How Good Is Your Verilog RTL Code?
A Quick Answer from Machine Learning

Prianka Sengupta
prianka.sengupta@tamu.edu
Texas A&M University
College Station, Texas, USA

Aakash Tyagi
tyagi@cse.tamu.edu
Texas A&M University
College Station, Texas, USA

Yiran Chen
yiran.chen@duke.edu
Duke University
Durham, N Carolina, USA

Jiang Hu
jianghu@tamu.edu
Texas A&M University
College Station, Texas, USA

ABSTRACT

Hardware Description Language (HDL) is a common entry point for designing digital circuits. Differences in HDL coding styles and design choices may lead to considerably different design quality and performance-power tradeoff. In general, the impact of HDL coding is not clear until logic synthesis or even layout is completed. However, running synthesis merely as a feedback for HDL code is computationally not economical especially in early design phases when the code needs to be frequently modified. Furthermore, in late stages of design convergence burdened with high-impact engineering change orders (ECO's), design iterations become prohibitively expensive. To this end, we propose a machine learning approach to Verilog-based Register-Transfer Level (RTL) design assessment without going through the synthesis process. It would allow designers to quickly evaluate the performance-power tradeoff among different options of RTL designs. Experimental results show that our proposed technique achieves an average of 95% prediction accuracy in terms of post-placement analysis, and is 6 orders of magnitude faster than evaluation by running logic synthesis and placement.

KEYWORDS

Verilog RTL, Machine Learning, Performance and Power

1 INTRODUCTION

Hardware Description Language (HDL), such as Verilog and VHDL, is the universal entry point for most digital designs including ASICs and IP blocks. For a given design specification, a designer can take multiple approaches for writing the corresponding RTL code. These include, but are not limited to different state encodings for finite-state machines, choice of procedural assignment types, multiple style definitions of conditional logic and more. Moreover, the choice of micro-architecture can be considerably different between two RTL solutions that implement the same design. These differences can significantly impact the circuit performance and power. This is illustrated in Figure 1, where post-placement timing and power results from two RTL solutions of the same AES encryption IP are compared. The AES IP consists of around 12K gates and the two versions of the RTL have been developed by modifying the state machine encoding and increasing the depth of FIFO units used in sub-modules. The total negative slack (TNS) vs. dynamic power distribution of the two RTL solutions provides clear feedback on the relative design quality and challenges towards convergence. For applications that prioritize power over performance, RTL solution 1 is preferable whereas solution 2 offers more performance headroom from its lower TNS values.

Quite often, the impact of RTL design choice on the quality of results (QoR) becomes known only once the implementation results are available from synthesis, placement and routing (PnR). Although performance and power estimates are most accurate at the post-routing stage compared to the post-synthesis or post-placement stage, this knowledge comes at the cost of significant runtime. Even the runtime of mere logic synthesis can be very long if it is frequently invoked to assess the impact of RTL changes. Indeed, an RTL design needs to be evaluated with different synthesis parameters. For modern SoC design, it is not uncommon to spend a week in evaluating the quality of results (QoR) of an HDL design [12]. The long feedback time from implementation stages makes the evaluation of the RTL rather expensive, especially in the early stages of development where changes are frequent. Also for design IPs which are sourced externally, assessing their performance and power quality at an early stage is crucial for successful SoC integration. Furthermore, late RTL design changes introduce costly engineering effort across the chain of hierarchy from RTL designer to physical design (PD) feedback in an industry scenario. Therefore, the ability to gain early RTL design feedback has immense potential for improving design QoR, as well as for reducing turn-around time.

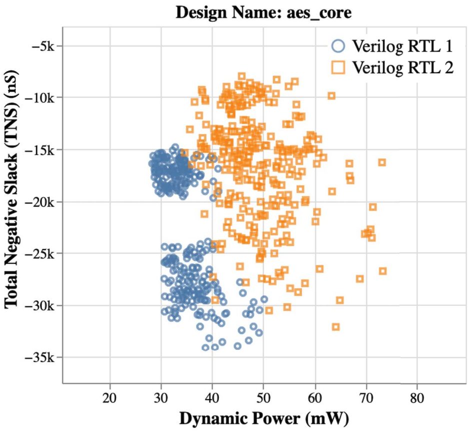


Figure 1: Impact of different Verilog codes for an AES encryption IP design. TNS and dynamic power values for each code are obtained through different synthesis parameter settings.

Most of the recent works utilize synthesis and PnR resources along with machine learning (ML) techniques to estimate power and or timing metrics[4, 6, 19, 20]. In these works, the features are derived from different stages of design implementation flow to leverage the ML prediction tasks. Among these works, RTL instrumentation techniques and simulation activities were studied for real-time power estimation in [20] and [6]. Both of these work achieve more than 90% accuracy in projecting workload specific dynamic power by automatic identification of representative RTL

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549375

signals for a given design. Although these approaches provide valuable power results, the models are not transferable to a new RTL design due to the requirements of design specific instrumentation. Another approach seen in the industry is to enhance the EDA tools in order to efficiently execute the design feedback cycle [2, 12]. These RTL power estimation and analysis tools rely on calibrated models to estimate power with claimed error margin of 5% - 10%. The runtime of these tools, although faster than legacy approaches, still consumes hours for each run and multiple days for exploring optimal implementation options. Machine learning-based PPA (Performance, Power and Area) modeling were developed in [4]. However, the chosen model features include only some RTL parameters instead of entire RTL code and therefore cannot capture the impact of different RTL coding styles. One related research explores design parameter tuning [19]. However, its machine learning model features consider only tool parameters and do not include RTL code or design information. Other related work explores delay prediction using RTL in [7], where statistics of internal cell connectivity is extracted as features to predict post-synthesis critical path delay. However, the prediction scheme has only been explored for combinational circuits such as adders and trained using synthetic design variations generated by a RTL generator. A former research[15] explores the idea of extracting the area and delay parameters from a subset of the design and uses a polynomial function based predictor to obtain estimates for the overall design. However, the approach is not transferable to unseen designs without manual refinement of model parameters. Also this work only focuses on the critical path delay estimation and doesn't report total negative slack (TNS), which is more useful when comparing the timing characteristics of two designs. To the best of our knowledge, no previous work can provide fast RTL code QoR assessment models that can be reused across different designs.

We propose a machine learning based approach to quickly predict post-placement total negative slack (TNS) and dynamic power without having the need to run synthesis and placement flow. The predictive models are trained offline and can be applied to unseen new designs outside of the training data set. Two prediction formulations are studied. In the first formulation, called value prediction, the total negative slack (TNS) and dynamic power for a specific set of synthesis parameter settings is predicted. This prediction additionally helps explore the synthesis parameter values that yield best performance-power tradeoff. The second formulation, called Pareto front prediction, predicts the Pareto front curve for the TNS-power tradeoff. Our proposed approach aims to deliver predictive analysis to help designers gauge the following:

- Assess the TNS and dynamic power trade-off of a given RTL design under a specific synthesis parameter setting.
- Assess the Pareto front of TNS and dynamic power tradeoff across multiple RTL sources.
- Compare quality differences between equivalent RTL IPs.

The results on a set of benchmark designs show that our approach can reach an average of 95% accuracy with respect to post-placement TNS and dynamic power. For evaluating timing and power for a single synthesis recipe, our approach is 6 orders of magnitude faster than running synthesis and placement flow. Here, a single synthesis recipe corresponds to specific values for a set of synthesis parameters which are used during a single synthesis run. For assessing the TNS-power tradeoff for multiple synthesis recipe, our proposed method performs 8 orders of magnitude faster than the synthesis and placement flow, i.e., our method can reduce the runtime from days to less than a second. Since our models can be directly applied to designs different from training data, the training cost is easily amortized. This is the first work that simultaneously achieves such high accuracy, fast speed and model reusability.

2 OVERVIEW

We take a two step approach to deliver the predictive analysis for any given RTL design. The first step decomposes the Verilog RTL code to AST (Abstract Syntax Tree) and graph structures and extracts important features from the design. The second step involves building and training machine learning models using these features collected from a wide range of designs along with synthesis parameters. Figure 2 shows the overview of the process to build the predictive model.

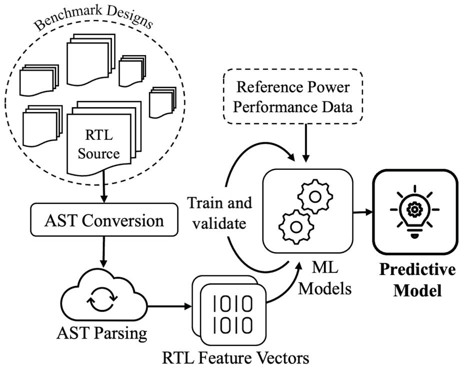


Figure 2: Overview of creating and training our model.

Building any reliable predictive model requires a representative training set, high fidelity feature extraction and careful model selection. In the proposed method in Figure 2, a wide set of designs from established benchmarks and multiple open source IPs are used for training and validating the models. For each of these designs, a novel AST based feature extraction method is applied to extract a set of features that can successfully represent and differentiate the characteristics of the RTL designs. To construct the spread of possible performance and power outcomes, each design is synthesized with a combination of multiple synthesis recipe parameters and placed using industry standard EDA tools. This implementation provides a cluster of data points which can be used to assess the TNS and dynamic power trade-off capability of a given design as well as to compare the tradeoff between two candidate IPs with equivalent functionalities. The predictive model uses the reference implementation data as target and is trained using a combination of extracted RTL features and synthesis parameters as input. A set of designs are kept detached from the training process to test the universal applicability of the Machine Learning models for any unseen design.

3 PROBLEM FORMULATION

In this work, we focus on the prediction of total negative slack (TNS) and dynamic power as these metrics align with the industry standard practice of performance and power optimization. TNS represents the ability of design to meet a specific operating frequency. A design with a low absolute TNS value implies that the RTL is suitable in quality with regards to timing. Often time, a trade-off between timing and dynamic power is decided based on the intended application of the design at hand. The trade-off characteristics between TNS and dynamic power is an inherent property of the design, which is dictated by the RTL coding style and also by the chosen micro-architecture. The two different predicted measures of TNS and dynamic power are described in the sections below. Our prediction is targeted to the accuracy of post-placement analysis as the first order layout effects on timing and power are mostly captured in a placed circuit. Our framework can be easily extended to handle other metrics, such as area, worst case slack and static power. In this work, we study two different formulations for the prediction.

3.1 Formulation 1: TNS and Power Value Prediction

In this formulation, TNS and dynamic power values are predicted for unseen Verilog RTL designs for specific settings of logic synthesis parameters (listed in section 6.1.1). This is because the same Verilog code may lead to different TNS and power results depending on the synthesis tool parameters. The dynamic power in focus is reported at a 20% average activity factor. When provided with RTL features and synthesis parameter sweeps, the predictor is trained to generate a set of dynamic power and TNS values. These predicted values closely resemble the outcomes from an actual implementation, as shown in Figure 3. Early prediction of the TNS and dynamic power allows longer headroom to optimize the RTL to achieve the desired trade-off.

3.2 Formulation 2: Pareto Front Prediction

Although the value prediction can provide an estimate of TNS and power, the estimate is for a single synthesis recipe. In order to have an overall assessment of quality for a Verilog code, one needs to run the model several times with the desired synthesis recipe parameter values. In the second formulation, our model will predict the TNS-power Pareto front curve among all results from a Verilog code independent of synthesis recipe parameters. In Figure 3, the Pareto front curve is indicated by the dotted line. A Pareto front curve can be described by a regression function. Based on our empirical study, we choose the function listed in equation 1.

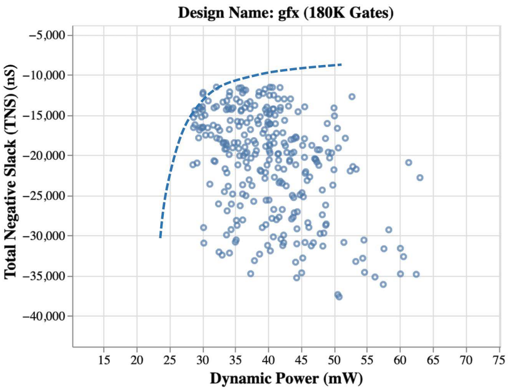


Figure 3: Post-placement solutions and the Pareto front curve for a lightweight graphics processor core.

$$y = a \ln(x - b) + c \tag{1}$$

Where x denotes dynamic power and y represents TNS. The values of coefficients a, b and c are obtained by fitting the training data. Since the Pareto does not relate directly to the synthesis tool parameters, we anticipate the Pareto curve coefficients to be established by the source RTL code features alone.

4 VERILOG RTL FEATURE EXTRACTION

Although Verilog is a hardware description language, it closely resembles C/C++. Therefore, we leverage a common programming language parsing technique called Abstract Syntax Tree to represent the Verilog source code as a tree structure that can facilitate the end goal of design feature extraction.

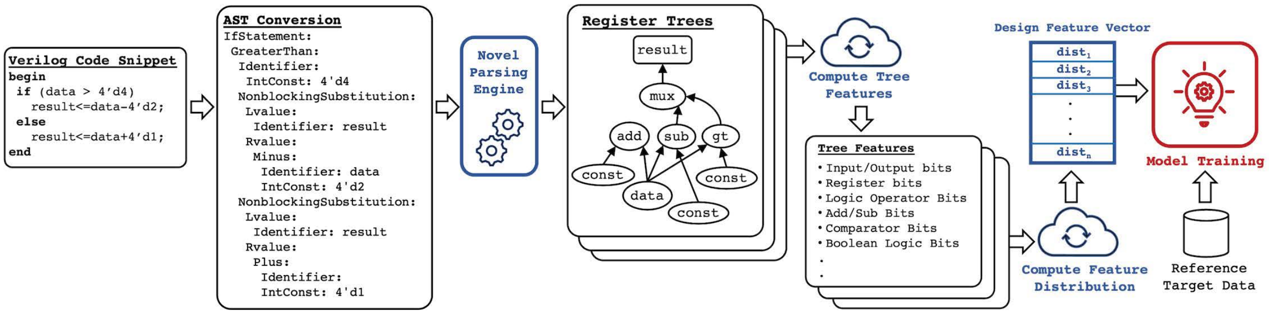
4.1 Verilog Abstract Syntax Tree (AST)

Abstract Syntax Tree is a tree like representation of any programming language with each node of the tree representing a language construct or keyword. The tree follows the hierarchy established in the source code starting from a root element and ending in many leaf nodes. Multiple open source tools exist for generating the AST from Verilog, namely Verilator [14], PyVerilog [16], Verible [1] among multiple others. In this work, we rely on Verilator to convert the source verilog of the benchmark designs to their respective AST representation. The AST generated from Verilator is saved in the XML format which is a markup format used to represent hierarchical data. Verilator also allows flattening the design during AST generation, which is beneficial as it resolves all the references to defined parameter and populates the logic from multiple module and function instances directly into the AST. The AST is provided as an input to the AST parsing engine which is one of the novel contributions of this work.

4.2 Parsing AST for Features

The AST generated from source RTL is parsed in two different approaches depending on the target machine learning model. For certain graph based ML models, the features need to be arranged in a graph format, whereas for the other non-graph based models the features need to be represented as a one dimensional vector. For both of these approaches, we discuss the parsing procedure and processing flow shown in Figure 4 and Figure 5.

4.2.1 Vector Based Feature Extraction: In our goal to build a predictive model, we explored a set of non-graph based ML models which expect a feature vector as input. For this mode of feature extraction every register assignment is identified in the AST and back-traced to build the logic tree responsible for driving the register. Once all register logic trees are created, a feature extraction procedure computes the distribution of multiple design features across all the



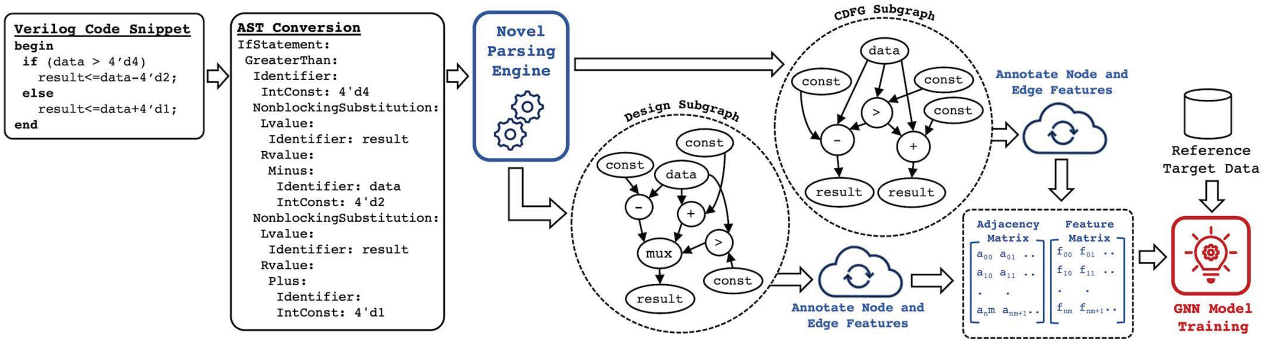
Verilog Code Snippet

```
begin
 if (data > 4'd4)
   result<=data-4'd2;
 else
   result<=data+4'd1;
end
```

AST Conversion

```
IfStatement:
 GreaterThan:
  Identifier:
   IntConst: 4'd4
  NonblockingSubstitution:
   Lvalue:
    Identifier: result
   Rvalue:
    Minus:
     Identifier: data
     IntConst: 4'd2
  NonblockingSubstitution:
   Lvalue:
    Identifier: result
   Rvalue:
    Plus:
     Identifier:
     IntConst: 4'd1
```

Figure 4: Processing flow for vector based feature extraction from Verilog



Verilog Code Snippet

```
begin
 if (data > 4'd4)
   result<=data-4'd2;
 else
   result<=data+4'd1;
end
```

AST Conversion

```
IfStatement:
 GreaterThan:
  Identifier:
   IntConst: 4'd4
  NonblockingSubstitution:
   Lvalue:
    Identifier: result
   Rvalue:
    Minus:
     Identifier: data
     IntConst: 4'd2
  NonblockingSubstitution:
   Lvalue:
    Identifier: result
   Rvalue:
    Plus:
     Identifier:
     IntConst: 4'd1
```

Figure 5: Processing flow for Graph based feature extraction from Verilog

logic trees. The design features of choice are: 1) Total Input bits and Output bits, 2) Total Register Bits, 3) Total Logic Operator Bits, 4) Total Adder/Subtractor Bits, 5) Total Multiplier Bits, 6) Total Comparator Bits, 7) Total Boolean Logic Bits. 8) Average Tree Depth and 9) Average Wire Width The features are computed for each logic tree and the distribution of these features across all the logic trees in the design is saved as a two dimensional feature vector. The processing flow for vector based feature extraction is presented in Figure 4.

4.2.2 *Graph Based Feature Extraction:* In this work we explore two different methods of graph feature extraction from AST.

The first type of feature graph back-traces the assignments to each register in order to acquire the cone of driving logic. All register components in the design are used to create unique nodes. All procedural assignments to these registers are identified in the AST followed by a search for logic operations, arithmetic operations or conditional logic that affect the assignment to the register in focus. The tracing method is continued until all inter-register connections are visited and appropriate edge connections are formed. In the resulting graph, nodes can be registers, arithmetic operators, boolean logic, constant register, input/output pins or muxes to represent two way conditional blocks. Edges are formed from direct assignments or dependency between two node elements. e.g. an arithmetic node will have edges connecting to its operators.

The second type of feature graph is constructed by tracing the control and data-flow graph (CDFG) of a given design. In this mode of graphs, the flow of data and intermediate controls are followed to construct the graph. The graph starts off with defined data elements or ports as nodes, such as input/output, registers and constant values. The progression of each data element within the design is traced out as a variable may get used in conditional operation, arithmetic operation or more. At the end of the control and dataflow tracing, the leaf node is often a data element that can be attached back to the root nodes. This element wise tracing procedure creates a single design graph which is exported as the design feature. Once a single graph for the design is constructed, a feature annotation procedure is executed to compute the following set of features for each node and edge - **Node Features:** 1) Total input bits, 2) Total output bits, 3) Encoded logic type (xor, and, add, sub, shifter etc), 4) Encoded node type (Arithmetic, Boolean, Data etc), 5) Number of neighboring nodes. **Edge features:** 1) Edge bit width, 2) Encoded source-destination pair type (reg>reg, reg>logic, logic>reg, logic>logic), 3) Edge type (data or logic). The processing flow for graph based feature extraction is presented in Figure 5.

5 MACHINE LEARNING MODEL SELECTION

In both modes of prediction, Value and Pareto Front, the output of the model is a numeric value representing either a metric value or a curve parameter. This implies the need for a regression based machine learning model. Multiple different machine learning regression schemes were evaluated against the prediction task. The ML models that were tested can be categorized into two main categories: models that expect a RTL feature vector as the input and models that expect an annotated design graph as its input.

5.1 Vector Based Regression Models:

Multiple ML models were tested that require a vector based input feature for prediction. The model types and their configuration is listed below:

- Linear Regression: We chose Linear Regression model [10] to understand the linearity of the data set.
- Random Forest Regression: In this work, a Random Forest model [8] with 100 estimation trees with a limited depth of 10 was used to ensure generalization across designs.
- Neural Network: In this work, a Neural Network model [18] was built consisting of three hidden layers with 256, 132 and 16 nodes respectively to predict the required numeric value of the prediction task.
- XGBoost Regressor: The XGBoost model [3] used in this work consisted of 80 tree estimators with a max depth of 7.

The feature vector extraction previously discussed in 4.2.1 generates a two-dimensional vector containing the distribution of features in a RTL design. This feature vector is flattened to a single dimension and normalized before providing as an input to the models mentioned above.

5.2 Graph Based Regression Models:

Graph Neural Network (GNN) [11] are a different branch of machine learning models that learns from graph like data and make graph level or node level predictions. We test two types of graph as input to our model:

- AST Based Graph Input
- CDFG Based Graph Input

In the TNS and dynamic power prediction task, we configure the GNN models to perform graph level numeric prediction. We focus mainly on GAT (Graph Attention Network) [17] for building the graph based regression model. The implementation of GNN and GAT models in Spektral [5] were used in this work.

6 EXPERIMENTAL RESULTS

6.1 Data Generation

A total of 51 Verilog RTL designs from the IWLS 2005 benchmark [9] and OpenCores were used to generate the training and testing data. Table 1 lists the design names and their size in terms of gate count that were considered to obtain the TNS and dynamic power values. It should be noted that the reported gate-count should only be used for relative size comparison between designs, as the absolute gate count may vary depending on the synthesis parameter settings. In order to familiarize the ML models with the parameter dependent variations, a large set of implementation data are generated using commercial synthesis and placement tools and used as the model target data. The target data to predict TNS and dynamic power values are collected from post-placement to capture any overhead from physical layout. The spread of these data values are obtained from synthesis recipe variations to construct 416 synthesis runs for every design. The designs were synthesized using a 45nm technology library with access to low, typical and high VT cells. The designs used for testing were kept separate from the training set to ensure confidence in the prediction models.

Table 1: 51 benchmark designs used for training data generation and testing.

45 Training Designs	Seq. Cells	Comb. Cell	Total Gates
ss_pcm	87	173	260
uart2bus	157	611	768
wb_dma	523	1,728	2,251
mem_ctrl	1,065	3,296	4,361
aes_core	530	11,887	12,417
⋮	⋮	⋮	⋮
fpu	663	31,881	32,544
xge_mac	13,301	33,366	46,667
scdma_viterbi	68,393	164,236	232,629

6 Testcase Designs	Seq. Cells	Comb. Cell	Total Gates
ethernet	2,344	7,232	9,576
vga_lcd	17,057	35,791	52,848
des3_perf	29,496	67,854	97,352
nova	29,083	90,870	119,953
gfx	48,490	148,475	196,965
jpegencode	39,113	223,411	262,524

6.1.1 Synthesis Parameters: Commercial synthesis tools provide a wide range of tune-able parameter that can impact the outcome of the netlist's timing and power characteristics. We leverage important tool parameters e.g Max Fanout, Max Transition, Max Capacitance, Max Leakage, High Fanout Net and Fanout Load to construct a combination of different values to generate the target data points for each design.

As mentioned earlier in section 3, two prediction approaches were considered in this work. The first one is for TNS and power value prediction and the second one is for Pareto front prediction. The training data collected from the 45 aforementioned RTL designs of varying sizes and functionalities are then used to evaluate each of these prediction tasks. 6 test designs listed in Table 1 were used for the testing.

6.2 Total Negative Slack (TNS) and Power: Value Prediction

In order to predict the values of TNS and dynamic power, six different types of ML models, mentioned in section 5, were trained for TNS and power separately. The RTL feature vectors extracted from the training designs are the primary input to these ML models. For every sample pair (input, target) the input feature vector F is appended with the specific synthesis parameter values p that were used to generate the target values y_p (Power) or y_t (TNS). For graph based models, p is appended to the feature vectors of every node. R^2 correlation coefficient has been used to measure the accuracy of the prediction models and is defined in equation 2.

$$R^2 = (1 - \frac{\sum(y_{pred} - y_{true})^2}{\sum(y_{true} - \overline{y}_{true})^2}) \tag{2}$$

The prediction results across different ML models for each test case design in Figure 7 show that our proposed method achieves an

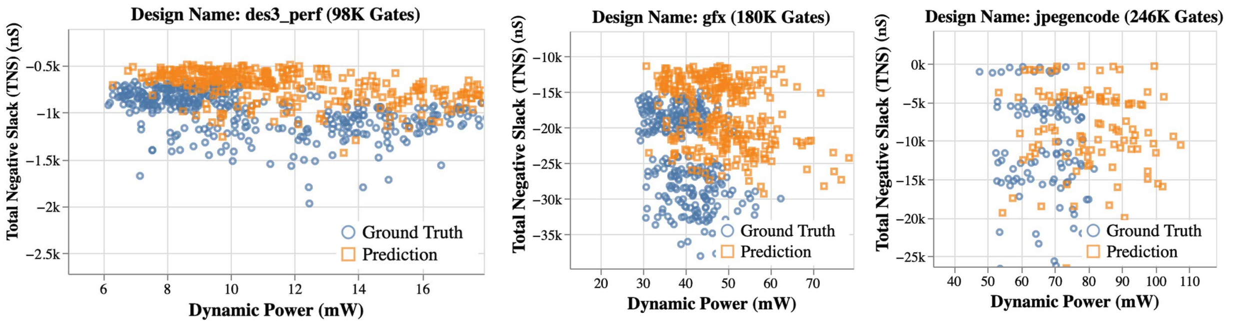


Figure 6: Prediction vs ground truth for the XGBoost prediction model for three testcases.

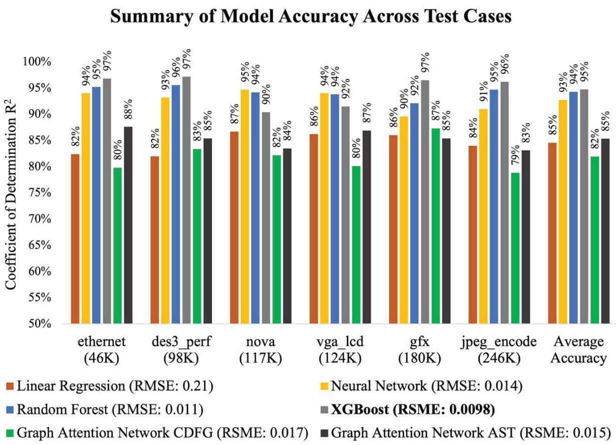


Figure 7: The R^2 coefficient of determination and normalized root mean squared error (RSME) for TNS and dynamic power prediction.

average of 95% accuracy and the XGBoost Regressor preforms best for the TNS and power value prediction. The best score possible for a model is 1.0 and for a constant model that always predicts a fixed value regardless of the input features would get a score of 0.0[10]. It was observed that simpler models such as Linear Regression are unable to fully capture the non-linear relation between RTL features and their corresponding impact on TNS and power. Also the generalization characteristics of Neural Network and Graph based models yield a narrower spread of predicted values compared to ground truth. Both Random Forest and XGBoost rely on tree based parameter segmentation and successfully learns not only the feature-to-target relation, but also the effect of inter-feature interaction as observed. The prediction quality vs ground truth for three testcases of increasing size, small, medium and large can be observed in Figure 6.

6.3 TNS vs. Power: Pareto Front Prediction

For predicting the Pareto front of each test design, the same six different ML models were trained with the prediction target being the a, b and c parameters in equation 1 required to fit the Pareto front curve. For each design, the post-placement solutions at the Pareto front of TNS-power tradeoff need to be fitted with the formulation listed in equation 1. However, this approach yields only one set of a, b and c value for each design which equates to only 45 sets of training samples. To avoid over-fitting to this small data set, we choose to increase the training data set using augmentation [13]. By adding a marginal amount of noise to the a, b, c parameters of the original Pareto curve, 20 new synthetic Pareto lines were constructed. Each of these lines are tightly coupled to the original Pareto curve. After augmentation, the training data consisted of 900 training samples. It should be noted that for the Pareto prediction, only the RTL feature vector F or design graph D_G is used as the input. The synthesis parameters p does not need to be appended to the input, which differs from the value prediction approach.

The metric of accuracy for this prediction task was set to be the Average Distance from Reference Set (ADRS) [19]. A lower ADRS value indicates that the data points along the predicted Pareto curve are closer to the data points on the reference Pareto curve. Assuming the set of data points along the true pareto-front is denoted by G and the predicted points are in I. Data points in G and I contain power and TNS value pairs (P_γ, T_γ) and (P_λ, T_λ) respectively. The ADRS between G and I is defined as:

$$ADRS(G, I) = \frac{1}{|G|} \sum_{\gamma \epsilon G} \min_{\lambda \epsilon I} \delta(\gamma, \lambda) \tag{3}$$

$$\delta(\gamma = (P_\gamma, T_\gamma), \lambda = (P_\lambda, T_\lambda)) = max(0, |\frac{P_\gamma - P_\lambda}{P_\lambda}|, |\frac{T_\gamma - T_\lambda}{T_\lambda}|) \tag{4}$$

The best possible score for ADRS is 0.0, indicating the closeness of the predicted Pareto front line to the reference Pareto front. The achieved ADRS for the test designs are summarized in Figure 9 which shows that the proposed method is successfully able to predict the pareto curve coefficients with minimal ADRS of 4.4. In this case, the Neural Network model performs the best followed by Graph Attention(GAT) Network model due to their ability to resolve compound features and discover non-linearity.

6.4 Runtime Comparison

One of the significant achievements of this work is the runtime savings of 6 orders of magnitude in the TNS and dynamic power prediction and 8 orders of magnitude in the Pareto front curve prediction. Table 2 compares the runtime for single prediction of TNS-power values and the pareto curve coefficients with respect to the traditional synthesis and placement. Each of the single value predictions represent the true TNS and dynamic power expected from a single set of synthesis parameter setting as mentioned in

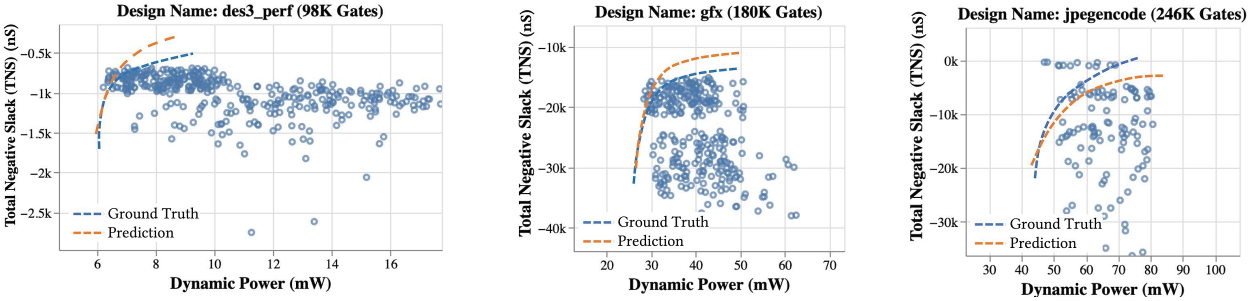


Figure 8: Pareto Front prediction using Neural Network prediction model for three testcases.

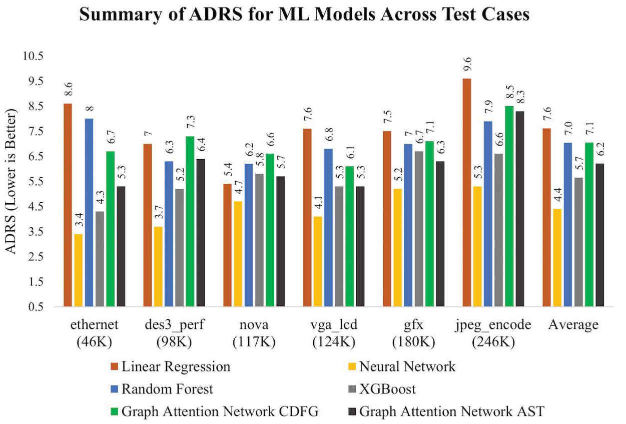


Figure 9: Measure of closeness (ADRS) between predicted Pareto Front and ground truth Pareto front.

Table 2: Runtime comparison for single synthesis recipe.

Model	Synthesis and Placement Runtime	Value Inference Time[1]	Pareto Inference Time[2]
ethernet	2820 Seconds	0.014 Seconds	0.05 Seconds
vga_lcd	19500 Seconds	0.014 Seconds	0.05 Seconds
des3_perf	9180 Seconds	0.014 Seconds	0.05 Seconds
nova	26220 Seconds	0.014 Seconds	0.05 Seconds
gfx	22320 Seconds	0.014 Seconds	0.05 Seconds
jpeg_encode	27720 Seconds	0.014 Seconds	0.05 Seconds
Average	17960 Seconds	0.014 Seconds	0.05 Seconds
Speedup	**1X**	**~1240000X**	**~360000X**

[1] For best value prediction model: XGBoost Regressor

[2] For best Pareto prediction model: Neural Network

section 6. Figure 10 compares the average runtime between 416 synthesis runs with different parameter settings, the runtime of the proposed method for predicting TNS and dynamic power and for predicting the Pareto front of TNS-power tradeoff. In this scenario, the runtime of Pareto front prediction is significantly faster due to the fact that it has to predict the Pareto curve coefficients only once, in contrast with the 416 predictions needed from the value prediction model.

The training time of the models that were studied fall in the range of 7~27 minutes depending on the model being trained. Although, generating the training target data takes considerable machine time, it is a one time effort that can be used universally once completed. The synthesis, placement and inference times in table 2 were collected on a 10 Core Intel(R) Xeon(R) CPU E5-2680 (2.80GHz).

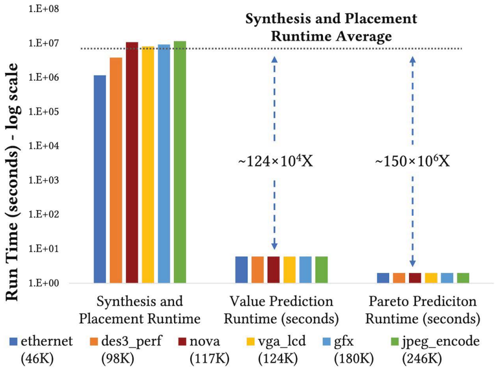


Figure 10: Average runtime comparison for capturing the effect of multiple synthesis recipe.

6.5 Application: RTL Comparison Study

One of the key application of our proposed prediction method is the inspection of two RTL codes that implement the same functionality. The coding style and the micro-architecture chosen during development can vary between RTL design teams or individual developers and affect the post-placement performance and power significantly. To evaluate the sensitivity of the prediction model to coding style and micro-architecture variations, the RTL of the test-cases were modified to specifically inject coding style changes and micro-architectural changes.

6.5.1 Effect of RTL Coding Style: Verilog HDL offers multiple ways to achieve the same logic, e.g one can use either blocking or non-blocking statements in the procedural blocks to assign values to register elements. Similarly, both if-else code blocks and switch-case code blocks can be used for implementing multi-criteria conditional logic. These coding style differences can introduce minor movement to the timing and power results. Figure 11 shows the sensitivity of the prediction models between a given design and its style varied version. In each of its sub-figures, the left part are the post-placement results and the Pareto front fitting of two RTL

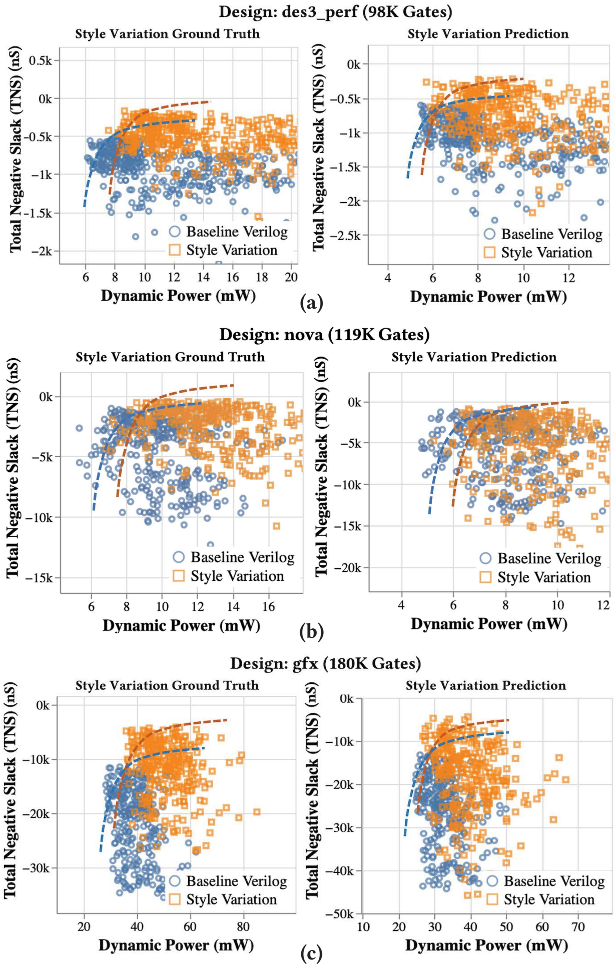


Figure 11: Prediction vs ground truth for three testcase with RTL Coding Style Variation.

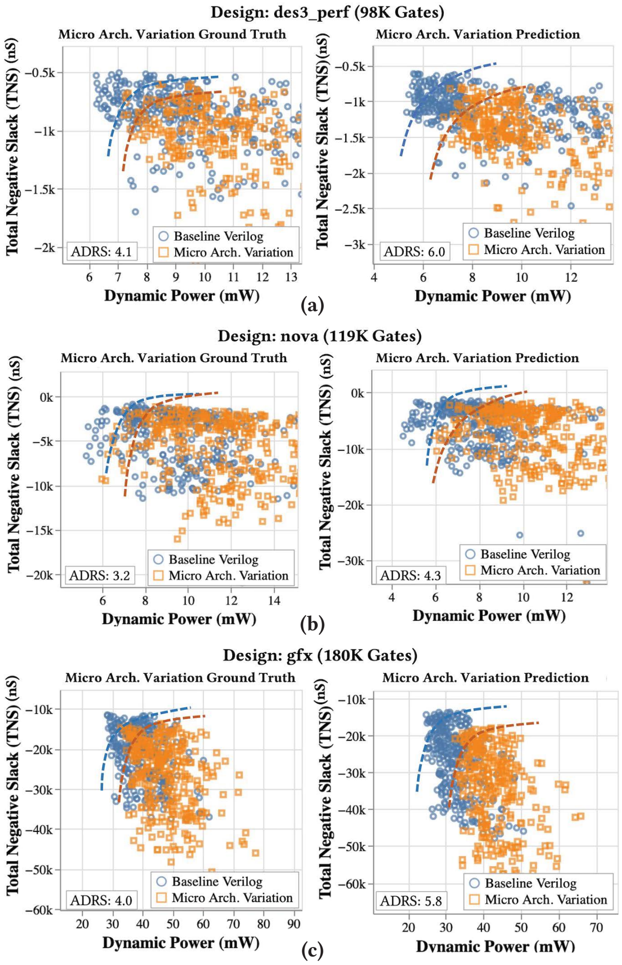


Figure 12: Prediction vs ground truth for three testcases with RTL Micro-architecture variation.

codes with style differences. The right part are predicted values prediction and Pareto front from our method for the same two RTL variants. One can tell that the left and right charts show the same trend and relative comparison. As such, our method can help designers to compare two Verilog codes in an accuracy similar to post-placement analysis but 6 orders of magnitude faster, i.e., reducing the evaluation time from days to less than a second.

6.5.2 Effect of Micro-Architecture Choice: The micro-architecture of a design defines low level details of the design, including but not limited to: state encoding of finite state machines, arrangement of data muxes and configuration of FIFO depths. These choices cast a direct effect on the TNS and dynamic power that can be achieved from the design. Figure 12 shows the sensitivity of the prediction models between baseline testcase design and a modified version where several micro-architectural changes have been made. The similarity of the true values and Pareto fronts (in left charts) and predicted results (in right charts) shows the ability of the predictive model to successfully adjust the predictions considering the effects of micro-architectural changes. In each chart, the ADRS between the two Pareto front curves of the two RTL variants is labeled. Both the ground truth and the prediction lead to the same ADRS ranking: $ADRS(des3\_perf) > ADRS(gfx) > ADRS(nova)$. Therefore, our predictive model provides good fidelity in not only understanding which RTL variant may dominate the other in power-TNS trade-off but also the difference between two Pareto front curves.

7 CONCLUSION

In this paper, we have presented a machine learning based total negative slack (TNS) and dynamic power prediction method from Verilog RTL code which utilizes a novel RTL code parsing mechanism. The proposed method successfully provides the assessment of a given RTL design in 6 orders of magnitude faster turn-around time in terms of TNS vs. dynamic power values for a given synthesis recipe (parameter setting) and the Pareto fronts. This enables a quick comparison feedback of equivalent RTL design qualities. The proposed work can be easily ported to other HDL languages and can be extended to predict workload specific power profiling.

ACKNOWLEDGMENTS

This work is partially supported by Semiconductor Research Corporation GRC-CADT 3103.001/3104.001 and National Science Foundation CCF-2106725/2106828.

REFERENCES

[1] Tim Ansell and Mehdi Saligane. 2020. The Missing Pieces of Open Design Enablement: A Recent History of Google Efforts: lnvited Paper. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–8.

[2] Ansys. 2021. *PowerArtist*. https://www.ansys.com/products/semiconductors/ansys-powerartist

[3] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[4] W Rhett Davis, Paul Franzon, Luis Francisco, Billy Huggins, and Rajeev Jain. 2021. Fast and Accurate PPA Modeling with Transfer Learning. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–8.

[5] Daniele Grattarola and Cesare Alippi. 2021. Graph neural networks in tensorflow and keras with spektral [application notes]. *IEEE Computational Intelligence Magazine* 16, 1 (2021), 99–106.

[6] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. Simmani: Runtime power modeling for arbitrary rtl with automatic signal selection. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 1050–1062.

[7] Daniela Sánchez Lopera, Lorenzo Servadei, Vishwa Priyanka Kasi, Sebastian Prebeck, and Wolfgang Ecker. 2021. RTL Delay Prediction Using Neural Networks. In *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*. IEEE, 1–7.

[8] Gilles Louppe. 2014. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502* (2014).

[9] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. 2006. Integrating logic synthesis, technology mapping, and retiming. In *Proc. IWLS'05*. Citeseer.

[10] 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. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[11] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[12] Jim Schultz. 2021. *RTL Architect: Parallel RTL Exploration with Unparalleled Accuracy*. Technical Report.

[13] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.

[14] Wilson Snyder. 2004. Verilator and systemperl. In *North American SystemC Users' Group, Design Automation Conference*.

[15] Arvind Srinivasan, Gary D Huber, and David P LaPotin. 1998. Accurate area and delay estimation from RTL descriptions. *IEEE transactions on very large scale integration (VLSI) systems* 6, 1 (1998), 168–172.

[16] Shinya Takamaeda-Yamazaki. 2015. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. In *International Symposium on Applied Reconfigurable Computing*. Springer, 451–460.

[17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[18] Sun-Chong Wang. 2003. Artificial neural network. In *Interdisciplinary computing in java programming*. Springer, 81–100.

[19] Zhiyao Xie, Guan-Qi Fang, Yu-Hung Huang, Haoxing Ren, Yanqing Zhang, Brucek Khailany, Shao-Yun Fang, Jiang Hu, Yiran Chen, and Erick Carvajal Barboza. 2020. FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 19–25.

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