Learning Based Timing Closure on Relative Timed Design

Tannu Sharma, Sumanth Kolluru, and Kenneth S. Stevens

University of Utah, Salt Lake City, USA

Abstract. Relative timed circuits leverage formal timing specifications to design and optimize integrated circuits. Relative timing can be applied to specify design correctness and performance properties of digital circuits in the form of a set of timing constraints. These circuits often show significant performance and power advantages over other approaches, but require assistance to automate timing driven synthesis and place and route in commercial electronic design automation (EDA) tools. A machine learning based automatic timing closure solution for relative timed circuits is presented. The machine learning implementation is expected to speed-up the process by learning from the features during each iteration, minimizing the overall run-time to timing close a design. A comparative study between regression model based and gradient boosting tree based solutions with an algorithmic approach is presented. Power and performance of the circuits are improved while reducing overall run-time required to timing close a relative timed design with commercial EDA tools.

Keywords: relative timing, timing closure, heuristic, greedy, machine learning, gradient descent, boosting, regression, EDA.

1 Introduction

Time delays are manifested in the components and wires of an integrated circuit (IC). Delays are evaluated based on a timing path between two points in a circuit, which consist of a sequence of components a signal must pass through. Time delays dictate the robustness, performance, and power of a system. Static timing analysis is employed to evaluate and optimize delays and to close timing during synthesis and layout [1].

Traditional techniques employed by commercial electronic design automation (EDA) tools are insufficient to close timing on a relative timed (RT) design. RT timing paths can be cyclical, and may be controlled by state bits in sequential logic implemented as combinational gates with feedback. To improve quality of relative timed designs, an engine that understands relative timing is required to obtain delay target values and sign-off timing in the current commercial framework.

1.1 Relative Timing

Relative timing (RT) is a universal representation of the sequencing property of time [2]. Sequencing in the time domain is a common correctness requirement used in integrated circuits. An example of a critical well known correctness requirement in the

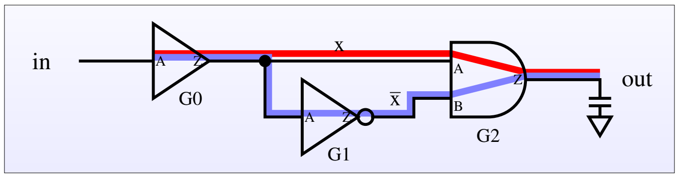

Fig. 1: Pulse generator circuit. The maximum delay path from the rising edge of `in` (pod) to the rising edge of `out` (poc_0) of Eqn. 2 is highlighted in red, and the minimum delay path from the rising edge of `in` (pod) to the falling edge of `out` (poc_1) is highlighted in blue. The capacitor models wire and gate capacitance of a latch array.

time domain is the storing of data in a flip-flop. Data must arrive at a flop earlier than the clock.

The simplified universal specification of a relative timing constraint is shown in Eqn. 1 [2]. Relative timing constraints require there must be a common timing start point called a point of divergence (pod) which has causal paths to two timing endpoints called points of convergence (poc_0, poc_1). To ensure delay of the early path of the constraint is always less than the delay of the late path requires taking the maximum delay from pod to poc_0 (plus margin m) and the minimum delay from pod to poc_1. RT expressions employ unbounded delays, and thus are a property of the circuit structure. Therefore, RT constraints are agnostic to specific implementation details of a design which affect circuit delays such as technology node, device sizes, or standard cell layout. Specific path delays are not known until design instantiation and timing closure.

$$\mathsf{pod} \mapsto \mathsf{poc}_0 + m \prec \mathsf{poc}_1 \tag{1}$$

Assume one needs to synthesize and verify a circuit that generates a pulse. Such a circuit could be used to pulse clock a latch bank. The circuit in Fig. 1 generates a pulse on the `out` net upon a rising edge of the `in` net when proper circuit delays are employed. The delay path through the inverter can be designed to generate the required minimum pulse width. The relative timing constraint (RTC) to correctly realize the pulse generator of Fig. 1 is shown in Eqn. 2. This produces a pulse on net `out` with a minimum width m. The '+' or '-' appended to each net name indicates a rising or falling transition respectively on the net. The causal path through the circuitwhich creates a rising edge on `out` transitions through pin `A` of gate `G2`; the causal path for the falling edge is through pin `B`.

$$\text{in+} \mapsto \text{out+} + m \prec \text{out-} \tag{2}$$

A relative timed design can contain as many as several million RT constraints. Many of these paths conflict, because maximum and minimum delay path segments can partially or completely overlap [3]. Manually converging timing by modifying the timing constraints on individual paths is not a feasible option for such designs, and it may not be possible to resolve the large number of violating paths through mere post-layout ECO [4]. As such, an automated aid to produce timing closed designs is required.

The timing on a relative timed design is complex, where the optimization of one path may affect timing of other paths (associated or non-associated). The entire problem is an interaction of non-convex optimization algorithms across often competing timing path constraints. In order to model the delay target value for each path, the algorithms consider device sizes, drive strength, transition capacitance, fanout, derating factors, and EDA uncertainty along with the affects of other paths.

This chapter discusses timing closure methodologies for relative timed design. The key contribution of this work describes a machine learning based timing closure engine (MLTC) developed to minimize the number of iterations required to converge timing on complex relative timed designs. Machine learning timing closure results are compared with a heuristic based timing closure (HBTC) CAD tool that automatically generates functional delay targets for a complete RT constraint set [5].

The MLTC engine is capable of generating a timing closed design with arbitrary initial delay targets, including initial maximum delay targets as zero (0 ps). The tool is usually able to produce a completely closed set of constraints with no negative slack violations.

The machine learning based timing closure tool is evaluated on a variety of designs for run-time (number of iterations), power, performance, and design robustness. Several types of RT constraints are employed in these designs. Pipelined designs employ the bundle data design style with handshake controllers [6].

2 Background

Relative timing passes a large set of overlapping maximum and minimum path delay constraints to the EDA tools for timing driven optimization. The interaction between these constraints in commercial timing driven EDA optimization algorithms is complicated and non-convex. A small variation in one maximum or minimum path delay constraint can create a large perturbation in slacks on seemingly unrelated paths. This unexpected variation is based on physical placement among other factors in the EDA algorithms.

While the HBTC algorithm produces solutions of reasonable quality, a number of factors encouraged us to search for better algorithmic solutions. In order to obtain good results, the heuristic based approach requires many iterations through the synthesis or place and route (PnR) tools resulting in very large run times. Assessing and configuring the heuristic tool to include a number of second order factors was difficult. These optimizations include important aspects such as competing maximum and minimum delay paths, whether the path is a timing critical delay path, determining when performance improvements are coming at the cost of too much power, and other interdependent factors.

Multiple gradient descent based algorithms were implemented and tested while exploring the algorithmic solution space to solve the relative timing closure problem. Because the search space is a non-convex optimization problem, none of the gradient descent solutions were able to generate a timing closed design with quality of results similar to those achieved by the greedy heuristic based timing closure engine. Therefore we investigated algorithms that employ a supervised learning based framework.

Machine learning (ML) algorithms are an excellent application to this problem space if data is carefully prepared. The ML based approach is not used to replace the current EDA tools but to improve the quality of RT design flow with commercial EDA tools. The timing constraints and their associated paths are unique to each design, so our initial work is not generalized by training on a sample designs. Rather this work employs the more flexible and programmable search properties of ML algorithms to achieve better solutions than heuristic or gradient descent based approaches.

Data plays an important role for any machine learning problem, especially when there may not exist a direct correlation between data among designs or data paths. Due to this ML has already carved its niche in the EDA industry to find design or timing solutions [7–14]. In our approach two supervised learning based regression models were implemented for a given set of input and outputs. One to handle smaller designs with a basic algorithm and second for complex relative timed designs with a more sophisticated algorithm. Both the approach works well on all designs (small and large), with a better run-time on complex relative timed designs with the later.

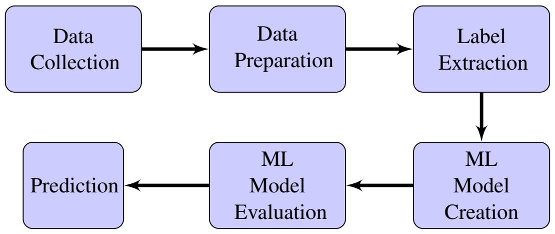

Fig. 2: General Approach to use Machine Learning in RT timing driven optimizations.

3 Approach

The general approach to use machine learning in relative timing driven EDA optimizations is shown in Fig. 2. The timing closure algorithm proceeds by using an ML algorithm to update path delay values for relative timing constraints. The updated path delays are passed to commercial EDA tools to perform timing driven synthesis or place and route (PnR) of the design. Each iteration with ML includes a synthesis or PnR run and timing analysis with PrimeTime. The results generated by the ML algorithm are also validated to ensure the RT constraints are obeyed with each new prediction.

3.1 Data Collection and Preparation

A careful selection of both timing and physical design parameters, is necessary during data collection. It is equally important to discard irrelevant/overlapping features to maintain the quality of data during data preparation. After extracting the relevant features, it is also important to adjust the hyper-parameters to improve the quality of the predictive model during model evaluation.

3.2 Machine Learning Algorithms

A tree based gradient boosting framework and a polynomial regression based algorithm were tested for RT timing closure.

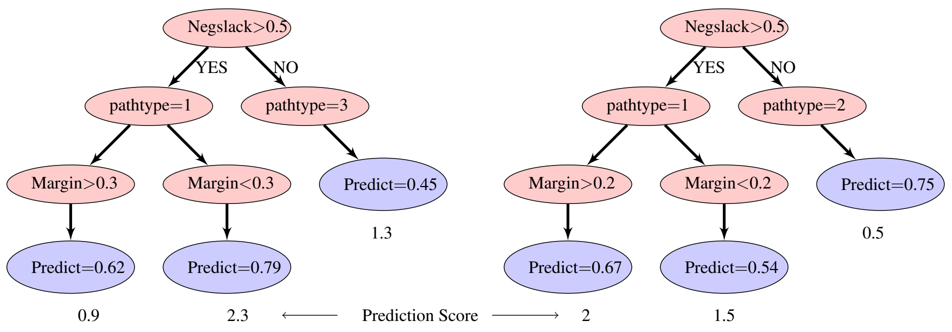

Fig. 3: Decision tree Example

Gradient Boosting Decision Trees Decision tree ensembles create a gradient boosted decision tree (gbdt) model [15]. The problem of estimating accurate delay target values with no negative slack is modeled here. The model created from the training data consists of an ensemble of trees, where each tree makes a prediction. All the predictions are taken into account by the ensemble model to make the final prediction. The prediction scores of each individual tree are summed up to get the final score. In Eqn. 3, K is the number of trees, $\overline{y_i}$ is the prediction, x_i is the input, f is a function, and F is all the possible decision trees.

$$\sum_{k=1}^{K} \overline{y_i} = f_k(x_i), f_k \in F \tag{3}$$

Based on the extracted features, decision trees are created to establish a relationship between inputs x_i and output y_i. The objective function and various tree pruning techniques are utilized to optimize the model. Based on the prediction scores and weights assigned to the leaf nodes, the final prediction value is calculated. A model similar to Fig. 3 is created during each iteration. In our case, the inputs are the timing constraint parameters and path characteristics. In Fig. 3 two such trees are illustrated. In practice, many such trees are constructed by the gradient boosting decision tree based learning algorithm.

The initial experiments were performed using extreme gradient boosting (*xgboost*) algorithm. However, it was observed, that the delay predictions of the *xgboost* model were inaccurate due to improper inclusion of encoded categorical features during each prediction. With the *xgboost* model, no timing closure could be achieved after $+500$ iterations on a small design and $+100$ iterations on a large design. The light gradient boosting model *lgbm* implementation can handle encoded categorical features well. It allows to establish a relationship between paths that are dependent on each other (especially in a complex design).

LightGBM (*lgbm*) is a gradient boosting model that can also use a tree based learning algorithm [16]. The framework is capable of handling large data sets. It performs supervised learning on data with multiple features in order to predict a target variable, which is a delay value in this case [17]. LightGBM is faster, outperforms other algorithms like extreme gradient boosting (*xgboost*) considerably [18], and can handle categorical data.

Polynomial Regression Polynomial regression is a technique used to establish a relationship between an input x and an output y. It is modeled to fit a non linear relationship, which means x has degree n. A linear regression model can be formulated as $y = \beta_0 + \beta_1 x$ where the β_i values are constants. A polynomial regression is modeled as $y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_n x^n$ [19]. The higher degree model is responsible for the raised non-linearity in the relationship of x and y. Polynomial regression models are much more accurate than linear models, but they tend to over-fit the input data. The model is more suitable to minimize variance in unbiased estimators of the coefficients.

3.3 Models

It is worth noting that inclusion of the right features as part of the data set is more important in this problem than the choice of algorithm to implement the learning based model. The behavior of the model is directly driven by the quality of the data provided to the model.

Different RT constraints have varying impact on timing driven optimization of a design. Most, if not all, of the relative timing constraints must hold for design correctness. Some of these correctness constraints have large margins and easily hold (e.g. the early path has two gates and the late path passes through nine gates). Other correctness constraints drive the overall performance of the design, such as paths that pass through pipeline registers. The quality of results can be improved if timing closure focuses on these performance driven constraints. Therefore, path types as well as their constraints need to be included in the feature set. Path types identify the performance criticality of RT constraints are passed to the ML algorithms.

EDA tools require the timing graph of a circuit to be represented as a directed acyclic graph (DAG). This is critical when relative timing is applied to asynchronous circuits with sequential functions implemented as combinational gates with feedback. Representing the graph as a DAG necessitates utilizing a subset of the full set of RT path constraints. In addition, many included paths are cut in producing the DAG, resulting in timing paths that are subsets of the full RT constraint path. The current implementation only employs a subset of the RT paths for both synthesis and validation, not the full RTC path. Thus, after timing closure as reported in this paper some timing paths may have violations, even though the path segments used for timing closure have all met timing. These are reported in Sec. 6.

Uniformity of the delay in performance driven paths is very important. Common designs which require such uniformity include dependent cycles (e.g. linear pipelines) and signals with large fan out (e.g. driving a large register bank). Uniformity of delay targets can be achieved in this tool by using a common tcl variables for the delay values. Another method for creating uniform delay targets is to use wild cards in the design based on design hierarchy. For example, the presence of the wildcard "*" in the sample constraint in Eqn. 4 indicates applicability of defined delay of 0.2 to all the paths between register clock pin to another register data pin as defined by the start and end point. A similar situation is applicable while extracting path delays using Synopsys

PrimeTime. Thus, it helps in reducing physical design variation due to EDA on the connected paths [4].

$$\texttt{set\_max\_delay 0.2 -from ctlreg0/qreg}*\texttt{/G -to doutl/qreg}*\texttt{/D} \tag{4}$$

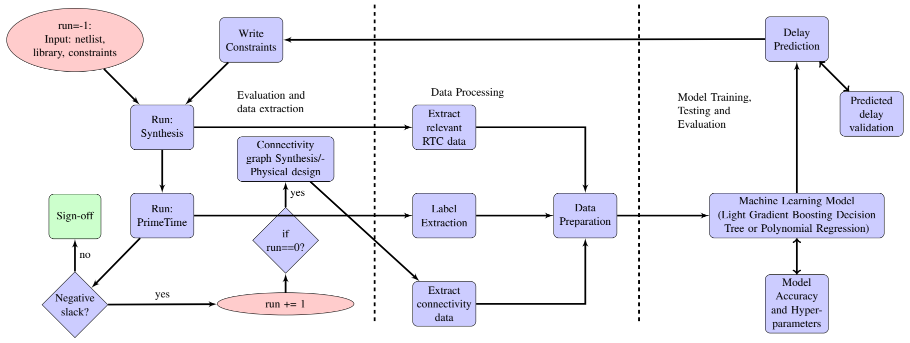
run=-1:
Input: netlist,
library, constraints
Write Constraints
Run: Synthesis
Evaluation and data extraction
Connectivity graph Synthesis/-Physical design
yes
Sign-off
Run: PrimeTime
no
if run==0?
Negative slack?
yes
run += 1
Data Processing
Extract relevant RTC data
Label Extraction
Data Preparation
Extract connectivity data
Model Training, Testing and Evaluation
Delay Prediction
Predicted delay validation
Machine Learning Model (Light Gradient Boosting Decision Tree or Polynomial Regression)
Model Accuracy and Hyper-parameters

Fig. 4: Workflow of machine learning based delay evaluation to sign-off timing on a RT design with commercial EDA tools in the inner loop of the implementation.

4 Implementation

The flow diagram in Fig. 4 shows the basic framework of the implementation. The complete timing closure system is powered by small sub-programs to interface with synthesis or PnR tools, to process data, to validate quality of predictions and to extract required data at various stages of the flow using timing tools.

The end goal of the ML workflow is to generate delay target values for the subsets of RT constrained paths while ensuring timing closure of the design with no negative slack. The ML based framework always ensures that delay margin is obeyed between associated minimum and maximum delay constraints during the predicted delay validation step.

The prediction of the model produces targets with the intent of generating a solution with positive slack. Timing closure is obtained when no negative slack exists on any timing path. The program terminates with non-convergence when three subsequent iterations have the exact same values of the negative slack on the exact same relative timing paths. At this moment, no upper bound is added to the number of iterations, however, the time-frame is restricted based on the allowed run-time of a program on a machine.

It is imperative to have a uniquely trained model for each design. The constraints and their paths are unique to each design netlist and iteration. The constraints remain relevant for a design unless there is a change in the netlist. If there is a change in netlist, discarding the old model and restarting timing closure run from the beginning will ensure the timing paths and the model are not outdated.

4.1 Learning

The data set passed to the ML workflow varies depending on the size of the design. Run time to converge timing on a complex design can extend up to many hours. Nearly all the run time is spent in the synthesis and/or PnR runs to reach converged delay values with no negative slack. Therefore our process focuses on minimizing the number of iterations required to reach a converged set of design constraints by learning from both positive slack and negative slack data generated during each iteration. The results of the model are evaluated during subsequent iterations to ensure the model is moving towards convergence in each iteration.

Feature Selection The selected feature set or columns in the data set currently includes the following. (a) Design connectivity data including timing end points for RT constraints such as: start point (pod) and end point (poc), and path type information identifying whether the path is a data path (register-to-register) or control path (controller-to-controller or controller-to-register). (b) Timing constraint data including margin, delay value, delay type (minimum delay or maximum delay), slack (positive or negative), and whether a path is a performance path or a design correctness path.

Certain features are categorical features. These are required to build a relationship between the data and/or prioritize certain paths over others. The cost function is composed of a negative slack value, delay, and margin. The results of the cost function i.e. *target feature (y_i)* is predicted by the model. Eqn. 5 and 6 define the basic estimated delay for maximum and minimum paths, respectively.

$$CF_{max} \text{ (target delay max)} < delay - negslack \tag{5}$$

$$CF_{min} \text{ (target delay min)} > delay + negslack \tag{6}$$

The cost function changes based on the margin values defined for minimum and maximum delay paths. If the margin is not maintained during estimated delay calculation, the difference is calculated and added to the cost function to maintain margin. The maximum delay is decreased (or minimum delay is raised) based on the positive slack on each path. A performance constraint in the form of a maximum delay constraint is present on minimum delay paths of performance type to reduce variability and more tightly constrain cycle times. The estimated maximum delay is raised to maintain the margin between minimum and maximum delay constraint on paths with the same start point and end point, which minimizes energy consumption but allows more variation. The algorithm learns from the data set in order to achieve optimized predictions. The data set increases over time, which leads to improved prediction accuracy. The predicted delay values replace the delay values in the constraints file for next synthesis or PnR run.

Training The training data includes all the extracted features and labeled data. The data set grows with each iteration while learning between subsequent iterations. Data with consistent trends simplifies the training of a machine learning algorithm. That luxury is generally not observed in this application as the identical delay value on a path can return vastly different slack values on different runs due to a change caused on a totally

Algorithm 1 Light Gradient Boosting

```
1: Data: $data_i$
2: current run = i
3: if column_type is ‘object’ then
4:     data encoding = encode($\sum_{k=1}^{i} data_k[column\_name]$)
5: end if
6: $train.data \leftarrow \sum_{k=1}^{i} data_k$
7: $x.train.data \leftarrow train.data-$ [column with cost function delay value]
8: $y.train.data \leftarrow train.data$[cost function delay value column]
9: $test.data \leftarrow data_i$
10: $lgb\_train$ = Create GBDT Model with data set ($x.train.data$, $y.train.data$, $test.data$)
11: $gbm$ = $lgb.train(lgb\_train)$ in iterations
12: if $test.data$ not empty then
13:     $gbm.predict(test.data)$
14: end if
```

different path or different synthesis run. Often such variations are caused by a path becoming an outlier as optimizations on other paths take priority. Such unpredictable changes tend to be itinerant, so by incrementally updating the training data set progress is made towards an accurate and consistent model.

Testing The test data set consists of data with a negative slack. The supervised learning model created from the training data set is used to predict delay for all the failing timing paths. The delay target is updated in the constraints for next iteration. The test data set is also updated between iterations.

4.2 Machine Learning Algorithms

A comparative analysis is performed between the results obtained from gradient boosting decision tree based model and polynomial regression based model. The later implements separate models for minimum and maximum delay paths.

There exists n relative time constraint sub-paths in each design. On iteration i, a path n has delay target tn_i and delay dn_i where $dn_i \geq 0$, and a slack sn_i. In the event of a negative slack ($sn_i < 0$) on path n, maximum delay fixing is prioritized over minimum delay. The cost function estimates the delay tn_i for each constraint path to be $dn_i - sn_i$ if it is a maximum delay path, and $dn_i + sn_i$ if it is minimum delay path. The estimated delay value data set is input into a machine learning algorithm to obtain predicted delay values based on the feature set and relationship built over iterations. In the event of non-convergence, slack on same paths between subsequent iterations is compared. If there is no change in slack for three iterations, the program terminates with non-convergence.

4.3 Light Gradient Boosting (LightGBM)

The pseudo code of the implementation is in Algorithm 1. The features were extracted and an input data set was created using commercial EDA tools. The data set generated for each iteration i with negative slack present on one or more delay paths is given by $data_i$. There exists categorical features in the data set, so data encoding is performed to convert them to numerical values. Training data $train.data$ and test data $test.data$ are separated. A check is added to exclude paths with negative slack that are being tested during an iteration from the training data set of that iteration. The created model is used to predict new delay values that would fit the design better. The process is repeated until the design converges with no negative slack.

Algorithm 2 Polynomial Regression Model

```
1: Data: $data_i$
2: current run = i
3: $train.data \leftarrow \sum_{k=1}^{i} data_k$
4: $test.data \leftarrow data_i$
5: $train.min \leftarrow \sum_{k=1}^{i} data_k$, when column ‘delay_type’ is ‘min’
6: $train.max \leftarrow \sum_{k=1}^{i} data_k$, when column ‘delay_type’ is ‘max’
7: $test.data\_pos \leftarrow test.data$ , when column ‘negslack’ value is 0
8: $test.data\_neg \leftarrow test.data$, when column ‘negslack’ value is less than 0
9: $test.min \leftarrow test.data\_neg$, when column ‘delay_type’ is ‘min’
10: $test.max \leftarrow test.data\_neg$, when column ‘delay_type’ is ‘max’
11: $x.train.min.data \leftarrow train.min -$[column with cost function delay value]
12: $x.train.max.data \leftarrow train.max -$[column with cost function delay value]
13: $y.train.min.data \leftarrow train.min$[cost function delay value column]
14: $y.train.max.data \leftarrow train.max$[cost function delay value column]
15: $model.min$ = create pipeline with polynomial features ($x.train.min.data$, $y.train.min.data$)
16: $model.max$ = create pipeline with polynomial features ($x.train.max.data$, $y.train.max.data$)
17: $min\_pred = model.min.predict(test.min)$
18: $max\_pred = model.max.predict(test.max)$
19: $pred$ = append($min\_pred$, $max\_pred$)
```

4.4 Polynomial Regression Model

The polynomial model was chosen over the linear regression package provided by the scikit-learn package [20] to better fit the data outliers. A linear regression model looks for a linear relationship between the features, which does not exists in our case. No linear model would serve the problem appropriately since this is a non-convex optimization problem. A polynomial regression establishes a non linear relationship between the feature set to obtain the predicted delay value. On a large complex design, too high of a model degree will result in memory out errors. We found that a polynomial of the order of four is works well to predict the delay target values from the feature set. Pseudo-code for this implementation is shown in Algorithm 2.

5 Designs

The machine learning based timing closure (MLTC) tool and HBTC have been applied to converge timing on a number of designs implemented in the 40 nm technology node. Designs with varying complexity are used to test the convergence engine: linear pipeline, watchdog timer, wakeup timer, timer, and FFT-64 design. The designs implemented here are hierarchical designs containing soft macros of asynchronous linear controllers and register banks, as well as free form RT constraints in the counters.

The pipeline design is a retimed 10-stage linear pipeline design implementing $dout = 2x^2 + 2x + 2$. This design can be scaled to arbitrary pipeline depth, but here contains 179 unique RT constraint sub-paths.

The general, wakeup, and watchdog timers are small designs contain 121, 86, and 108 RTCs respectively. They contain 16 bit programmable timers including multiple clocked and asynchronous time sampled inputs and prescale dividers. These designs all show a power reduction of $30\times$ or more in common operational modes compared to a clocked design.

The FFT design is a 32-bit, 64-point multirate fast Fourier transform (FFT) design that is hierarchically decomposed at the top level to operate at multiple frequencies [21]. This is a large design that contains over 50,000 RTCs, that takes approximately 40 minutes to synthesize with Design Compiler.

6 Results

Synopsys Design Compiler is used for synthesis. Synopsys PrimeTime for timing analysis, power analysis, and slack output. Modelsim is used for simulation. The results are compared for power, performance, and simulation errors that occur after the timing closure run due to the partial paths used in synthesis and timing analysis. The same starting point is employed for all the designs during the two machine learning models by assigning zero maximum delay targets.

The maximum delay targets for all of the designs in Table 1 and 2 are set to zero to start the timing closure optimization. This allows the ML algorithms to optimize the designs targeting maximum achievable frequency. Table 1 shows results obtained from light gradient boosting decision tree model, and Table 2 shows results for polynomial regression model. Each iteration includes synthesis/Primetime/ML runs and one PnR run. The number of iterations are identified.

Table 1: Results with light gradient boosting model (lgbm)

Designs	**No. of iterations**	**sim. errors**	**avg cycle time (ns)**	**Power (uW)**	**Energy (fJ)**	$e\tau^2$
Pipeline	5	0	0.945	0.27	0.26	0.023
Wakeup	8	0	0.548	51.23	28.07	0.843
Watchdog	13	0	1.247	46.72	58.26	9.06
Timer	22	6	1.903	57.71	106.02	38.39
FFT-64	4	0	1.692	17,300	$29.27e^3$	$8.38e^3$

For small designs like the wakeup and watchdog controllers, the overall run-time was 20 minutes with lgbm and convergence with prm was much quicker. The largest

Table 2: Results with polynomial regression model (prm)

Designs	**No. of iterations**	**sim. errors**	**avg cycle time (ns)**	**Power (uW)**	**Energy (pJ)**	$e\tau^2$
Pipeline	1	0	0.945	0.27	0.26	0.023
Wakeup	6	0	0.548	50.39	27.61	0.830
Watchdog	7	1	1.247	44.31	55.25	8.59
Timer	7	6	1.903	57.44	109.31	39.59
FFT-64	6	0	1.690	18,400	$31.10e^3$	$8.88e^3$

design example, FFT-64, took 16 hours to converge with lgbm implementation whereas prm took 23 hours to converge on the same design. Most of the run-time is spent running commercial EDA tool, whereas machine learning data-set generation, feature and label extraction, model training and testing takes a few seconds to a few minutes based on the complexity of the design.

Table 3: Comparison between LGBM and PRM.

Designs	**No. of iterations**	**Energy**	$e\tau^2$
Pipeline	5.00	1.00	1.00
Wakeup	1.33	1.02	1.01
Watchdog	1.86	1.05	1.05
Timer	3.14	0.97	0.97
FFT-64	0.67	0.94	0.94

6.1 Comparative Analysis

LGBM vs PRM A comparison of results from the polynomial regression model (prm) and light gradient boosting model (lgbm) learning algorithms is presented in Table 3. Both designs produce results with nearly identical cycle times for the small and large designs. They produce equivalent results for the simple pipeline design with prm converging in one iteration. The polynomial regression model performs better in both run time and energy efficiency on the small designs. The opposite is true for large designs, where lgbm model converges more quickly with better energy results. The prm model is better trained with positive slack data in comparison to including the complete data set for light gradient boosting model. Also, the train data set needs to be substantial to run light gradient boosting model which is achieved in subsequent iterations for a small design. This also makes lgbm model a better choice for complex relative timed designs.

MLTC vs HBTC Table 4 compares the results obtained from two machine learning models: light gradient boosting (lgbm) and polynomial regression model (prm) with respect to the heuristics based timing closure (HBTC) engine [5]. Both the machine learning algorithms perform better in comparison to HBTC based engine in terms of number of iterations (shown in Fig. 5), power, energy and performance on the smaller designs. The HBTC algorithm produces improved results over the lgbm model, but requires 4.5× more run time.

Table 4: Power and performance contrast between ML and HBTC results.

Design	**Cycle Time (lgbm)**	**Cycle Time (prm)**	**Cycle Time (hbtc)**	**Energy (fJ) (lgbm)**	**Energy (fJ) (prm)**	**Energy (fJ) (hbtc)**
Wakeup	0.548	0.548	0.550	28.07	27.61	28.80
Watchdog	1.247	1.247	1.238	58.26	55.25	64.87
Timer	1.903	1.903	2.115	109.82	109.31	85.55
FFT-64	1.692	1.690	1.688	$29.27e^3$	$31.10e^3$	$20.93e^3$

7 Conclusion

Machine learning is employed to implement gradient descent algorithms in combination with boosting to solve the non-convex timing closure problem for relative timed circuits. These algorithms drive synthesis or place and route to produce a full timing closed design. The designs are started in a state where maximum delay values are set to zero, and convergence is reached when there are no negative slacks in the designs.

Various ML algorithms were investigated and the results were compared to heuristics based timing closure (HBTC) method. The better version of gradient boosting algorithms in the form of light gradient boosting algorithm is implemented which is faster and works well with the desired encoded features. The polynomial regression model was also implemented. Timing data generated during synthesis or place and route is incorporated while implementing the two machine learning models.

The algorithms were compared using a set of five designs, ranging from a simple linear pipeline using retiming to solve a polynomial function, to a complex 64-point fast Fourier transform function. Both machine learning algorithms converged to produce

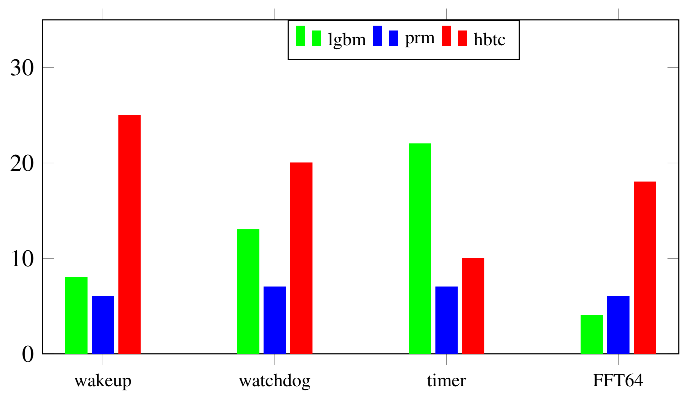

Fig. 5: Number of iterations, ML vs HBTC

results of similar quality in terms of circuit cycle time. The *lgbm* models use many weak learners like the decision trees shown in Fig. 3, so, the learning process is slower than polynomial regression which is mainly numerical based relationship to learn and predict. This makes LGBM model best suited for complex designs where one path delay/slack is intertwined with other paths and polynomial regression is best suited for smaller/simpler designs.

The polynomial regression model showed 5.4% better energy results on the small design, whereas the light gradient boosting model showed 6% better energy efficiency on the large 64 point FFT design. The machine learning algorithms served as a solution to build a relative timing closure tool when other gradient descent based approaches failed. The contrast with HBTC approach shows scope of improvement in ML approaches by making them power aware. At the same time, applying ML based timing closure on a small hierarchical blocks serves to improve the quality of top level design. Finally, by implementing learning based timing closure, we could minimize the number of iterations required to generate a timing closed relative timed design.

References

1. R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 8, pp. 860–874, Aug 1989.
2. K. S. Stevens, R. Ginosar, and S. Rotem, "Relative Timing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 11, pp. 129–140, Feb. 2003.
3. J. V. Manoranjan and K. S. Stevens, "Qualifying Relative Timing Constraints for Asynchronous Circuits," in *International Symposium on Asynchronous Circuits and Systems*, May 2016, pp. 91–98.
4. T. Sharma and K. S. Stevens, "Physical Design Variation in Relative Timed Asynchronous Circuits," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2017, pp. 278–283.
5. Tannu Sharma and Kenneth S. Stevens, "Automatic Timing Closure for Relative Timed Designs," in *28th IFIP International Conference on Very Large Scale Integration*. IEEE, Oct. 2020.
6. I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.
7. W. Bao, P. Cao, H. Cai, and A. Bu, "A learning-based timing prediction framework for wide supply voltage design," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '20, NY, USA, 2020, p. 309–314.
8. I. Turtletaub, G. Li, M. Ibrahim, and P. Franzon, *Application of Quantum Machine Learning to VLSI Placement*, 2020, p. 61–66.
9. N. Kapre, B. Chandrashekaran, H. Ng, and K. Teo, "Driving timing convergence of fpga designs through machine learning and cloud computing," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2015, pp. 119–126.
10. A. B. Kahng, "Machine learning applications in physical design: Recent results and directions," in *Proceedings of the 2018 International Symposium on Physical Design*, ser. ISPD '18, 2018, pp. 68–73.
11. K. Airani and R. Guttal, "A machine learning framework for register placement optimization in digital circuit design," *CoRR*, vol. abs/1801.02620, 2018.

12. Q. Yanghua, H. Ng, and N. Kapre, “Boosting convergence of timing closure using feature selection in a learning-driven approach,” in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–9.
13. I. A. M. Elfadel, D. S. Boning, and X. Li, *Machine Learning in VLSI Computer-Aided Design*. Springer International Publishing, 2019.
14. P. A. Beerel and M. Pedram, “Opportunities for machine learning in electronic design automation,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.
15. G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS’17, 2017.
16. “Welcome to LightGBM’s documentation! — LightGBM 2.2.4 documentation.”
17. “A novel cryptocurrency price trend forecasting model based on lightgbm,” *Finance Research Letters*, 2018.
18. “Light GBM vs XGBOOST: Which algorithm takes the crown.”
19. L. Huang, J. Jia, B. Yu, B. gon Chun, P. Maniatis, and M. Naik, “Predicting execution time of computer programs using sparse polynomial regression,” in *Advances in Neural Information Processing Systems 23*.
20. “sklearn.preprocessing.PolynomialFeatures — scikit-learn 0.21.3 documentation.”
21. W. Lee, V. S. Vij, A. R. Thatcher, and K. S. Stevens, “Design of Low Energy, High Performance Synchronous and Asynchronous 64-Point FFT,” in *Design, Automation and Test in Europe (DATE)*. IEEE, Mar 2013, pp. 242–247.