Using Machine Learning to Predict Operating Frequency During Placement in FPGA Designs

M. Fathi, T. Martin, G. Grewal and S. Areibi
Faculty of Engineering and Computer Science
University of Guelph, Ontario, Canada
Email: {mfathiri, tmarti14, ggrewal, sareibi}@uoguelph.ca

***Abstract*—Circuit placement is an NP-hard problem and is considered to be one of the most challenging steps in the FPGA design flow. The goal of this paper is to explore how machine-learning regression models can be used during placement to predict the maximum frequency of operation. Each model uses static features from the circuit netlist, and dynamic features from the current placement, as input. Results obtained using standard benchmarks indicate that ensemble based machine learning models are capable of accurately predicting the maximum frequency of operation with an average error of 1.72%.**

***Index Terms*—Machine Learning, Max Frequency, FPGAs**

I. Introduction

Short design cycles are one of the primary advantages of Field Programmable Gate Arrays (FPGAs). However, as FPGAs increase in size and complexity, and applications become larger, this advantage is under stress due to the increasing amount of time required to achieve timing closure[1]. In the FPGA design flow, *placement* and *routing* consume the most amount of time in the process. After logic synthesis and technology mapping, placement assigns cells in the circuit's netlist to legal locations on the FPGA, while optimizing objectives such as routability, wirelength and timing.

The goal of timing-driven placement is to minimize interconnect delay for the most timing-critical paths with the aim of achieving a specific clock frequency. The final clock frequency of a design, however, cannot be known until the routing step decides which routing resources on the FPGA to use to connect the cells. Routing solutions that fail to achieve timing closure may require many iterations of place-and-route at a very high computational cost.

Due to their NP-hard complexity, placement and routing are traditionally treated as separate problems. During timing-driven placement, Static Timing Analysis (STA) is performed to identify timing-critical paths, thus loosely coupling the placement and routing steps. STA, however, has its own drawbacks. First, as the routing resources on the FPGA are fixed, large designs that require large numbers of resources must spread out over the FPGA fabric to access the resources they need. This has the effect of reducing the accuracy of the delay model, as delay becomes dominated by (harder to assess) interconnect rather than (easier to assess) logic. Second, the routing delays used in STA do not, typically, take congestion into account. Congestion can have a significant effect on path delay, as the router must search for alternative paths in regions of high congestion. This can result in the placer and router identifying different critical paths. Third, STA does not, typically, take the behavior of the router into account. This, also, can lead to the placer optimizing different critical paths from the router. Fourth, modern FPGAs are heterogeneous and contain regions of special logic blocks, like DSPs and BRAMs. These regions often give rise to congestion due to high demand and limited routing resource availability. Fifth, STA is very expensive to perform during placement and, therefore, is only performed periodically. Frequently, the end result of all of this is a placement solution that when routed fails to meet the target clock frequency for the design.

In this paper, we present six Machine-Learning (ML) models for quickly and accurately predicting the maximum clock frequency (F_{max}) of a design at any point in the placement process. Each model is trained on a large number of place-and-route solutions so that it is able to learn how the location of cells on the FPGA, the use of special logic blocks, the presence of local and global congestion, the routing resources used to connect cells, and the router's behavior combine to determine the final clock frequency that is achieved for a design. The ML models are not intended to replace STA, but to complement it. Each model can be used in a placement engine as a probe to accurately and efficiently estimate post-routing clock frequency. This information can then be combined with information provided by STA to help the placement engine make better decisions during placement to meet a specific clock frequency.

The remainder of this paper is organized as follows. Related work and relevant background are briefly presented in Section II. Section III introduces the ML framework used to create the ML models to predict F_{max}. Results and conclusions are given in Sec. IV and Sec. V, respectively.

II. Preliminaries

A. Related Work

The authors in [1] present an ML framework for predicting the optimal performance and resource utilization of a High Level Synthesis (HLS) design. The model achieves an accuracy of 95%. The authors in [2] describe an ML framework for modeling signal-processing applications on FPGAs. The model predicts clock frequency, resource utilization, and latency with an average error of 3.6%.

[1]Timing closure is the ability to satisfy all setup and hold time constraints, and to achieve a specific target clock frequency.

B. FTPlace Flow

In this paper, we use both Xilinx Vivado and FTPlace – a timing-driven, analytic placement tool [3]. An overview of FTPlace is given in Fig. 1. Step 1 is responsible for optimizing

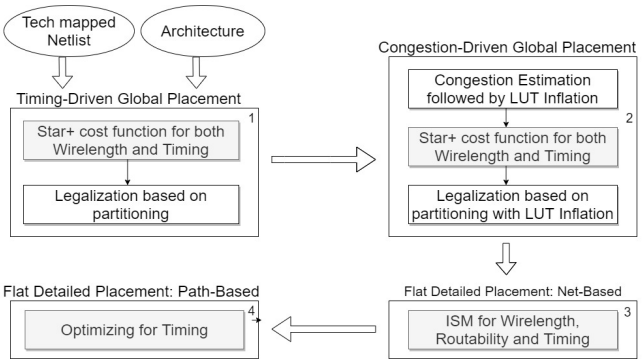

Fig. 1: FTPlace Flow [3]

both wirelength and timing. The global placement algorithm used in Step 1 (and step 2) uses the Star+ net model [3] and an appropriate delay model when optimizing wirelength and timing. Step 2 aims to identify and remove any congestion identified in the placement produced in step 1. Steps 3 and 4 are responsible for performing detailed placement to further improve wirelength, timing, and routability through local optimization.

III. Methodology

Figure 2 illustrates the proposed framework for predicting F_{max} for a given placement. The details of the framework are described in the subsections that follow.

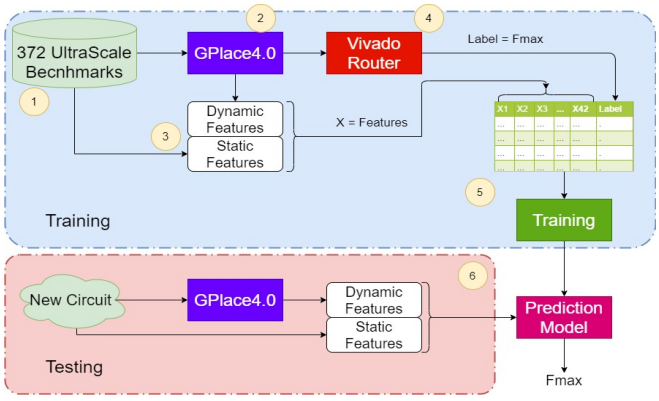

Fig. 2: F_{max} Prediction Framework

A. Data Preparation

Accurately predicting F_{max} for a placement requires having access to a large quantity of high quality data. With this in mind, we use the 372 benchmarks available from [4]. These benchmarks target a modern Xilinx Ultrascale VU095 FPGA device. Table I shows the ranges of key circuit components for all 372 benchmarks. As shown in Fig. 2, each benchmark is placed using FTPlace [3]. Legal placement solutions are extracted throughout each step of FTPlace, so that the resulting prediction model can be used during placement and not simply at the end of placement. Each placement is then routed using Xilinx Vivado 2015 to determine its maximum clock frequency. These frequencies are the labels for each placement (i.e., record). In total, 3618 records (i.e., placements and their maximum clock frequency) are generated, with 70% of the records randomly assigned to a training set, 15% to a validation set for hyper-parameter tuning and feature selection, and 15% to a testing set.

TABLE I: Range of Key Circuit Features

#LUTs	#FFs	#BRAMs	#DSPs	#CSets	#IOs	R.E.
44K-518K	52K - 630K	0 - 1035	0 - 620	11 - 2684	150-600	0.4 - 0.8

B. Feature Engineering

Another important step to accurately predicting F_{max} is determining the appropriate features to serve as input to the model. For this purpose, we use both static and dynamic features. Static features are extracted once prior to placement directly from the circuit netlist. These features capture information about the numbers and types of components, as well as their logical connectivity. Dynamic features are extracted (possibly multiple times) during placement, and capture information like congestion, timing, and routing demand. A complete list of the features is given in Table II.

- Static Based Features:
  1) **Resource Features** capture the resource usage of the FPGA. Features 1-11 show the number of LUTs with sizes of 2-6, FFs, DSPs, BRAMs, I/O pins, and control sets, respectively.
  2) **Connectivity Features** capture the connectivity of modules in the netlist. Features 12-17 show the percentage of nets with 2, 3, 4, 5-10, 11-30, and 30+ pins. Features 18-19 show the total number of pins and nets in the netlist.
  3) **Longest Path Features** correspond to the longest logical path in the netlist. For each sink node in the netlist, we find all of the source nodes that have a path to it. The length of the path between source and sink is then calculated by assuming unit delays for each module. The longest path for each sink node is then calculated. Features 21 and 22 show the mean and standard deviation of the longest path for all sink nodes. Feature 20 shows the length of the longest path in the netlist. Features 23 and 24 show the percentage of paths within 90-100% and 80-90% of the longest path, respectively.
- Dynamic Based Features:
  1) **Design Congestion Features** characterize the congestion in the placement, and are produced using the MLCONG [5] congestion estimator. Features 25-26

show the mean and standard deviation of congestion among used switches.

2) **Net-Based Features** capture the distribution of the nets in the placement and the routing network. Features 27 and 28 estimate the mean and standard deviation of the routing demand of each net by estimating the wire length of the net within a bounding box [5]. Features 29-32 represent the absolute number of net cuts in a 5*5 and 9*9 window around a site. Features 33 and 34 represent the distribution of pins in the placement, which are related to switch congestion.
3) **FPGA Utilization Features** capture the utilization of the FPGA resources. Features 35 and 36 are the Half-Perimeter Wire Length (HPWL) [5] and the switch utilization of the FPGA, respectively. These features affect the routing stage.
4) **Critical Path Features** capture the main characteristics of the critical path in the placement. The critical path is estimated using static timing analysis [3]. Features 37 and 38 show the length and delay of the critical path, respectively. Features 39 and 40 show the mean and standard deviation of the size of the nets in the critical path. Features 41 and 42 show the mean and standard deviation of the congestion for the switches that are in the critical path. F_{max} is calculated by using the actual critical path during routing.

The previous features were all determined based on intuition and experience. As will be explained later, we perform feature selection to identify the most important.

C. ML Models

The choice of ML algorithm can also affect the performance of any prediction model. Therefore, we explore several ML models including Linear Regression (LR), Decision Trees (DT), and Support Vector Machines (SVM). We also seek for additional performance improvements using ensemble-based models. Ensembles use multiple ML models to further improve accuracy and reduce error. The ensembles used in this work include XGBoost (XGB), Random Forest (RF), and Stacking (SC).

D. Feature Selection and Hyper-parameter Tuning

Feature selection and hyper-parameter tuning are two important steps to improve the performance of an ML model. The features listed in Table II have different correlations with each other and with the label F_{max}. Therefore, we perform feature selection using Sequential Backward Selection (SBS) to determine the best sets of features to use with each of the simple models in Sec. III-C. The input to SBS is the set of all 42 features described in Table II. On each iteration of the SBS algorithm, the feature with the lowest regression performance is removed from the set. This continues until the Mean-Square Error (MSE) of the predictions starts to rise. The output of SBS is a subset of features with the best regression performance. As the subset of best performing features is different for all six ML models, Fig. 3 shows only those features used by at least four models. For hyper-parameter tuning, we perform a grid search to find the best set of hyper-parameters for each model.

TABLE II: List of All Features

Number	Feature Name	Calculation
1-8	Cell Type Counts	$\#Cell_i \in NL$ $i = \{LUT2..6, FF, DSP, BRAM\}$
9	LUTS	$\#LUT2 + \#LUT3 + \#LUT4 + \#LUT5 + \#LUT6$
10	IO	$\#I/Os \in NL$
11	CSET	$\#CSETs \in NL$
12-18	Net Type Counts	$\frac{\#i-pin\ nets \in NL}{\#nets \in NL}$ $i = \{2, 3, 4, [5-10], [11-30], 30+\}$
19	PINS	$\#pins \in NL$
18	NETS	$\#net \in NL$
20	LP	$max(lp_{sinks})$
21	LPMEAN	$mean(lp_{sinks})$
22	LPSTD	$std(lp_{sinks})$
23	LP10	$\frac{\#sinknodes \in NL \mid lp \in [90\%LP, LP]}{\#sinknodes \in NL}$
24	LP20	$\frac{\#sinknodes \in NL \mid lp \in [80\%LP, 90\%LP)}{\#sinknodes \in NL}$
25	CONGMEAN	$mean(Cong_{sw} \mid sw \in S_d)$
26	CONGSTDEV	$std(Cong_{sw} \mid sw \in S_d)$
27	WLPAMEAN	$mean(WLPA_{sw} \mid sw \in S_d)$
28	WLPASTD	$std(WLPA_{sw} \mid sw \in S_d)$
29	NCPR5MEAN	$mean(NCPR_{W_{5\times5}(sw)} \mid sw \in S_d)$
30	NCPR5STD	$stdev(NCPR_{W_{5\times5}(sw)} \mid sw \in S_d)$
31	NCPR9MEAN	$mean(NCPR_{W_{9\times9}(sw)} \mid sw \in S_d)$
32	NCPR9STD	$stdev(NCPR_{W_{9\times9}(sw)} \mid sw \in S_d)$
33	PINMEAN	$mean(pins_{sw} \mid sw \in S_d)$
34	PINSTD	$std(pins_{sw} \mid sw \in S_d)$
35	HPWL	$\sum_{net \in NL} HPWL_{net}$
36	SW	$\#sw \mid sw \in S_d$
37	CPL	$\#module \mid module \in cp$
38	CPD	arrival time of last module in cp
39	NETMEAN	$mean(netsize \mid net \in cp)$
40	NETSTD	$std(netsize \mid net \in cp)$
41	CPCONGMEAN	$mean(Cong_{sw} \mid sw \in S_{cp})$
42	CPCONGSTD	$std(Cong_{sw} \mid sw \in S_{cp})$

Note: S_d represents switches used in the design, $W_{n\times n}(sw)$ represents an $n \times n$ window centered on sw, lp represents the longest path for a sink node, cp represents the critical path, S_{cp} represents the switches used in the critical path, and NL represents the circuit netlist.

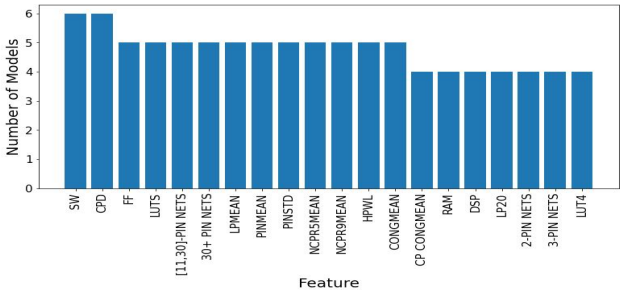

Fig. 3: Feature Importance

IV. Results and Analysis

A. Experimental Setup

The ML models used in this paper were developed using the Scikit learn [6] library in python. The C source code for FTPlace [3] was compiled using the gcc (Red Hat 4.4.7-18) compiler. All experiments were carried out using a Linux machine (CentOS release 6.9) running on an Intel (Xeon CPU E3-1270 V2 @ 3.50GHz) processor.

B. Accuracy Metrics

To evaluate the performance of the six ML models we use three common metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE). All three metrics are computed over the entire testing data set. Since all all three work with average error, lower values are considered better. In particular, with RMSE, larger errors are given higher weight, but when using MAE, all differences are given equal weight. MAPE computes a percentage error for each point to compare the predictability of different scaled data.

C. Results

Table III shows the prediction quality obtained by each of the ML models before performing feature selection and hyper-parameter tuning. Among the three simple models, DT has the best performance and achieves an RMSE, MAE, and MAPE of 3.58, 2.28, and 2.45, respectively. Among the ensembles, performance of XGB, SC, and RF is close, but RF has the best performance achieving an RMSE, MAE, and MAPE of 2.83%, 1.79%, and 1.90%, respectively. Table IV shows the prediction

TABLE III: Initial Prediction Quality

Model	RMSE	MAE	MAPE
LR	5.84	3.98	4.67
SVM	11.93	8.34	9.55
DT	3.58	2.28	2.45
RF	**2.83**	**1.79**	**1.90**
XGB	2.85	1.81	1.94
SC	2.90	1.86	1.96

quality of the ML models after performing feature selection and hyper-parameter tuning. All of the models benefited from hyper-parameter tuning and feature selection. However, XGB now has the best performance with RMSE, MAE and MAPE values of 2.58, 1.72, and 1.72, respectively. Table V shows the

TABLE IV: Final Prediction Quality

Model	RMSE	MAE	MAPE
LR	5.69	3.94	4.58
SVM	9.68	6.79	7.56
DT	3.56	2.23	2.38
RF	2.75	1.73	1.85
XGB	**2.58**	**1.72**	**1.72**
SC	2.90	1.85	1.96

training and inference time by each of the ML models. Among the simple models, LR has the least training and inference time, while among the ensembles RF has the best performance due to its parallel nature. Figure 4 shows the actual and predicted

TABLE V: Training and Inference Times

Model	LR	SVM	DT	RF	XGB	SC
Training Time (s)	0.01	1.41	0.08	0.22	0.85	2.21
Inference Time (s)	0.001	0.16	0.001	0.003	0.006	0.08

frequency (red) of the best (XGB) model for all the samples in the test set sorted from lowest to highest by their actual frequency. It is clear that the model remains accurate across low- to high-speed designs.

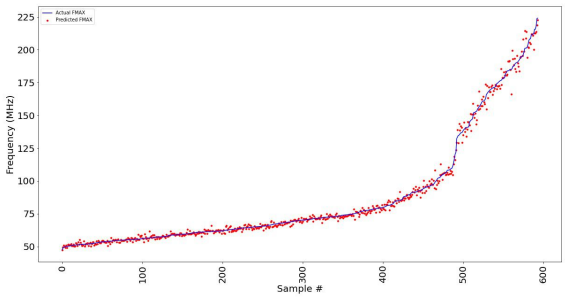

Fig. 4: Prediction Accuracy

Figure 5 shows how the MSE of each ML model is affected by the *size* of the training set. The training sets range in size from 10 to 3000 records. With a training set of size of just 1200, all of the models are within 5% of their final MSE.

Fig. 5: Model Loss versus Training Set Size

V. Conclusions

This paper proposes a set of ML models for accurately predicting the maximum frequency of operation for a placed FPGA design. All of the models are fast to compute, and have a high accuracy across low- to high-frequency designs. Our future work involves exploring how to use these accurate predictions during placement to dynamically guide the placer's optimization strategies.

References

[1] H. Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. Dinakarrao, L. Zhao, A. Sasan, H. Homayoun, and S. Rafatirad, "Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design," in *International Conference on Field Programmable Logic and Applications*, 2019, pp. 397–403.

[2] G. Wang, G. Stitt, H. Lam, and A. George, "Core-Level Modeling and Frequency Prediction for DSP Applications on FPGAs," *Int' Journal of Reconfigurable Computing*, vol. 2015, no. 2, pp. 1–20, April 2015.

[3] T. Martin, D. Maarouf, Z. Abuowaimer, A. Alhyari, G. Grewal, and S. Areibi, "A Flat Timing-Driven Placement Flow for Modern FPGAs," in *DAC Conference*, Las Vegas, USA, June 2019, pp. 1–6.

[4] G. Grewal and S. Areibi, "Guelph FPGA CAD Group: Xilinx Benchmarks," https://fpga.socs.uoguelph.ca/benchmarks/Xilinx.

[5] D. Maarouf, A. Alhyari, Z. Abuowaimer, T. Martin, A. Gunter, G. Grewal, S. Areibi, and A. Vannelli, "A Machine-Learning Based Congestion Estimation for Modern FPGAs," in *IEEE Int'l Conf. on Field Programmable Logic (FPL)*, Dublin, Ireland, August 2018, pp. 427–434.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.