Technical Report

Data-Dependent Cycle-Accurate Power Modeling of RTL IPs Using Machine Learning

Malek Srour
Andreas Gerstlauer

UT-CERC-18-01

May 2, 2018

Computer Engineering Research Center
Department of Electrical & Computer Engineering
The University of Texas at Austin

2501 Speedway, Stop C8800
Austin, Texas 78712-1234

Telephone: 512-471-8000
Fax: 512-471-8967

http://www.cerc.utexas.edu

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Data-Dependent Cycle-Accurate Power Modeling of RTL IPs Using Machine Learning

Malek Srour, Andreas Gerstlauer

Electrical and Computer Engineering,
The University of Texas at Austin

Abstract

In a chip design project, early design planning has a strong impact on the schedule and the cost of design. Power estimation is part of early design planning, and it greatly affects design decisions. Power modeling performed at a high level of abstraction is fast but inaccurate due to lack of circuit switching activity information. By contrast, power modeling performed at a low level of abstraction is more accurate as the synthesized circuit synthesis is known, but this simulation is typically slow. This report explores a power modeling approach performed at register transfer level (RTL). It exploits machine learning models in order to have a fast yet relatively accurate cycle-by-cycle power estimation. The approach is data-dependent, where cycle-specific models are trained based on the switching activity of signals obtained from RTL simulation and cycle-by-cycle power values obtained from a reference gate-level simulation of an existing RTL design. Therefore, if any changes are applied to the RTL design, re-training of models is required. The approach aims at obtaining fast yet accurate power predictions for new invocations of a given trained model using signal activity information collected during simulation of the unmodified RTL. At a low level, the complete visibility of signals in a design unintuitively might cause overtraining the model leading to inaccurate estimation.

The suggested model employs automatic feature selection in each cycle. Based on the

invocations used to train the cycle-by-cycle models, only signals that may switch during a given cycle will be selected as the features for their respective cycle-specific model. The method was tested on an 8-by-8 DCT design and the power estimates were within 6.5% of those from a commercial power analysis tool. This report also simulates and compares the approach of cycle-specific models to the approach of a single global model for all cycles and show that the cycle-specific approach is twice as accurate.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Power consumption in intellectual property (IP) designs is a crucial limiting factor for both performance and chip density. Even though transistor scaling intensified the significance of static power consumption, dynamic power consumption still accounts for a significant portion of the power consumed in a chip. Moreover, with compact and portable designs, cooling mechanisms are quite difficult to embed, which leads to another problem posed by power consumption. At the same time, design complexity is growing continuously; directly translating into growth of design cost which makes design time of an essence. Given all these factors, early design planning (EDP) is becoming more and more important, but in an ideal design environment it is required to be completed rapidly. Early power estimation is a major part of EDP and could be done at different levels of abstraction.

Depending on several factors, power modeling is performed at a functional model level, register-transfer level (RTL) or gate level. Mainly, there is a trade-off between speed of simulation and accuracy of estimation among these approaches. To obtain fast power estimates at a high level of abstraction, like a C/C++ functional model, several approaches have been proposed [1], [4], [8]. These approaches rely on coarse-grain state-based methods yielding inaccurate yet fast power estimation. To obtain more accurate estimation, fine-grain slow simulations that capture the switching activity are performed either at the intermediate representation level [2], [9], [10], [11] or the RTL/gate level [3], [12], [13]. Research is always being conducted on how to improve this trade-off by employing a fast yet accurate power modeling approaches for early analysis of designs. This report investigates the effectiveness of a learning-based cycle-by-cycle power modeling approach performed at the RTL level. Dynamic power consumption is data-

dependent as it is directly related to the switching activity of signals in a design. For that reason, switching activity of signals is important to estimate the power. This approach utilizes switching information available at the RTL level, for accurate power estimation. Cycle-specific models of a design are trained based on switching activity coming from RTL simulation and cycle-by-cycle power consumption values obtained from gate-level simulation. This makes the models specific to the RTL design on which training was applied and thus the approach is used to improve speed of estimation by performing accurate data-dependent power predictions. During prediction, for a given set of new inputs, RTL simulation is performed, and the switching activity of the signals is used by the trained models to predict the power.

The main challenge in any learning-based approach is feature selection. In previous works as in [3] and [12], feature selection is either done manually or by trial-and-error. In the new approach, relevant features are identified automatically during training. The effect of the size of the training set on the accuracy of the estimations is investigated for a given benchmark. Moreover, it is shown that having cycle-specific power models is better in terms of accuracy than a global model used across cycles.

The remaining of the report is structured as follows: Chapter 2 includes an overview of previous work, Chapter 3 presents the design methodology, Chapter 4 shows experimental results, Chapter 5 lists a summary of possible future work and Chapter 6 concludes the report with a recap.

Chapter 2: Previous Work

Power modeling at a high level of abstraction has been extensively researched as it is the fastest approach to arrive at power estimates but is coarse-grain. As design complexity grows, it is not always feasible to simulate the entire design at the RTL level to obtain power properties. Typically, at high levels of abstraction, a power state machine approach is used for estimation, yielding inaccurate results. Thus, research mainly focus on extending the power state machine model to account for data dependency while keeping the model efficient. Lorenz et al. [1], is one example where they used Hamming distances of inputs and internal pipeline stage registers - in case of a pipelined design - to take into account the switching activity and consequently improve previous power state machine models. These approaches as well as others that are performed at a high level of abstraction attempt to lessen the limitation of internal information about switching activity being not entirely visible, by integrating novel methods while trying to keep the simulation rapid.

One approach is to divide the design into separate components and simulating them independently is not enough as the interaction among these components is necessary for the power analysis [2]. Reference [2] presented an approach for estimating power consumption of hardware and software components of a multi-processor system-on-chip through back-annotation of the power properties from the low-level to the functional model source code. Indirect back-annotation was performed by combining three separate approaches that dealt with software, custom hardware and black-box IP.

At lower levels of abstraction, speed of simulation becomes the bottleneck. For that, several researches have proposed new methodologies that speed-up the simulation while maintaining the accuracy within an error range. In library-based approaches [9],

[10], [11], estimation is based on pre-characterized component models. The limitation of such approach is that it does not capture the glue logic that represents interaction among the components and can have a great effect on power consumption. Learning-based approaches were proposed in [3], [12] and [13]. These approaches derive a regression-based model of the design or macro-block, which is not necessarily accurate for complex architectures. FPGA-accelerated simulation [3], shows to increase speed in orders of magnitude while sacrificing accuracy slightly.

Lee et al. presented in papers [5], [6], [7], a complete learning-based power modeling framework for system-level C/C++ hardware IPs, used for power estimation at three different levels of granularity: invocation-, basic block-, and cycle- level. At a coarse granularity, existing high-level synthesis flows were extended with automated back-annotation that allows for data-dependent cycle-accurate power estimations. Machine learning techniques were leveraged to increase accuracy of estimation and reduce complexities. Reference [6] targets black-box IPs where RTL models are not available and thus fine-grain power estimation is not possible. Although the work is presented at different levels of granularity, they all share the property of being data-dependent and learning-based which enables fast fine-grain estimations. At the same time, the work targets power modeling at a high C/C++ level of abstraction. The approach presented in this report adopts the latter approach by leveraging state-of-art machine learning techniques to generate data-dependent cycle-by-cycle trained power models for a given RTL design for faster power predictions for new invocations.

Chapter 3: Design Methodology

This chapter presents the design flow for the proposed power model approach shown in Figure 1. The design flow is separated into two distinctive parts: the training flow generates the cycle-by-cycle models for a given design and the prediction flow then uses the generated models to predict power estimates on new invocations.

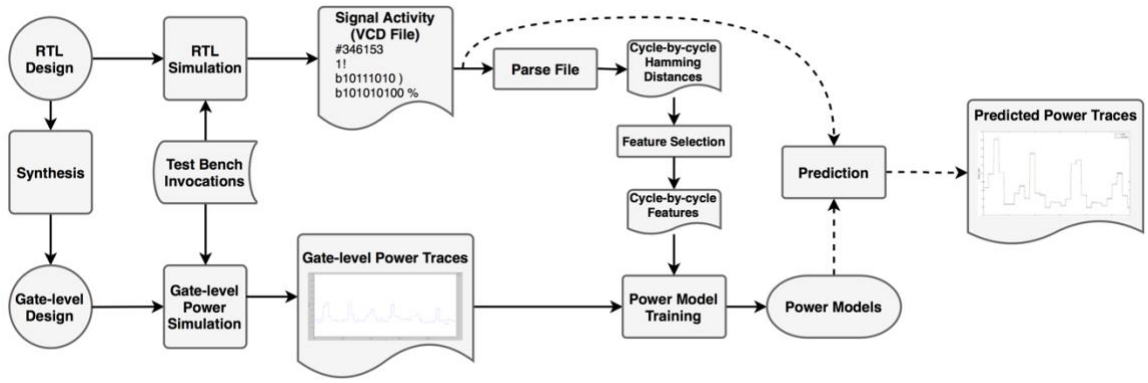

Figure 1: Proposed Power Modeling Flow.

Dynamic power is dependent on the switching activity of signals in a given design, so Hamming distances of signals were utilized to train the machine learning models. As this approach is data-dependent, the training relies on a collection of invocations with different data inputs. During the training flow of a certain design, a value change dump (VCD) file is generated for a given number of invocations and then used to efficiently compute Hamming distances only for signals that changed in a given cycle. The VCD file is parsed and Hamming distances of signals are computed on a cycle basis. The same cycle in different invocations of the design, might have different signals that are switching; thus, for each cycle, the union of signals that had switching activity in this cycle across all invocations are considered as the potential signals that contribute to

dynamic power for that cycle. In the training step, a set of the simulated invocations is chosen to train the cycle-by-cycle model and the remaining invocations are used for cross-validation of the models.

After well-trained cycle-by-cycle models have been generated for the given benchmark, the following prediction flow is followed to get cycle-accurate power estimates. A VCD file generated from simulation of RTL with the input data given by the invocations to be predicted is fed into the prediction framework, which outputs average cycle-by-cycle power estimates. Internally, the framework takes in the VCD file, extracts the Hamming distances of relevant signals for each cycle and these values are used by the trained models to generate the cycle-accurate power estimates.

3.1 HAMMING DISTANCE COMPUTATIONS

This section goes into further details of how the required pre-processing of data, to be used for both training the power models and prediction, is implemented. An RTL simulation of a design entails applying input vectors to the input ports and obtaining the output vectors from the output ports. Simply put, a design usually consists of a set of registers to store values and combinational logic. Consider that a testbench used to simulate the RTL consists of N invocations that drive the inputs of the design where each invocation completes in C clock cycles. In the testbench of the RTL simulation, the output signals of all the registers available in the design are dumped into a VCD file. The VCD file includes all switching activity of signals timestamped relative to the timescale provided in the testbench. The switching activity reported in the file is provided as new bit-vector values of only the signals that changed during a certain timestamp. A sample VCD file format is shown in Figure 2.

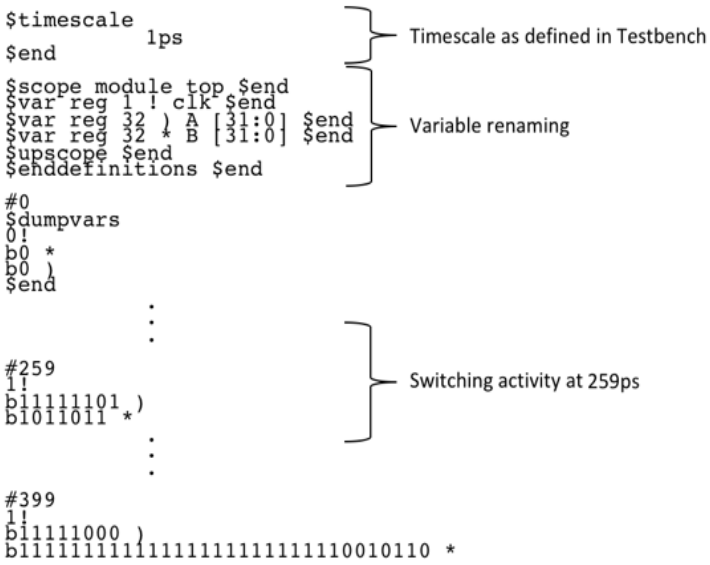

Figure 2: Sample VCD file annotated.

The VCD file is then parsed. To efficiently find Hamming distances of signals from the VCD file, the skeleton of an open-source VCD parser Python script, was used and modified in order to incorporate computing the Hamming distances of signals and writing them into an output file while parsing the VCD file.

To start with, a few terminologies are defined and will be used throughout this chapter. Let *R* represent the set of all $\boldsymbol{r_i}$ in the RTL, where $\boldsymbol{r_i}$ is the output bit-vector of register *i* in the design, as in Figure 3. Let $\boldsymbol{r_{i,c}}$ be the value of bit-vector $\boldsymbol{r_i}$ in cycle *c*. The Hamming distance of $\boldsymbol{r_i}$ in cycle *c*, $HD(\boldsymbol{r_{i,c}})$, is computed by finding the number of bit differences between $\boldsymbol{r_{i,c-1}}$ and $\boldsymbol{r_{i,c}}$. In Figure 2, assuming that registers A and B did not change between timestamps 259ps and 399ps, the Hamming distance for timestamp 399ps of registers A and B is 2 and 29, respectively. The pseudo-code of the parse, compute and store script is provided in Figure 4.

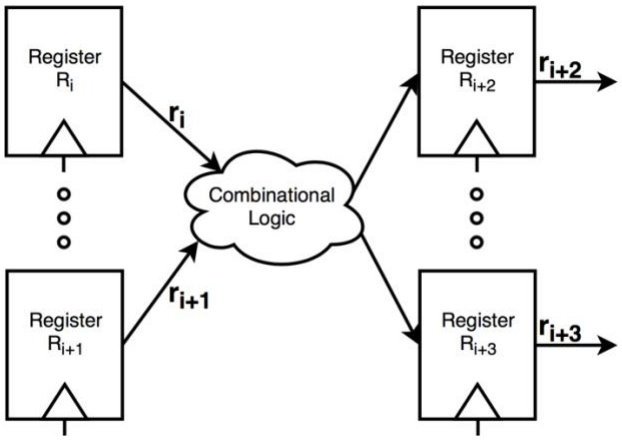

Figure 3: Illustration of an RTL design showing respective r_i.

```
for each line in VCD file
    case(line[0])
        ('b', 'B', 'r', 'R'):
            new_value <- bits up till the space
            signal <- symbol at end of line
            hamming_distance <- bitwise_difference(new_value, old_value)
            old_value <- new_value
            write (signal, hamming_distance, time_stamp) to file

         ('0', '1', 'x', 'z'):
            new_value <- bits
            signal <- signal symbol
            hamming_distance <- bitwise_difference(new_value, old_value)
            old_value <- new_value
            write (signal, hamming_distance, time_stamp) to file

         ('#'):
            time_stamp <- value
    end
end
```

Figure 4: Pseudo-code of part of the VCD parser.

Now that the Hamming distances of signals have been obtained with their corresponding timestamps, the output file containing all this information is given to a MATLAB script that applies some processing steps for feature selection before training the cycle-specific models, as presented in the next subsection.

3.2 PROCESSING DATA FOR FEATURE SELECTION AND MODEL TRAINING

The file generated during the pre-processing step is imported into MATLAB to further process the data for feature selection. The data is split into blocks of *C* cycles where each block corresponds to an invocation *n* from the testbench since as mentioned earlier, each invocation *n* of the *N* total invocations equally consists of *C* cycles. This means that it is necessary to train *C* unique power models P_c. Each of these models would have different features based on which signals had Hamming distances during cycle *c*. As an RTL design behavior is generally data-dependent, it is possible to have different set of signals with non-zero Hamming distances for the same cycle across different invocations. Therefore, to obtain the set of features for cycle *c*, the following was applied. Let V_c be the set of signals corresponding to the power model P_c. Then,

$$\boldsymbol{r_i} \in V_c \iff \exists n \; s.t. \; HD(\boldsymbol{r_{i,c}}) \neq 0$$

Now that the features have been selected for each cycle-specific model P_c, these models can be trained by learning a function F_c for each cycle *c*. The values of the features that are used as inputs to learn the function F_c, are the Hamming distances of the elements of the set V_c, i.e. $HD(\boldsymbol{r_{i,c}})$ where $\boldsymbol{r_i} \in V_c$. If for a given invocation *n* and cycle *c*, there exists an $\boldsymbol{r_i} \in V_c$ with no Hamming distance for that specific invocation *n*, then the value of that feature is considered to be zero. The general formulation of the obtained power models can be written as follows:

$$P_c \equiv F_c\left(HD(\boldsymbol{r_{i,c}}) \; \forall \boldsymbol{r_i} \in V_c\right)$$

In MATLAB, $V_c \; \forall c$ across all *N* invocations is obtained and an $n \times \|V_c\| + 1$ table is formed for each cycle *c*. These tables contain *n* rows corresponding to the different invocations and the columns contain the $HD(\boldsymbol{r_{i,c}})$ where $\boldsymbol{r_i} \in V_c$; with the last column containing power values obtained from gate-level simulation. Each of the tables are then used independently to train the model for the corresponding cycle by fitting the data

using a regression-based machine learning model; with predictor values being the first $\|V_c\|$ columns and the response values being the last column. At this point, *C* cycle-specific trained regression models, P_c, would have been synthesized based on a training set of size *N* invocations. As evident by the power modeling synthesis flow, the framework is fairly straight forward and could be easily modified to leverage any machine learning model.

3.3 POWER PREDICTION USING TRAINED MODELS

Although the prediction scheme is very similar to the training scheme, they are treated as separate entities as the actual performance metrics, i.e. speed and accuracy, are measured based on the prediction scheme performance. When the training phase is completed, and the cycle-by-cycle power models are available, they can be used to predict cycle-accurate power consumption for the same RTL design but new invocations. This approach aims at replacing the need for commercial power simulation tools that perform power estimation at the RTL or the gate-level when no changes have been applied to the RTL and a number of predictions needs to be done. To start with, the RTL design needs to be simulated with a testbench that provides the input data for which power prediction is required. This simulation produces the VCD file which is then passed through the VCD parser to compute the Hamming distances. Now that the feature selection process has been completed during the training phase based on the training invocations, only the relevant signals for each cycle are selected and other signals are disregarded. In other words, let T_c be the set of features selected during cycle *c*, then:

$$\boldsymbol{r_i} \in T_c \iff \boldsymbol{r_i} \in V_c$$

The Hamming distances of these features are the inputs to the trained model P_c where:

$$HD(\boldsymbol{r_{i,c}}) = \begin{cases} HD(\boldsymbol{r_{i,c}}), & \boldsymbol{r_i} \in T_c \\ 0, & otherwise \end{cases}$$

Finally, cycle-accurate predictions of power $P(c)$ in cycle c for each invocation are obtained from the cycle-specific trained models P_c as:

$$P(c) = P_c\left(\left(HD(\boldsymbol{r_{i,c}})\ \forall \boldsymbol{r_i} \in V_c\right)\right)$$

The next chapter presents results attained from applying this approach on a given benchmark.

Chapter 4: Results

This chapter of the report, presents the results of applying the approach on a given RTL design. The design is an 8-by-8 Discrete Cosine Transform (DCT) composed of a state machine with 7 states and 14 registers of different bit lengths. The design was simulated in Mentor Graphics ModelSim, to generate the VCD file, with an input image of total size 256x256 pixels and thus 1024 8x8 blocks. This means that the total number of independent invocations was 1024, where each invocation took 1155 clock cycles to complete. The 1024 invocations were divided into a training set and a test set. The VCD parser was implemented as a Python script that reads in the VCD file and outputs a file that contains Hamming distances of signals, the corresponding cycle number and signal name. MATLAB was then used to process the data, by reading the file outputted by Python, and selecting appropriate features for each cycle. Decision Tree and Gaussian Support Vector Machine (SVM) regressors were the two different machine learning regression-models used to model the cycle-by-cycle power behavior of the circuit. Figures 5 and 6 show two samples of a trained regression decision tree model for two different cycles, where based on the given training data, signals 1, 6, 7 and signals 1, 2, 7 are the selected features, respectively.

The RTL was synthesized into gate-level netlist using Synopsys Design Compiler. The reference power values were obtained by simulating the gate-level netlist in Synopsys PrimeTime PX. To assess accuracy, the average relative error of predictions for the test set across all the cycle-by-cycle models was computed as follows:

$$\frac{\sum_{j=1}^{C}\left(\frac{\sum_{i=1}^{N}\left(\frac{|Expected\ Power - Predicted\ Power|}{Expected\ Power}\right)}{N}\right)}{C}$$

Where *N* is the number of invocations and *C* is the number of cycles per invocation. The inner sum of the numerator term is the sum of the absolute relative errors of predictions across all test invocations for a given cycle-specific model. Then the average relative error of predictions is computed by taking the mean of the average absolute relative errors of predictions for each cycle-specific model.

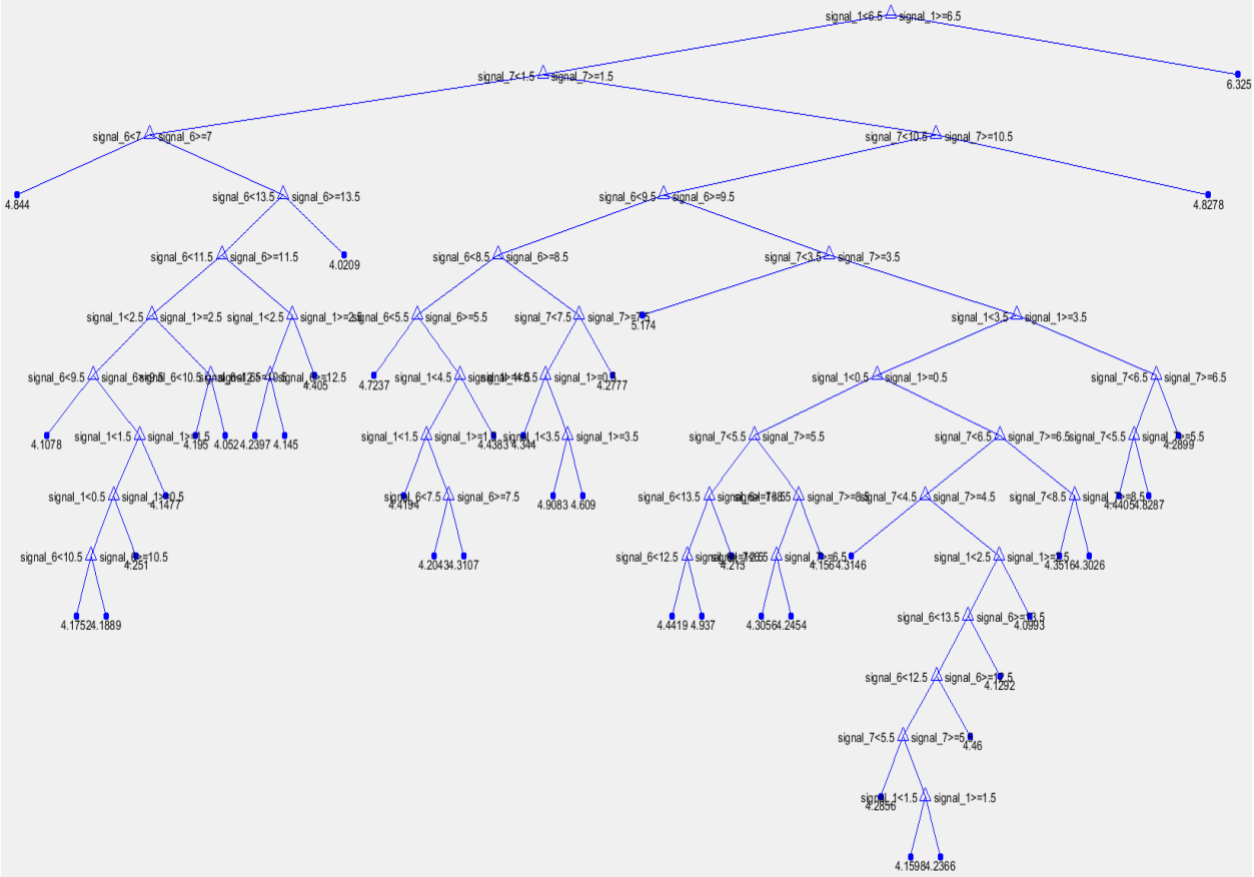

Figure 5: A trained decision tree regression model with three features (signal_1, signal_6 and signal_7).

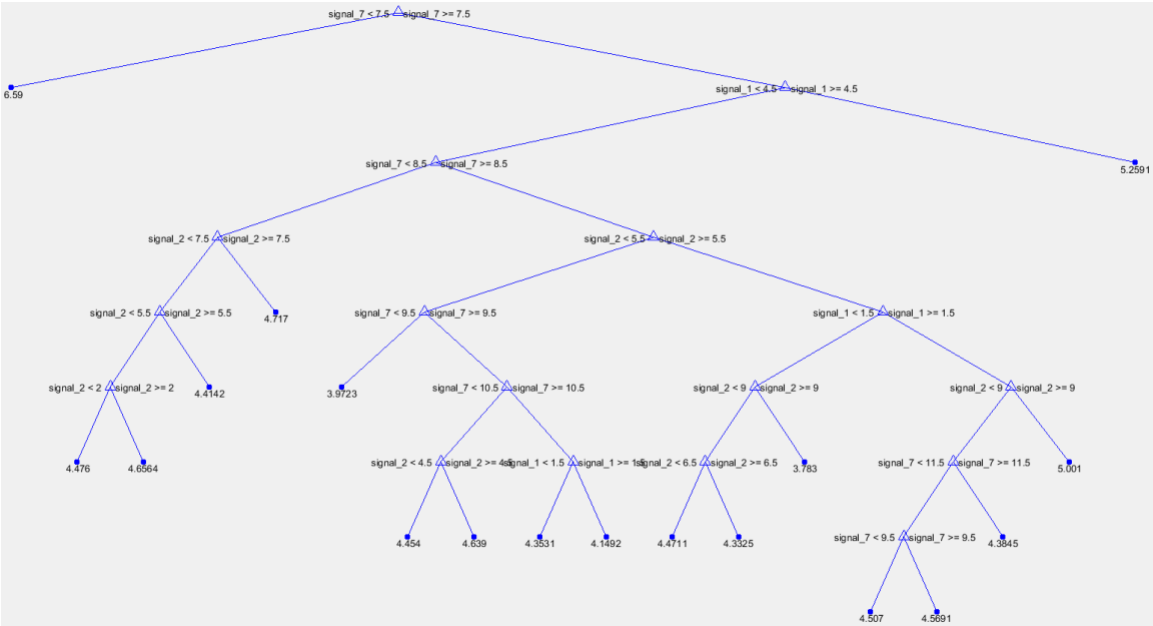

Figure 6: A trained decision tree regression model with three features (signal_1, signal_2 and signal_7).

4.1 Power Prediction Compared To Expected Power Consumption

Figures 7 and 8 show the cycle-by-cycle average predicted power and the cycle-by-cycle average measured power for a specific window of cycles. The predicted power values were generated based on a cycle-by-cycle gaussian SVM with training set size being around 25% of the total number of invocations used for training and testing. The choice of the window of cycles shown in the figures was in a way to show a case where prediction was perfectly aligned with the measured values (Figure 7) and a case where there was an error of prediction (Figure 8).

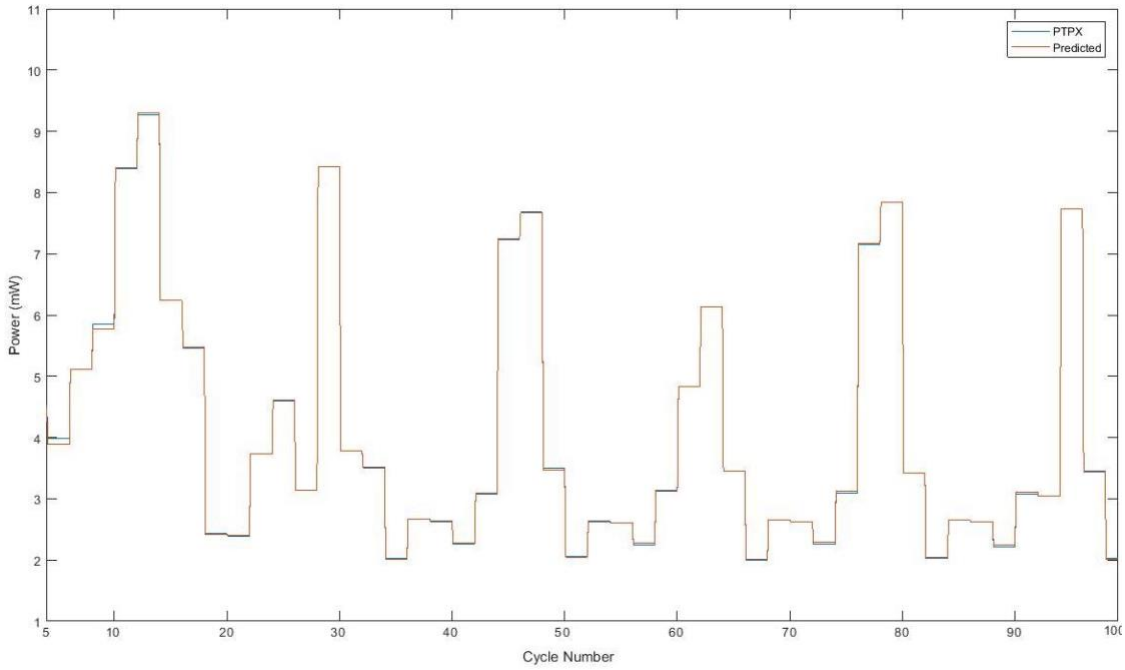

Figure 7: Window of cycles that show accurate predictions.

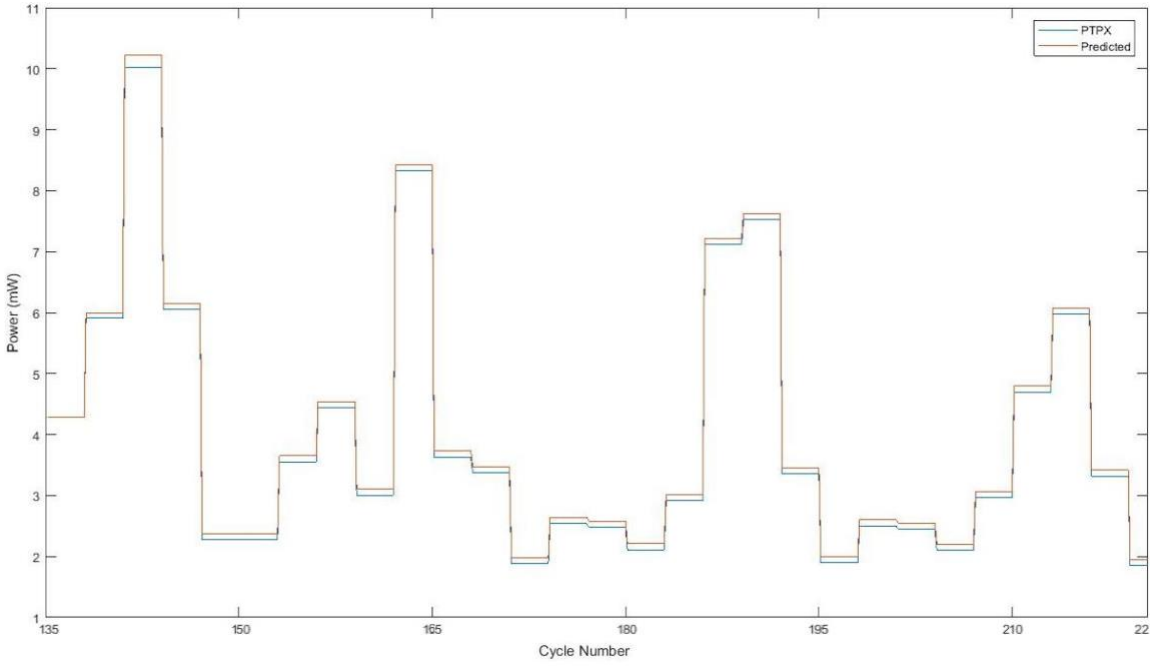

Figure 8: Window of cycles that show inaccurate predictions.

4.2 Comparison of Different Machine Learning Models

Figure 9, shows the cycle-by-cycle error of prediction for both the gaussian SVM and the decision tree models for all cycles. The gaussian SVM model was more accurate than the decision tree model, which can be seen as a general trend across cycles. Figure 10, zooms-in into a range of cycles where the gaussian SVM model is evidently more accurate than the decision tree model. Figure 11, shows a range of cycles where both models behave equally inaccurately with relative percentage error going as high as 22.5%. Possible reasons that could be behind this behavior are discussed in the future work chapter.

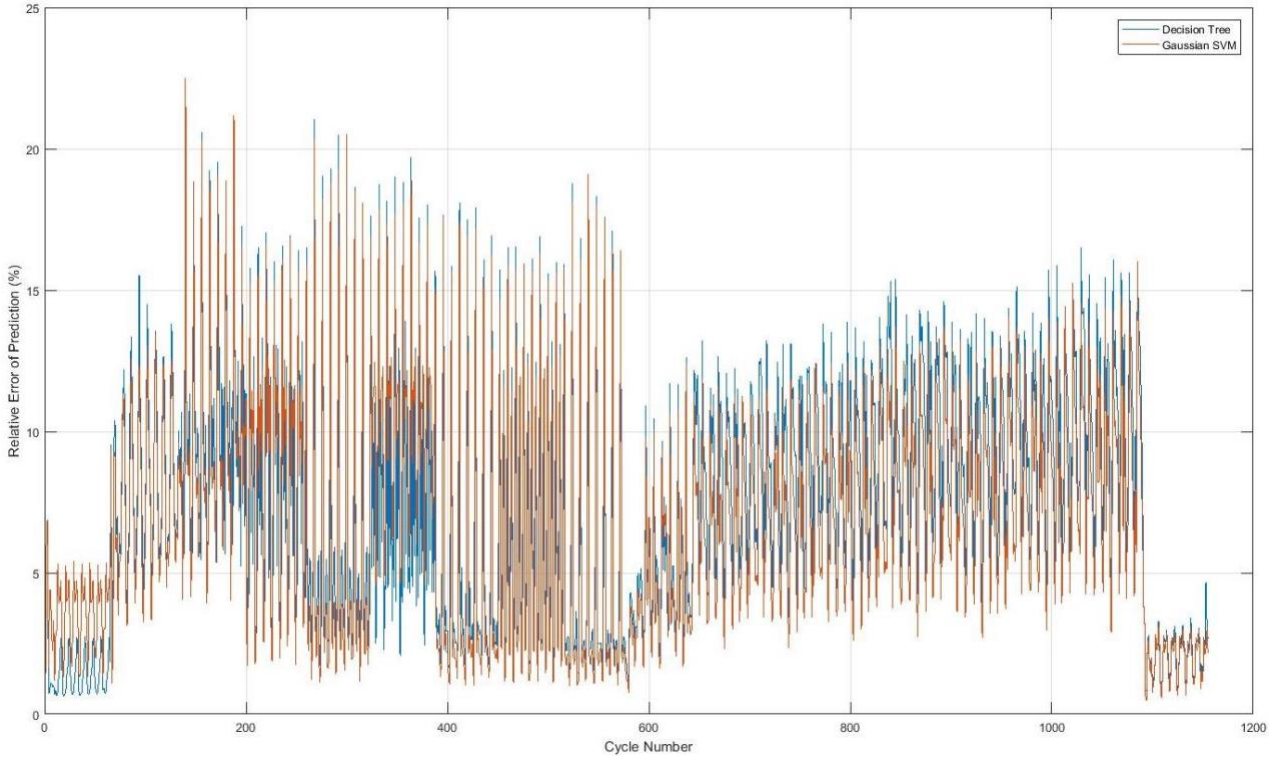

Figure 9: Cycle-by-cycle relative error of prediction for both decision tree and gaussian SVM models.

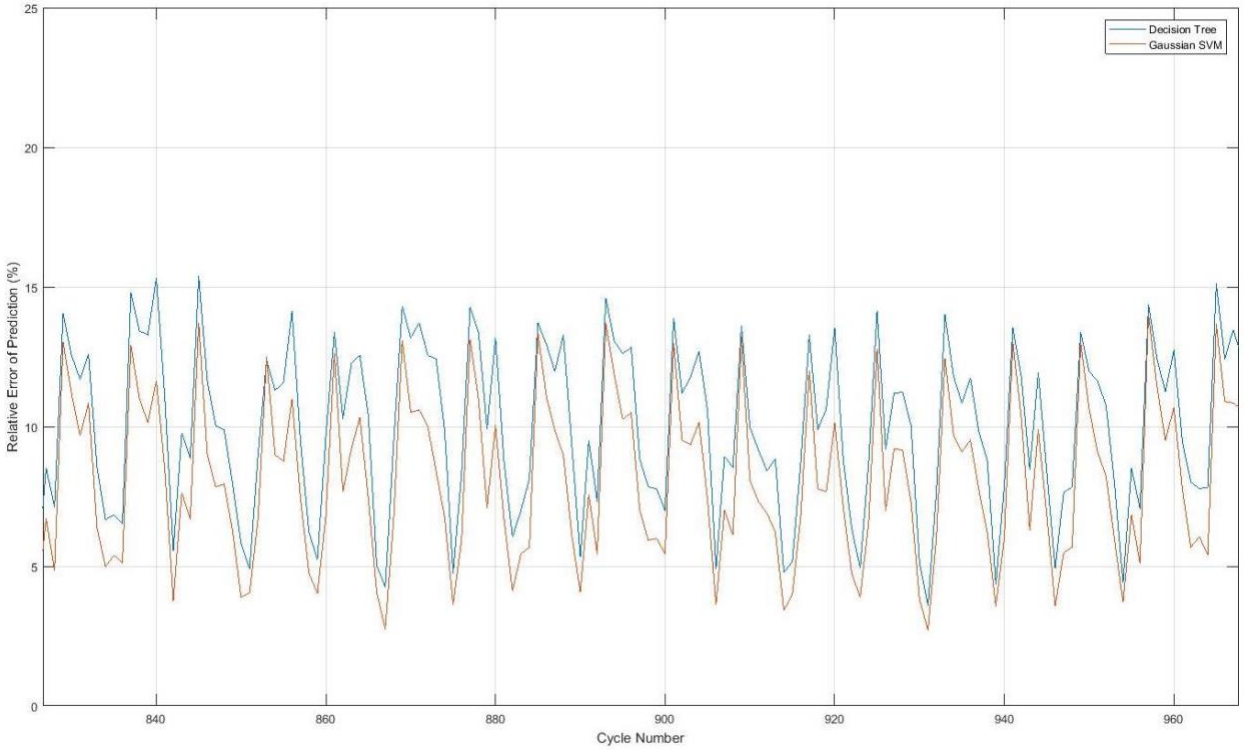

Figure 10: A window of cycles that shows cycle-by-cycle relative error of prediction for both decision tree and gaussian SVM models and illustrates gaussian SVM model being better.

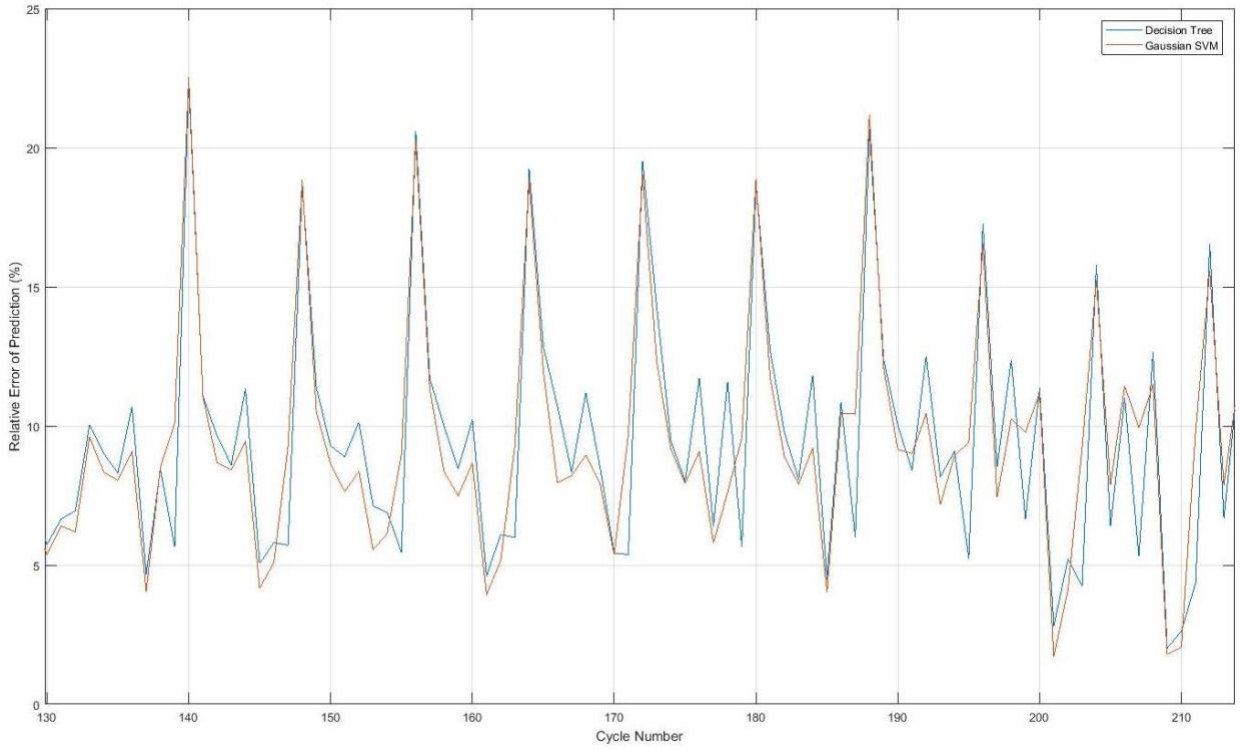

Figure 11: A window of cycles that shows cycle-by-cycle relative error of prediction for both decision tree and gaussian SVM models and illustrates the case where both models were inaccurate.

4.3 Effect of Training Set Size on Different Machine Learning Models

To test the effect of the training set size on the accuracy of the models, the average relative error of predictions was plotted against the size of the training set as shown in Figure 12. As expected, the accuracy increased as the size of the training set increased for both regression models. However, again, it is evident that the gaussian SVM was more accurate than the decision tree, but the simulation of the decision tree model approach was slightly faster as the size of the training invocations increased. Due to limitation in the total number of invocations available for training and testing, the training set size was not increased further although the simulated sizes are not considered

enough for obtaining a well-trained machine learning model. It is expected that with a much larger training set size even more accurate predictions could be obtained.

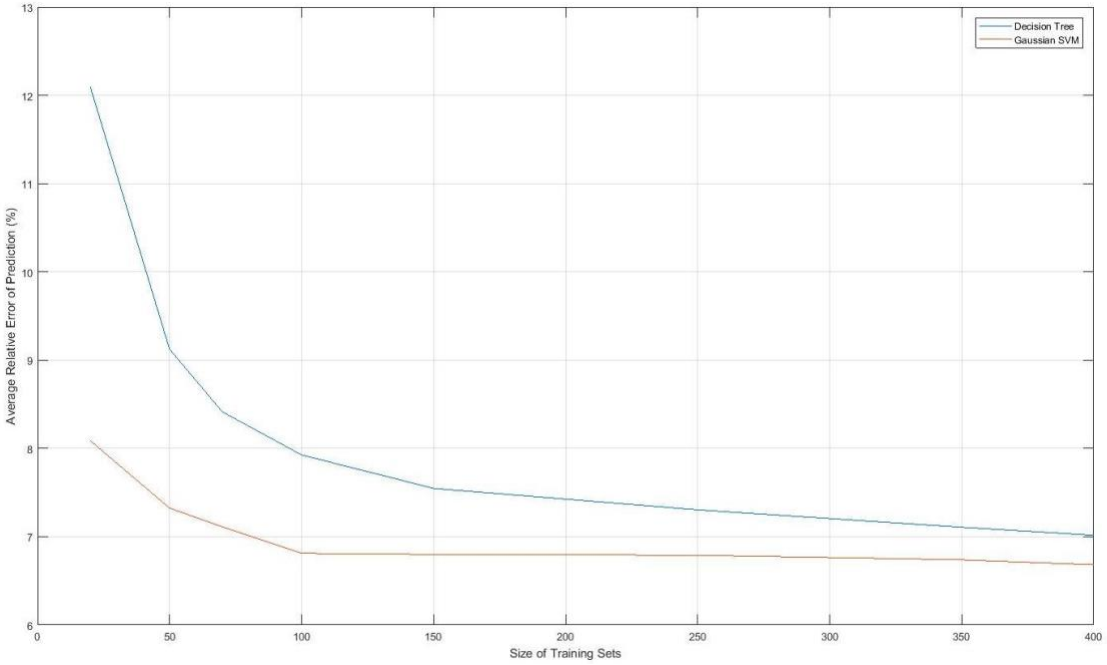

Figure 12: The effect of the training set size on the average relative error of prediction for different machine learning models.

4.4 Comparison of Cycle-Specific Models and Global Model Approaches

Normally, for a certain invocation, the cycle-by-cycle behavior of the RTL circuitry is different due to variety of signals switching during these cycles. For that reason, the design framework involves training independent cycle-by-cycle models with different features. The intuition behind this is that having one global model for all cycles is not going to be as accurate and effective; since considering the entire set of signals in the design as the features for the model would cause irrelevant features being selected for the power model, known as overfitting. This hypothesis is illustrated in Table 1, where an approach was simulated with cycle-by-cycle decision tree models versus an approach

with a single global decision tree model and measured the average relative error of predictions across different sizes of training sets. To start with, the accuracy of the cycle-by-cycle models approach is always almost twice that of the global model approach. Moreover, with increasing the size of the training set the accuracy of the cycle-by-cycle models approach improves significantly as opposed to the global decision tree model which seems to saturate at the same value of accuracy.

Size of Training Invocations (as % of constant # of total invocations)	Average Relative Error of Predictions (%)	
	Cycle-by-Cycle Decision Tree Models	Single Global Decision Tree Model
5	9.1147	13.6759
7	8.3678	13.5838
10	7.894	13.3625
15	7.5225	13.3306
20	7.3919	13.1893
25	7.3052	13.1647
30	7.1927	13.0799
40	7.0047	12.9943

Table 1: Comparison of average relative error of predictions between a cycle-by-cycle decision tree models approach and a single global decision tree model approach with varying size of training invocations.

4.5 RUN-TIME ANALYSIS

The approach presented in this report, aims at improving the speed of power predictions performed using a trained model for a given unmodified RTL, when compared to commercial gate-level power simulation tools. This subsection presents the time overhead of training the models and a comparison between the simulation speed of

the generated models and that of the gate-level. Table 2 illustrates the run-time values recorded for different steps of the design flow, that are specific to the 8-by-8 DCT design with training and testing sets consisting of 250 and 774 invocations, respectively. For both the PTPX gate-level and RTL simulations, all the 1024 invocations were simulated, where the cycle-by-cycle power values and the VCD file were obtained, respectively. The VCD file parsing step includes reading the generated VCD file, computing the signals' Hamming distances and writing the results to an output file. As for the feature selection step, the run-time reported includes the total time taken to perform appropriate feature selection, for each of the 1155 cycle-specific models that need to be trained, by considering the invocations from the training set. Similarly, the training step run-time includes the total time taken to train 1155 cycle-specific models with the given training set. Lastly, the prediction step run-time is the time taken to predict cycle-accurate power for the entire testing set, by simulating the trained cycle-specific models. In Table 2, the training and testing steps are analyzed for both the decision tree model (DT) and the gaussian SVM (GSVM) model. The run-time for both the training and prediction steps, is twice for the GSVM models when compared to the decision tree models.

Design Step		Run-time (s)
PTPX simulation		2920
RTL simulation		58
VCD file parsing		172
Feature selection		384
Training the cycle-specific models	DT	12
	GSVM	23
Cycle-accurate power predictions using the trained models	DT	4
	GSVM	7

Table 2: Total run-time values recorded for different steps of the design flow, that are specific to the 8-by-8 DCT design with a training set size of 250 invocations and testing set size of 774 invocations.

Table 3 shows the speed of simulation of the proposed approach, under two different models, and the speed of gate-level simulation.

Speed (Kcycles/second)						
RTL simulation	VCD Parser	DT	GSVM	Total DT	Total GSVM	Gate-level
21	7	224	128	5.13	5.04	0.405

Table 3: Speed of simulation of the trained models and gate-level power analysis.

The prediction flow involves RTL simulation, then parsing the generated VCD file, followed by the simulation of the trained models. Each of the first two steps, run at the speed of 21K and 7Kcycles/second, respectively. The speed of the last step in the prediction flow depends on the type of the regressor used to train the models, where the speed of prediction for the DT and GSVM cycle models is 224K and 128Kcycles/second, respectively.

Given the speed of the individual steps of the prediction flow, the total speed of the prediction flow, for each of the trained DT and GSVM cycle models, was calculated. Although the speed of simulating the DT trained model is almost twice that of GSVM, this step is not the bottleneck of the flow thus, both models' performance is approximately the same. The total prediction speed of both models is around 12x faster than the gate-level simulation. This shows the significant improvement in speed of simulation with the proposed approach. It is worth mentioning that while the performance of the RTL simulation step is limited by the commercial tools used, the performance of parsing the VCD file is probably improvable.

The training flow involves the steps of gate-level simulation, RTL simulation, VCD parsing, feature selection and training the models. The gate-level simulation and the feature selection steps are the major overhead with them running at a speed of 405 and 750 cycles/second, respectively. Although the total overhead of the training flow is significant, it is incurred once for a given unmodified RTL design. The proposed approach aims at achieving fast data-dependent predictions for new invocations based on a given trained model.

Chapter 5: Future Work

This chapter aims at providing an insight into the various improvements and modifications that could be applied to the proposed approach. It is clear from the results section that more testing needs to be done both by increasing the number of the invocations and by applying the approach to different benchmarks. Moreover, based off the preliminary results it seemed that for some cycle-specific models, the Hamming distances of signals across different invocations is not correlated with the power consumption. Raw signal data might need to be additionally employed as part of the features. One reason is that for control registers, the value of the register is what determines the behavior of the controlled logic thus, for the same Hamming distance, a completely different switching activity of the controlled logic is possible.

The results chapter presented a comparison of different machine learning models applied to the DCT benchmark. However, this might not always hold true for any benchmark. Therefore, by slightly increasing the training phase time, the flexibility of choosing the better model for the given benchmark could be implemented. One way of implementing that during the training phase is by running k-fold cross-validation [14] and choosing the model yielding a better accuracy for the given RTL. Moreover, the performance of the VCD parsing step, which is used during both the training and prediction flows, is a bottleneck in the latter and could probably be improved. One way of doing so is by integrating the VCD parser into the MATLAB framework, where the training and simulating of the models is performed, instead of performing this step in Python and transferring the data over to MATLAB.

Lastly, the results were compared against the commercial PTPX tool provided by Synopsys. Ultimately, the approach is targeting designs that will run on an FPGA and

thus real average power measurements need to be collected by emulating the design on an FPGA and these measurements will be used as the true reference for the accuracy metric.

Chapter 6: Summary and Conclusions

Early power estimation is critical for early design planning and it shapes design choices. This estimation is usually performed at different levels of abstraction, such as high-level C/C++ functional models, at the intermediate representation level, or for low-level RTL designs. A speed versus accuracy trade-off exists between these levels of abstraction, where going lower in the levels of abstraction enables the exposure to more information at the expense of slower simulations. This report presented an approach that leverages machine learning tools to attain cycle-by-cycle power prediction models at the RTL level by attempting to improve the speed of simulation. The current initial design framework mainly relies on the Hamming distance of signals as it is considered to be the switching activity of the circuit, which is directly correlated with the dynamic power consumption. As machine learning continues to prove its success, importance and efficiency in various applications, and with the vast interesting future work that could be investigated, this approach seems to be promising.

References

[1] D. Lorenz, K. Grüettner and W. Nebel, "Data-and State-Dependent Power Characterisation and Simulation of Black-Box RTL IP Components at System Level," *17th Euromicro Conference on Digital System Design*, Verona, 2014, pp. 129-136.

[2] K. Grüttner et al., "An ESL timing & power estimation and simulation framework for heterogeneous socs," *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, Agios Konstantinos, 2014, pp. 181-190.

[3] D. Sunwoo, G. Y. Wu, N. A. Patil and D. Chiou, "PrEsto: An FPGA-accelerated Power Estimation Methodology for Complex Systems," *International Conference on Field Programmable Logic and Applications*, Milano, 2010, pp. 310-317.

[4] E. Copty, G. Kamhi and S. Novakovsky, "Transaction level statistical analysis for efficient micro-architectural power and performance studies," *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, New York, NY, 2011, pp. 351-356.

[5] D. Lee, L. K. John and A. Gerstlauer, "Dynamic power and performance back-annotation for fast and accurate functional hardware simulation," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, 2015, pp. 1126-1131.

[6] D. Lee, T. Kim, K. Han, Y. Hoskote, L. K. John and A. Gerstlauer, "Learning-based power modeling of system-level black-box IPs," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Austin, TX, 2015, pp. 847-853.

[7] D. Lee and A. Gerstlauer, "Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 3, February 2018.

[8] I. Lee et al., "PowerViP: SoC power estimation framework at transaction level," *Asia and South Pacific Conference on Design Automation*, Yokohama, 2006, pp. 8 pp.-.

[9] D. Chen, J. Cong, Y. Fan and Z. Zhang, "High-Level Power Estimation and Low-Power Design Space Exploration for FPGAs," *Asia and South Pacific Design Automation Conference*, Yokohama, 2007, pp. 529-534.

[10] L. Zhong, S. Ravi, A. Raghunathan and N. K. Jha, "Power estimation for cycle-accurate functional descriptions of hardware," *IEEE/ACM International Conference on Computer Aided Design. ICCAD-2004.*, 2004, pp. 668-675.

[11] N. R. Potlapally, A. Raghunathan, G. Lakshminarayana, M. S. Hsiao and S. T. Chakradhar, "Accurate power macro-modeling techniques for complex RTL

circuits," VLSI Design. *Fourteenth International Conference on VLSI Design, Bangalore*, 2001, pp. 235-241.

[12] C. W. Hsu et al., "PowerDepot: Integrating IP-based power modeling with ESL power analysis for multi-core SoC designs," *48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, New York, NY, 2011, pp. 47-52.

[13] Y. H. Park, S. Pasricha, F. J. Kurdahi and N. Dutt, "System level power estimation methodology with H.264 decoder prediction IP case study," *25th International Conference on Computer Design*, Lake Tahoe, CA, 2007, pp. 601-608.

[14] A. Sylvain, C. Alain. A survey of cross-validation procedures for model selection. *Statist. Surv. 4* (2010), 40--79.