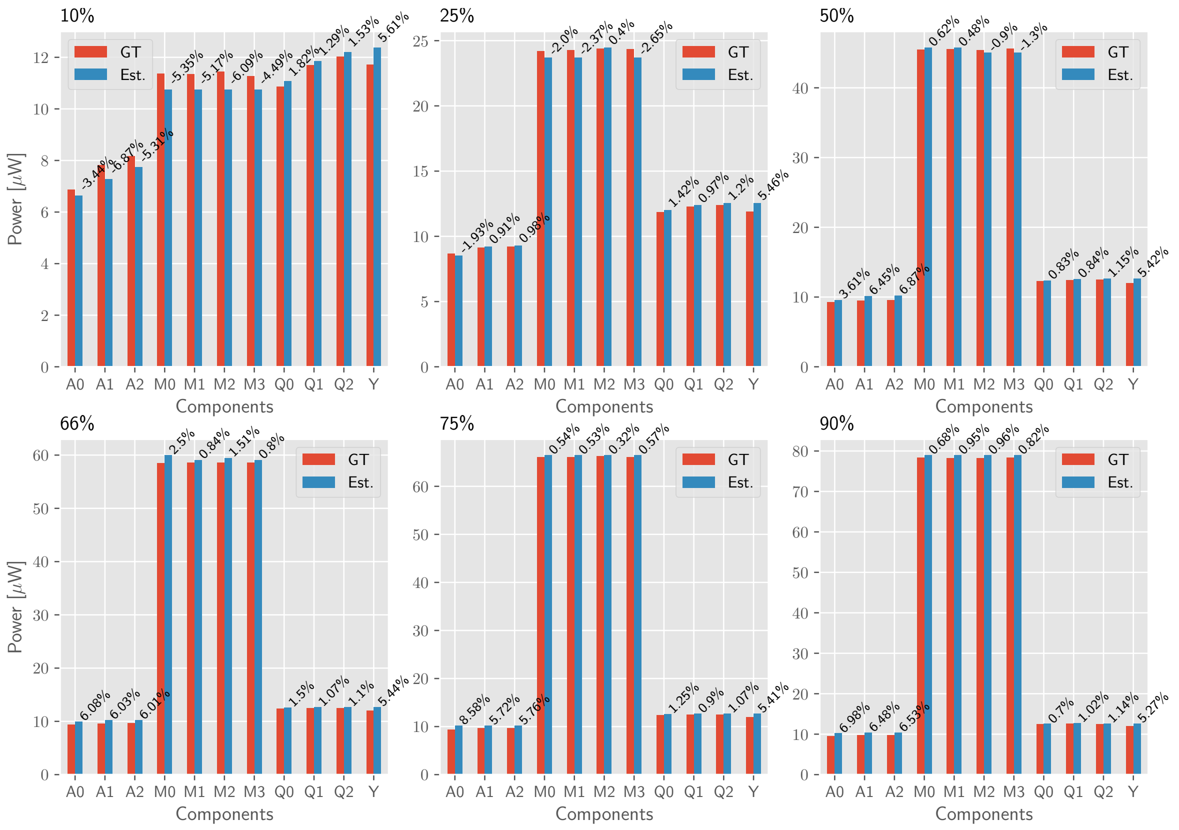

Fig. 6. Average power consumption of each component for different computed average toggle activities

input, indicated as $X[n]$ in Figure 5. In this way the stimulus propagates in the design and the activity at the input of one component depends on the previous component's output. For example, the input activity of *A0* depends on the output activity of *M0* and *M1*. After simulation, the toggle activity corresponding to the workload of each component is collected, and the input and output average toggle activity is computed. These 2 features, alongside the bitwidth information collected from the structural design, are used to infer the power consumption from the power models, as described by the run-phase of Figure 2. Note that these estimations are performed without the need to synthesize the whole FIR filter.

Figure 6 compares the inferred power (Est.) of each component against the post-synthesis power numbers referred as ground-truth (GT), for six different design choices of the FIR. The labels over each pair of bars indicate the percentage relative error of the estimation. The arithmetic sign of the relative error is kept to indicate underestimation when it is a minus and overestimation when it is a plus. Analyzing the charts, it can be seen that the inferred power numbers coherently follow the GT trend; also for the multiplier, that is the component that varies its power consumption the most, ranging from 11μW to almost 80μW for the 8 bit version. This has to be attributed to the component's features that are correctly parsed and computed for each simulation run. Also, the following trend can be derived: for adders and multipliers, the error starts negative for low activity (10%) and becomes positive for increasing activity, and for adders in particular reaches the highest error of 8.58%. These figures can be improved with better training data. Ideally, there we would like to have a bijective relationship between input activity, output activity and power. This is because for the same input activity we might have very different power consumption since this approach averages across all inputs. Therefore, to distinguish such cases, we should train the model for fixed input activity while sweeping the output activity. However, this is not feasible since the output activity depends on the component's kernel. Another source of error is that, when averaging over the component's bitlines, we lose information on whether a bitline toggles more than others; this choice has been made to favour quick training and inference over accuracy. Regarding registers, it can be seen that while *Q* registers have a consistent error offset of 1% throughout the whole activities, for the *Y* register this offset is fixed to 5.40%. This can be attributed to the fact that *Y* is an output register and has a different C_L to drive at its output. Similar charts can be found in [5] for the following BW choices: 12bit, 16bit, 24bit, and 32bit.

When dealing with DSE, the goal is to have an overview of the power consumption for the whole design for different design choices. This can be done by lumping the power inferred by each module, and comparing it to the ground truth

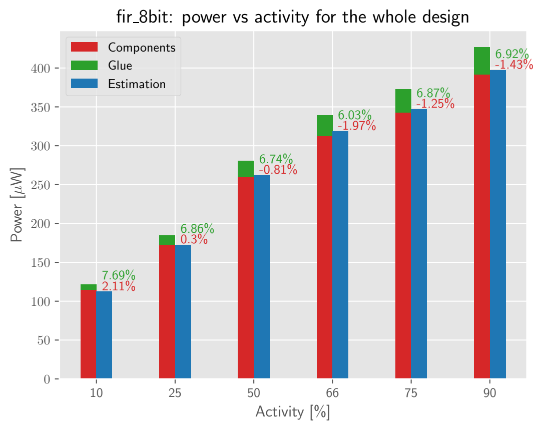

Fig. 7. Average power dissipation at top design level for 8 bit input

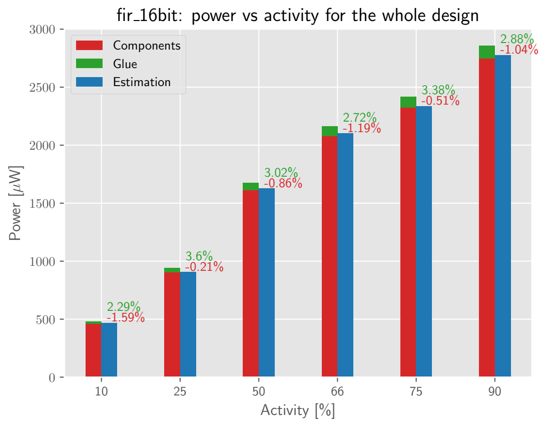

Fig. 8. Average power dissipation at top design level for 16 bit input

power figure computed by the power engine, as depicted in Figure 7 and Figure 8. These bar charts show how the power models perform under different input and output activities for 2 different FIR filter designs. The contribution highlighted in green in these two charts is the glue logic contribution. For glue logic, we refer to any type of logic inferred by the synthesis tool in-between components. The labels in red indicate the estimation error considering only the components' contribution. The labels in green indicate the estimation error including the glue logic. Although the glue logic contribution is ignored in this current version of the approach, it is taken into account when validating the results against the ground truth numbers. Taking the 8 bit case in Figure 7, although the absolute error considering the glue logic ranges from 6.03% to 7.69%, the relative error across the activities shows a small variability, highlighting the consistency of the proposed approach. The same goes for the 16 bit case in Figure 8. However, not considering the glue logic, the estimation error varies between 2.11% and -1.97% for the 8 bit case, while it varies between -1.59% and -0.21% for the 16 bit case. Compared to the results observed on the components in Figure 6, these estimation results in a better accuracy. This is mainly to attribute to the multiplier's power model accuracy. In fact, since the multiplier is the component that dissipates more power, when it comes to lumping together all the components' contribution into one bar, the total error results in an weighted average of the component's errors. Hence, the more one component contributes to the power dissipation compared to the others components, the more its error weights for the total design estimation. In this case, the low error of the multiplier provides a low design error.

Approach limitations

The results discussion highlights the main limitations of this approach. Such limitations are defined by the approach itself, and cannot be overcame unless the approach is radically changed. Here there are described the most important ones.

1) Glue logic: As seen in Figure 7 and Figure 8, glue logic is responsible for a clear discrepancy between the design estimated power and the ground truth one. This is because while the former is just the sum of the inferred components' power, the latter is the one computed from synthesizing the whole design. It is tricky to establish during the component's characterization time, the amount of glue logic inferred by the synthesis tool in between components. The same component may be utilized in different designs, hence it may be surrounded by different glue logic. For this reason, and for the early-stage nature of this approach, this problem is not addressed.

2) Output activity: Both input and output activities are an indication of a component power consumption, and that is why both figures take part in the feature selection. However, while input ports can be arbitrary swept for any average input toggle activity, the output one is in function of the activity and the component's kernel. Hence, we do not have every combination of input and output activity. To quantify the power impact of the output activity, there is a need for a systematic way to produce sufficient output average toggle activities while keeping the input one fixed. Only in this way the training data can be considered to be exhaustive.

3) Control logic: This work addresses the problem of power estimation in stream processing applications in digital integrated circuits. Such an approach works better when the assumption that each toggling bitline impacts power uniformly is reflected in the real case. For control logic, it is essential to use cycle-accurate activity information, where the power impact of each triggered action can be annotated back to the corresponding bitline(s). This approach is feasible but it would be radically different from the current one, opening a new tradeoff study.

VII. Future work

The validation highlighted pitfalls that can be further addressed to improve the estimation accuracy of this approach. However, as discussed in Section VI, there are also some feasibility limitations that are inherently defined by the approach. This section discusses the first steps that can be taken to improve the approach accuracy.

1) weighted contribution: As discussed in section VI, the ideal goal of this approach is to find a bijective relationship between the component's interface activity and its power consumption, avoiding the signal propagation step. However, the same average toggle activities can lead to different power consumption, since the position of toggling bits is different for each simulation, and hence different parts of the component are stimulated. The error that arises from this is currently mitigated by running long simulations where the impact of the bit position is averaged out. However, this is not an ideal solution since in the case of workloads during run-phase where certain bit positions are stimulated more than others, this peculiarity is not captured by the model. To approach this, a study on the power impact of each bit position can be performed. Then, a weighted sum of the activity can be implemented by the parser, returning an *effective activity* instead of the current average toggle activity. However, this approach should be preceded by a feasibility study, since the effort from this in-depth study may not be justified by the accuracy gains.

2) glitching: Glitch toggles refer to the nets' activity that is not captured by the clock edge. In other words, a signal level is sampled by the clock edge after t_{setup} and before t_{hold}. The activity that happens outside these two time windows is referred to as glitching activity. In this work, we ignored the glitching activity. However, it could be taken into account in the activity files dumped after simulation.

3) Hierarchies: This work explores only one level of hierarchy, where the top module (FIR filter) power module is a composition of only one level of submodules. However, the iterative nature of this approach leaves room for further studies in this direction by applying iteratively the same approach. For example, the FIR filter power model can be a submodule of a larger design.

VIII. Conclusions

This work proposes a learning-based approach that leverages the RTL component's input and output activity information, as well as the extracted component's feature(s) to estimate power consumption with post-synthesis accuracy in stream processing applications. This bridges the gap between RTL and gate-level in the digital design flow, using the average toggle rate of a workload as a way to quickly estimate the dynamic power consumption. This prevents the need for the time consuming synthesis step and signal propagation inside highly interconnected hardware, enabling power estimation at RTL with post-synthesis power accuracy for early DSE. However, by averaging the component's activity over the entire simulation time, the information on per-cycle toggle events is lost, defining a source of accuracy loss for this power estimation approach.

The experimental results in section VI use a FIR design to compare the power models built with this approach against ground truth post-synthesis numbers. It shows that the models respond to workloads the same way the original design does. The results illustrate that a component level, the multiplier and the register model accurately follow the real numbers, showing accuracy that vary from 5% to 0.5% depending on the workload's activity. For the adder, the results are less accurate, ranging from -6.87% to 8.58% . However, at the design level, the accuracy of the results when not considering the glue logic ranges between -1.97% and 2.11% for the 8 bit case and between -1.59% and -0.51% for the 16 bit case. This high accuracy is achieved by the estimation errors that are averaged across the components' power models. Since the multiplier is the component that dissipates power the most, and its model returns high accuracy estimations, the estimation error at design level results to be small. When considering the glue logic, the design errors increase to a maximum of 7.69% for the 8 bit case and 3.38% for the 16 bit one. However, such an error adds up as an offset and does not alter the results' consistency across the activity range.

Scripts, data, as well as plots for the FIR DSE (8bit, 12bit, 16bit, 24bit and 32bit from 1% to 100% average toggling activity) are available on GitHub [5].

References

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.

[2] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. Regression-based rtl power modeling. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(3):337–372, 2000.

[3] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News*, 28(2):83–94, 2000.

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[5] D.M. This work repo, 2022. https://github.com/damonti/Characterization,.

[6] Yaseer A Durrani and Teresa Riesgo. High level statistical power estimation.

[7] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, and Yu Wang. Machine learning for electronic design automation: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), jun 2021.

[8] Ajay Krishna Ananda Kumar, Sami Alsalamin, Hussam Amrouch, and Andreas Gerstlauer. Machine learning-based microarchitecture- level power modeling of cpus. *IEEE Transactions on Computers*, pages 1–14, 2022.

[9] Dongwook Lee, Lizy K John, and Andreas Gerstlauer. Dynamic power and performance back-annotation for fast and accurate functional hardware simulation. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1126–1131. IEEE, 2015.

[10] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(1):1–29, 2013.

[11] R. Marculescu, D. Marculescu, and M. Pedram. Probabilistic modeling of dependencies during switching activity analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(2):73–83, 1998.

[12] Yehya Nasser, Jordane Lorandel, Jean-Christophe Prévotet, and Maryline Hélard. Rtl to transistor level power modeling and estimation techniques for fpga and asic: A survey. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):479–493, 2021.

[13] Daniele Jahier Pagliari, Valentino Peluso, Yukai Chen, Andrea Calimera, Enrico Macii, and Massimo Poncino. All-digital embedded meters for on-line power estimation. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 737–742, 2018.

[14] Jan Rabaey. *Low Power Design Essentials*. Springer, 2009.

[15] Efraim Rotem, Ran Ginosar, Avi Mendelson, and Uri C Weiser. Power and thermal constraints of modern system-on-a-chip computer. *Microelectronics Journal*, 46(12):1225–1229, 2015.

[16] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 201–210. IEEE, 2012.

[17] Dam Sunwoo, Gene Y Wu, Nikhil A Patil, and Derek Chiou. Presto: An fpga-accelerated power estimation methodology for complex systems. In *2010 International Conference on Field Programmable Logic and Applications*, pages 310–317. IEEE, 2010.

[18] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: A power-performance simulator for interconnection networks. In *35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002.(MICRO-35). Proceedings.*, pages 294–305. IEEE, 2002.

[19] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. Quantifying sources of error in mcpat and potential impacts on architectural studies. In *2015 IEEE 21st International symposium on high performance computer architecture (HPCA)*, pages 577–589. IEEE, 2015.

[20] Zhiyao Xie, Xiaoqing Xu, Matt Walker, Joshua Knebel, Kumaraguru Palaniswamy, Nicolas Hebert, Jiang Hu, Huanrui Yang, Yiran Chen, and Shidhartha Das. 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*, pages 1–14, 2021.

[21] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. Grannite: Graph neural network inference for transferable power estimation. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[22] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. Primal: Power inference using machine learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.