Case for Design-Specific Machine Learning in Timing Closure of FPGA Designs

Que Yanghua
quey0001@e.ntu.edu.sg

Chinnakkannu Adaikkala Raj
adaikkal001@e.ntu.edu.sg

Harnhua Ng[1]
harnhua@plunify.com

Kirvy Teo[1]
kirvy@plunify.com

Nachiket Kapre
nachiket@ieee.org

School of Computer Engineering, Nanyang Technological University, Singapore 639798
[1]Plunify Inc., 67 Ayer Rajah Crescent, Singapore 139950

ABSTRACT

We can achieve reliable timing closure of FPGA designs using machine learning heuristics to generate input parameter settings for FPGA CAD tools. This is enabled by running multiple instances of CAD tool with different sets of these input parameters and logging of resulting timing slack values into a database. We incrementally build this database and run learning routines to develop suitable classifier models that correlate input parameter combinations to resulting slack. As each CAD run in independent, we can trivially parallelize our exploration. The classifier model developed using this approach can help predict whether a given combination of tool parameters will improve the timing score of that particular FPGA design. Through repeated trials and use of cheap cloud computing resources, we are able to reliably improve timing scores for a variety of industrial and academic FPGA designs. We show how to build design-specific classifier models that easily outperform generic models that are trained by combining results across all circuits in a benchmark.

1. INTRODUCTION

With FPGA capacities rising to millions of LUTs per chip, the ambition and complexity of modern FPGA designs is growing proportionally larger. FPGAs today can fit designs requiring millions of LUTs, thousands of hard DSP-blocks, thousands of on-chip Block RAMs, hundreds of IO ports supporting a rich set of IO protocols, and complex wiring requirements. Modern CAD tools have struggled to keep up with this increase in design size and heterogeneity of the underlying FPGA fabric resulting in long compilation cycles for these designs. A typical single run of the CAD tool can take hours to days of compilation time under user-supplied constraints. Developers often iterate through this painful compilation process multiple times hoping to improve timing scores by modifying their RTL through pipelining, logic restructuring and providing suitable constraints to the CAD tool. They can fail and in many cases are prevented from modifying verified RTL source (*e.g.* black-box IP) thereby restricting their freedom.

When RTL modifications are not possible, we can still rely on the tuneability and configurability choices available in the CAD algorithms itself. However, modern CAD tools are a complex series of NP-complete heuristics that are hard to understand and configure properly for a given design. A typical FPGA developer may require years of experience before mastering the art of guiding the CAD tool to produce desired results. These tools are organized as a series of passes such as synthesis, technology mapping, placement, and routing before the bitstream is produced. These key stages are NP-complete problems that are driven by tuneable heuristics. Consequently, they significantly affect solution quality. The exact selection of CAD tool parameters (command-line options, GUI switches, and drop-down boxes) depends heavily on the specifics of the user design and user constraints,

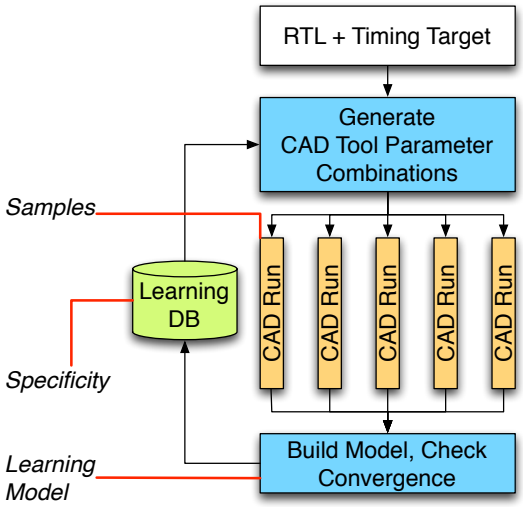

Figure 1: Organization of InTime Flow. Overall goal is to achieve timing closure for a given RTL design. InTime runs the RTL through multiple rounds of parallel CAD runs with different CAD parameter configurations. Based on history of what improved timing, we modify the configurations for subsequent rounds of execution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
FPGA'16, February 21–23, 2016, Monterey, CA, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3856-1/16/02 . . . $15.00.
DOI:http://dx.doi.org/10.1145/2847263.2847336.

and its interaction with the exact device family selected for mapping.

In this study, we use InTime [2, 3], a plugin for CAD tools, that can efficiently select these parameters for each combination of RTL design, FPGA device, and timing constraint combination. InTime helps designs achieve timing closure by running multiple CAD runs in parallel with different parameter selections and refines these selections per design through machine learning. This addresses the long runtimes of the CAD tools inherent in the edit-compile-debug loop of a typical FPGA design, as well as select input CAD parameters for the CAD tool in an automated fashion. In this paper, we focus on the selection of appropriate machine learning algorithms and techniques to identify and tune the most promising solution. The goal is to deliver timing closure with high classifier accuracy and consequently fewer iterations to timing closure. The principle of operation of InTime has already been established in prior work [2, 3].

The key contributions of this report include:

- Development of plugins for InTime to evaluate the impact of design specificity on the overall solution of our learning algorithms.
- Quantification and characterization of design-specific learning routines across various real-world open-source benchmarks.

2. INTIME

2.1 Execution Flow

As described in [2, 3], InTime is a plugin for FPGA CAD tools from Altera and Xilinx that helps the developer select a suitable combination of CAD tool parameters to achieve timing closure. Modern FPGA CAD tools employ tuneable heuristics that export hundreds of parameters. Some of these are boolean (on/off) parameters, some offer discrete choices, while others are continuous. In all these cases, selecting the combination of assignments to these parameters to guide timing closure is hard and usually handled through experience or trial-and-error. For Altera Quartus 14.1 CAD tool with 80 selected boolean parameters, a trivial brute-force exploration of all possible combinations will take 2^{80} runs of the CAD tool which is clearly infeasible using contemporary computing technology. Instead, InTime *learns* these insights through automated parallel trials of far fewer combinations and accumulating wisdom through machine learning. As shown in Fig. 1, InTime is organized as an iterative computation broken down into a series of parallel CAD *runs*. Each *iteration* (or *round*) is an opportunity to acquire data for the learning database. Typically, we need 30 runs in each *round* to acquire sufficient data points to drive the learning algorithms. In [3, 2], InTime used a Naive Bayesian learning framework to classify the timing results and drive the learning process (maximum observed accuracy of ≈70%). While these results were promising, our work differs from InTime by clearly exploring design specificity.

We evaluate the classifier accuracy gap between general models that are trained in a single-shot with all design data vs. a design-specific learning flow where each circuit trains its own classifier. FPGA vendor DSE (design space exploration) tools typically use the generic approach to greatly save on the compute time and compute costs required by running a pre-calibrated set of CAD parameter combinations. More importantly, we also consider the scenario where different machine learning routines may be suitable for different kinds of benchmarks. This requires a further meta analysis of how to select the most appropriate learning approach for a given circuit benchmark.

2.2 Formal Model

The key idea in InTime is to track which CAD parameter combinations improve timing slack and which combinations make timing worse over a reference baseline. Thus, we can formulate a supervised learning approach to develop a classifier model that can determine if a given combination of CAD parameter assignments will help the design converge towards timing closure. The combinations themselves can be generated statistically. We represent this formally in Figure 2 where x_{ij} are the boolean parameters for each CAD parameter i and y_j is a timing slack result for a given CAD execution j.

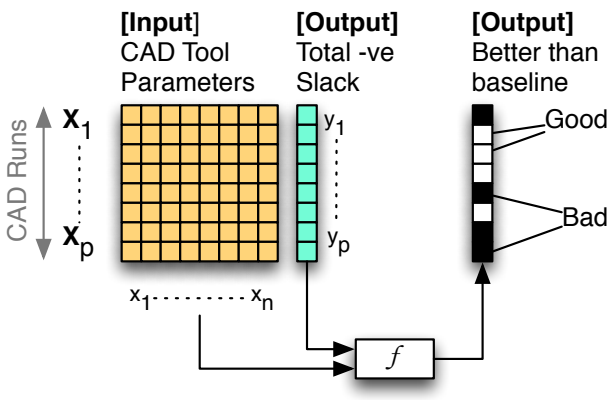

Figure 2: Data Structures used to build classifier model. We tabulate all combinations of CAD parameters tested in various runs, and record the resulting TNS (Total Negative Slack). This is used to drive a supervised learning routine that classifies timing scores better than a reference baseline as `GOOD` or `BAD`. This then trains a model which is used to generate better candidates for subsequent CAD runs.

3. METHODOLOGY

In this section, we introduce our set of benchmark circuits used to evaluate the various learning algorithms, and the system setup required to run these experiments.

3.1 R Packages

We wrote machine learning routines as plugins for InTime using `R` [5] and appropriate `R` libraries and packages. The interface supplies a CSV-formatted dataset with input columns containing the boolean values of the various input CAD parameters and an output column containing the resulting timing score. We make extensive use of *caret* [1] to train our models. It provides a unified interface to use multiple machine learning algorithms. The runtime of running these algorithms on the data are minuscule (few seconds) compared to the much longer runtimes of the CAD algorithms.

3.2 Benchmarks

In Table 1, we list the key characteristics of the FPGA benchmarks used in this study that are taken from OpenCores as well as industrial designs. These benchmarks were compiled using Quartus 14.1 and mapped to the Cyclone devices using the free Web Edition licenses. Certain industrial designs also targeted other Altera devices and tool versions. These benchmarks occupy a range of sizes and the operating frequencies also cover a spectrum of values thereby stress-testing our toolflow with design-specific requirements and constraints. In particular, the `viterbi` benchmark has a high TNS score and highest number of failing paths and is painful for timing closure. Our framework also works with Xilinx ISE and Vivado flows and the resulting mapping costs are listed in Table 2.

Table 1: Opencore Benchmarks (Altera)

Bench.	FFs	LUTs	P&R mins.	Freq. MHz	TNS	Fail/Tot. Paths
Open-Cores Examples						
aes	6K	11K	22	500	0.2	5/5.7K
switch	0.5K	2.6K	15	250	0.7	14/2.3K
vga	0.7K	1K	12	400	1.7	17/262
viterbi	1.6K	4K	20	285	22.8	300/6.6K
xge	1672	2.5K	17	333	10.8	79/1.7K
bitcoin	14K	22.3K	28	78	2.8	10/96K
Industrial Examples						
SOC	4.9K	3.5K	5	150	0.6	10/42K
flow	20.7K	21.8K	12	320	5.9	47/87K
vip	62.3K	80.1K	58	150	0.6	15/435K
eight	3.3K	3.5K	2	174	1.02	10/16K

Table 2: Opencore Benchmarks (Xilinx)

Bench.	FFs	LUTs	P&R mins.	Freq. MHz	TNS	Fail/Tot. Paths
aes	6K	11K	0	500	1219	4630 / 5775
switch	800	2.4K	0	200	42.3	57 / 2190
vga	966	106	0	400	108	128 / 290
viterbi	3.5K	4K	0	333	5081	4399 / 7299
xge	1.6K	1.6K	0	250	50	229 / 1778

3.3 Compute Resources

We ran all our experiments on the Google Compute Engine [4]. We configured an *instance group* template using `n1-standard-2` CPU configuration with 2 virtual CPUs (Intel Xeon E5s) and 7.5G RAM each. We chose this configuration over the dense 32-CPU configurations to make optimum use of the free web-edition licenses and to ensure sufficient RAM state is available for the CAD tool executions. We also enabled the Google `auto-scaling` feature to trigger the launch of parallel VM instances (up to 10 machines launched in parallel) when the CPU utilization threshold got over 65%. This allowed us to keep costs low and only spawn machines are needed during the benchmarking process. The cumulative costs of using the Google machines for our entire experiment set for over a fortnight was under ≈500$ USD[1]. This means for a single design we roughly needed a couple of days of cloud compute time which compares favorably against an RTL engineer struggling to manually deliver timing closure.

[1]summer 2015 prices, not considering InTime licensing costs.

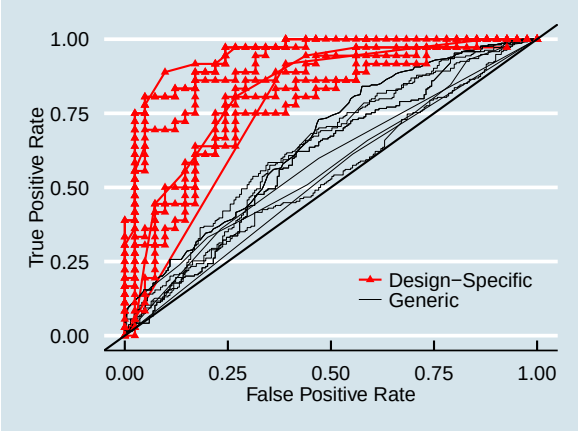

Figure 3: Comparing ROC [6] characteristics of various Machine Learning routines for the `aes` benchmark. Each curve is a particular Machine Learning algorithm. Even the worst-performing Design-Specific model does better than the best-performing Generic model.

4. RESULTS

To evaluate the effectiveness of the various machine learning algorithms, we measure several metrics: (1) Prediction Accuracy, (2) F1 score or FMeasure, and (3) ROC [6] (Receiver Operating Characteristics) plots. To understand these metrics mathematically, we define TP as True Positives, TN as True Negatives, FP as False Positives and FN as False Negatives in our prediction set. Ideally, we want high TP and TN and low values for FP and FN. These are defined based on a classification threshold and can be thought of as a tag or attribute assigned to each row in Figure 2 of the testing set. High prediction accuracy $A = \frac{TP+TN}{TP+TN+FP+FN}$ is useful as it helps us correctly determine how a given combination of FPGA CAD parameters affects timing slack. However, this is not sufficient as we are ultimately interested in generating good candidates that are positively correlated with reducing timing slack. Additionally, false positive rate (FPR) helps us determine if we are making judicious use of our parallel compute resources during the exploration phase by avoiding wasted compute on false positives. It should be noted that we do want certain bad combinations (FP combinations) to teach us what kinds of combinations to avoid in the future. Even a failed CAD run is a teachable moment for InTime flow. We also calculate the FMeasure that is a harmonic mean of precision and recall metrics $F = \frac{2 \cdot P \cdot R}{P+R}$ that provides a unified view of the classifier effectiveness. Precision is the fraction of positive predictions that are relevant $P = \frac{TP}{TP+FP}$ while Recall is the fraction of relevant predictions that are positive $R = \frac{TP}{TP+FN}$. A great visual tool for evaluating the effectiveness of the machine learning algorithm is the ROC (Receiver Operating Characteristics) curve. This provides a complete picture of the tradeoffs between accuracy of the prediction and wasted time on fruitless CAD runs.

Table 3: Features Ranked by Chi-Squared (χ^2) for *aes*.

Rank	Design-Specific	Generic Model
1	Optimize_Ioc_Register_ Placement_For_Timing	Remove_Redundant_ Logic_Cells
2	Physical_Synthesis_ Register_Retiming	Remove_Duplicate_ Registers
3	State_Machine_Proces.	Auto_Ram_Recog.
4	Physical_Synthesis_ Map_Logic_To_ Memory_For_Area	Not_Gate_Push_Back
5	Auto_Rom_Recognition	Physical_Synthesis_ Register_Duplication
6	Synth_Timing_ Driven_Synthesis	Allow_Synch_Ctrl_Usg.
7	Extract_Vhdl_ State_Machines	Auto_Resource_Shar.
8	Dsp_Block_Balancing	Physical_Synthesis_Eff.
9	Fitter_Aggressive_ Routability_Optimiz.	Allow_Any_Ram_Size_ For_Recognition
10	Cycloneii_Optimiz._ Technique	Optimize_Timing

One may be tempted to consider building a generic global model for timing score in a manner that is not specific to a particular design. FPGA vendors are likely to prefer a single design-agnostic model that they can train using their customer benchmarks in-house and ship a single model with their CAD tools to all customers. A key benefit of such an approach is the ability to construct such a model offline across a wide range of benchmarks without resorting to an online learning-based incremental approach advocated in this paper. Additionally, the model can operate in feed-forward manner and execute a set of canned strategies (CAD parameter mixes) based on some design criteria. In contrast, a design-specific model is tailored to each individual design and involves an online learning phase with feedback as described earlier in Section 2.

4.1 ROC for Design-Specific Models

In Fig. 3, we show the ROC plot for the various machine learning routines applied to the `aes` benchmark. Here, we observe that the cluster of design-specific models offer higher accuracy at the expense of higher false positives. Across other benchmarks as well, the prediction accuracy of the general model hovers around 50–55% which is no worse than a random coin toss. It is interesting to note that even the worst-fitted model built from the design-specific scenario is still better than the best-case generic model. There is a clear case for constructing and developing the correct model tailored for each individual design.

4.2 CAD Parameter Ranking

Next, we rank the most important CAD tool parameters with the χ^2 metric in Table 3. This ranking is computed by identifying the CAD tool parameters when modified cause the most impact on the final TNS result. Here, again, we see that the top-10 features for the `aes` benchmark have nothing in common with the top ranked features when considering the generic model that ignores the effect of design-specific preferences. While this is admittedly an extreme case scenario, it highlights the need for a design-specific model yet again. The kind of CAD parameter that matters for a given design is also related to the underlying architecture when we see the Cyclone-II specific optimizations becoming prominent. In general, we expect the combination of design, FPGA architecture and possibly even the CAD tool version having a joint influence on these rankings. This implies we need our models to be built in a manner that is specific to the FPGA architecture and CAD tools version in addition to design alone.

5. CONCLUSIONS

Machine learning algorithms can be configured to deliver timing closure for digital designs in the presence of rising complexity and noise in modern FPGA CAD toolflows. To achieve these goals, we use cheap, parallel cloud computing resources to run multiple instances of the CAD tools guided by these machine learning routines. Across a range of benchmarks, we show that design-specific learning routines outperform generic models. Overall, we observe that InTime works well when the timing constraints are realistic and we observed a few cases where none of the machine learning algorithms delivered useful performance.

6. REFERENCES

[1] M. K. C. from Jed Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, the R Core Team, M. Benesty, R. Lescarbeau, A. Ziem, and L. Scrucca. *caret: Classification and Regression Training*, 2015. R package version 6.0-52.

[2] N. Kapre, B. Chandrashekaran, H. Ng, and K. Teo. Driving timing convergence of fpga designs through machine learning and cloud computing. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 119–126, May 2015.

[3] N. Kapre, H. Ng, K. Teo, and J. Naude. Intime: A machine learning approach for efficient selection of fpga cad tool parameters. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 23–26, New York, NY, USA, 2015. ACM.

[4] S. Krishnan and J. L. U. Gonzalez. Google compute engine. In *Building Your Next Big Thing with Google Cloud Platform*, pages 53–81. Springer, 2015.

[5] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[6] T. Sing, O. Sander, N. Beerenwinkel, and T. Lengauer. Rocr: visualizing classifier performance in r. *Bioinformatics*, 21(20):7881, 2005.