UNDERSTANDING DESIGN SPACE EXPLORATION OF FPGAS FOR EFFICIENT ACCELERATED CORE PROCESSING

by

Sara Bondi Ogburn

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
the Requirements for the Degree
of
Master of Science
Electrical and Computer Engineering

Committee:

_____	Dr. Houman Homayoun, Thesis Director
_____	Dr. Avesta Sasan, Committee Member
_____	Dr. Setareh Rafatirad, Committee Member
_____	Dr. Monson H. Hayes, Chairman, Department of Electrical and Computer Engineering
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Spring 2019
George Mason University
Fairfax, VA

Understanding Design Space Exploration of FPGAs for Efficient Accelerated Core Processing

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Sara Bondi Ogburn
Bachelor of Science
George Mason University, 2014

Director: Dr. Houman Homayoun, Professor
Department of Electrical and Computer Engineering

Spring 2019
George Mason University
Fairfax, VA

Dedication

I would like to dedicate this thesis to my loving husband Rob, without whose constant encouragement and support this work would not be possible. To my parents, who have steadfastly believed in me and my abilities throughout the graduate school process. To my older siblings Kathryn and Steven, for their reassurance and uncanny ability to keep me grounded. And to all my friends and family, for their love and support throughout the challenges of graduate school. Finally, to my cats Miranda and Jack, and their unwavering dedication to keeping my workspace occupied, for better or for worse.

Acknowledgments

I would like to sincerely thank my thesis advisor, Dr. Houman Homayoun, for his encouragement and dedication to seeing my research through. Without his thoughtful support and insightful guidance, the research work I have accomplished in my graduate career certainly would not be possible. His optimistic outlook and great patience have been tremendous. I would also like to thank the rest of my thesis committee, for their comments and support of this work. And lastly I would like to thank all the graduate students I have had the wonderful opportunity to work with: Hosein Mohammadi Makrani, Maria Malik, and Farnoud Farahmand, as well as many others.

Table of Contents

List of Tables

List of Figures

Abstract

UNDERSTANDING DESIGN SPACE EXPLORATION OF FPGAS FOR EFFICIENT ACCELERATED CORE PROCESSING

Sara Bondi Ogburn

George Mason University, 2019

Thesis Director: Dr. Houman Homayoun

Field-Programmable Gate Arrays (FPGA) are powerful processing platforms to support an efficient processing for a diverse range of applications. Recently, High-Level Synthesis (HLS) tools emerged and shifted the paradigm of hardware design and made the process of mapping high-level programming languages to hardware design such as C to VHDL/Verilog feasible. HLS tools offer many techniques to optimize designs for both area and performance, but resource usage and timing reports of HLS tools mostly deviate from post-implementation results. In addition, to evaluate a hardware design performance, it is critical to determine the maximum achievable clock frequency. Obtaining such information using static timing analysis provided by CAD tools is difficult, due to the multitude of tool options. Moreover, a binary search to find the maximum frequency is tedious, time-consuming, and often does not obtain the optimal result. To address these challenges, this thesis proposes a framework, called Pyramid, that uses machine learning to accurately estimate the optimal performance and resource utilization of an HLS design. For this purpose, first a database of C-to-FPGA results from a diverse set of benchmarks was created.

To find the achievable maximum clock frequency, Minerva was used, which is an automated hardware optimization tool. Minerva determines the close-to-optimal settings of tools, using static timing analysis and a heuristic algorithm, and targets either optimal throughput or optimal throughput-to-area. Pyramid uses the database to train an ensemble machine learning model to map the HLS-reported features to the results of Minerva. To this end, Pyramid re-calibrates the results of HLS's report in order to bridge the accuracy gap, and enables developers to estimate the throughput or throughput-to-area of a hardware design by more than 95% accuracy, without performing the actual implementation process.

Chapter 1: Introduction

The end of the Dennard Scaling [1] era and the thrive to achieve high performance led to the evolution of new computer architecture designs. ASICs are not the best hardware execution platform anymore due to the design complexity, involved cost, and time-to-market challenges.

New platforms such as FPGAs emerged as the potential solution, despite the fact that FPGAs are nearly one order magnitude slower than specialized ASICs [10]. FPGAs enjoy other benefits such as on-the-fly programmability, reconfigurability, energy-efficiency, and the development of hardware/software co-design platforms. This also facilitates developers in designing hardware without requiring deeper insights into such hardware [6].

HLS tools such as Xilinx's Vivado HLS [19] and Intel's HLS [14] are widely used to simplify the design efforts and expedite time-to-market. HLS tools translate a design written in high-level languages such as C/C++/SystemC into a low-level hardware description language. However, HLS-generated register-transfer-level (RTL) models are, in general, not human-readable. HLS has shortened the learning curve of hardware accelerator design by obscuring the details of the hardware execution model. Moreover, HLS enables quick modification of a design by adding directives such as pipeline and unrolling factors that allow programmers to explore the design space.

Additionally, HLS tools report an estimation of the expected timing, latency, and resource utilization of the design. These reports are important since most of the time,

they are the only evidence that a designer can use to repeatedly modify the design and achieve better results.

Throughput and throughput-to-area ratio are some of the most important metrics used for hardware evaluation. In hardware, the maximum throughput depends on the maximum clock frequency supported by the design. The maximum achievable clock frequency of a given HLS design can be estimated or measured at different phases of the design process. An estimation of the maximum clock frequency can be obtained from HLS timing reports [23]. Despite the importance of these reports, many of them are highly inaccurate, as final resource usage and timing reports of HLS depend on the implementation phases (such as logic synthesis and place&route) that are beyond HLS tool capability. Therefore, it is difficult for even state-of-the-art HLS tools to accurately estimate the performance and resource utilization of a design.

On the other hand, it is also possible to calculate the maximum clock frequency in the implementation process. Timing results can be obtained after synthesis, placing&routing, or using actual experimental testing on the board. The post-synthesis and post-place&route results are determined by the FPGA tools using static timing analysis. However, there are challenges associated with the static timing analysis of digital systems designs: The latest version of CAD tools provided by Xilinx (Vivado), do not have the capability to report the maximum frequency achievable for the corresponding code. The user must request a target frequency, and the tool reports either a "pass" or "fail" for its attempt to achieve this goal. While there are 25 optimization strategies predefined in the tool, applying them sequentially or in a number of combinations is extremely tedious and time consuming.

To address these challenges, this thesis proposes a framework called Pyramid

which uses an ensemble machine learning model to estimate the optimal throughput or throughput-to-area of an HLS design only by using information extracted from the reports of HLS tool. To this goal, first, Minerva [24] –an automated hardware optimization tool that employs a unique heuristic algorithm which is customized for frequency search using CAD toolsets, is used. By using Minerva, obtain results were obtained in terms of throughput, and throughput-to-area ratio for the RTL code generated by HLS tool. Then, using HLS reports and their corresponding results from Minerva, a comprehensive dataset was built. This work leveraged hundreds of features that can be readily extracted from the HLS reports to accurately estimate the results of Minerva without actually running the implementation flow. By using this dataset, the Pyramid framework trains an ensemble learning model to achieve high accuracy (more than 95%) for the estimation tasks.

Chapter 2: Related Work

Early works [2] have long attempted to address the issue of expedited design space exploration for a large number of input parameters toward the goal of a lower bound for resource usage while satisfying performance constraints. In [10], the performance prediction for Zynq-SoC is proposed, which estimates the performance based on the execution time of an application on the FPGA. [26] utilizes linear programming apart from machine learning, and focuses on device selection based on analyzed resource usage. In [25], an analytic model is used for area estimation in HLS for SoCs, and also explores HLS optimization pragmas. [12] focuses on a machine leaning solution toward the selection of optimization for synthesis, map, and place-and-route tools using sampling-based reduction of the parameter space. In [11], the authors present InTime, a machine learning approach, supported by a cloud-based compilation infrastructure, to automate the selection of FPGA CAD tool parameters and minimize the TNS (total negative slack) of the design. InTime does not have the capability to find the actual maximum frequency with positive TNS near zero.

Recently, several studies applied machine learning for auto-tuning of frameworks to explore design space of tool parameters for improving FPGA synthesis and implementation [20], [27]. Machine learning was used in HLS to reduce the number of design candidates required to run for implementation [16]. Others have used machine learning for design space exploration by focusing on the regression inaccuracy, rather

than its accuracy [18]. In [15], a three-layer ANN model was trained to estimate the resource usage of post-implementation from pre-characterized area models of a small set of template-based designs. Different from prior work, this work uses machine learning to re-calibrate the results of HLS reports to provide an accurate post-implementation estimate of resource utilization and maximum supported frequency of HLS design for developers.

Chapter 3: Foundational Investigation

Prior to the solution proposed in this thesis, a number of foundational investigations studying the impact of FPGA core processing relative to other processing methods were conducted. This study bears significance to this work because it is important to recognize the benefits and drawbacks of FPGA-accelerated processing before attempting to characterize its design space exploration. Additional study was conducted with the goal of design space exploration in mind, but did not have the correct depth of analysis to bear the same significance in ML training and outputs as is presented in the main body of this work.

3.1 Image Processing Architecture Exploration

The first study involved a comparison among different processing architectures, including CPU, GPU, and FPGA for a selection of 10 OpenCV applications. These applications included filters, computational image processing, input processing, and several general purpose image processing algorithms. This work was built upon the analysis done in [17], and attempted to further understand how optimization of processing cores could factor into performance against a baseline. This includes optimization of the programmable logic (PL) to process several smaller cores in parallel (described in Figure 3.1), and comparison of this optimization to the standard library functions themselves with and without optimization.

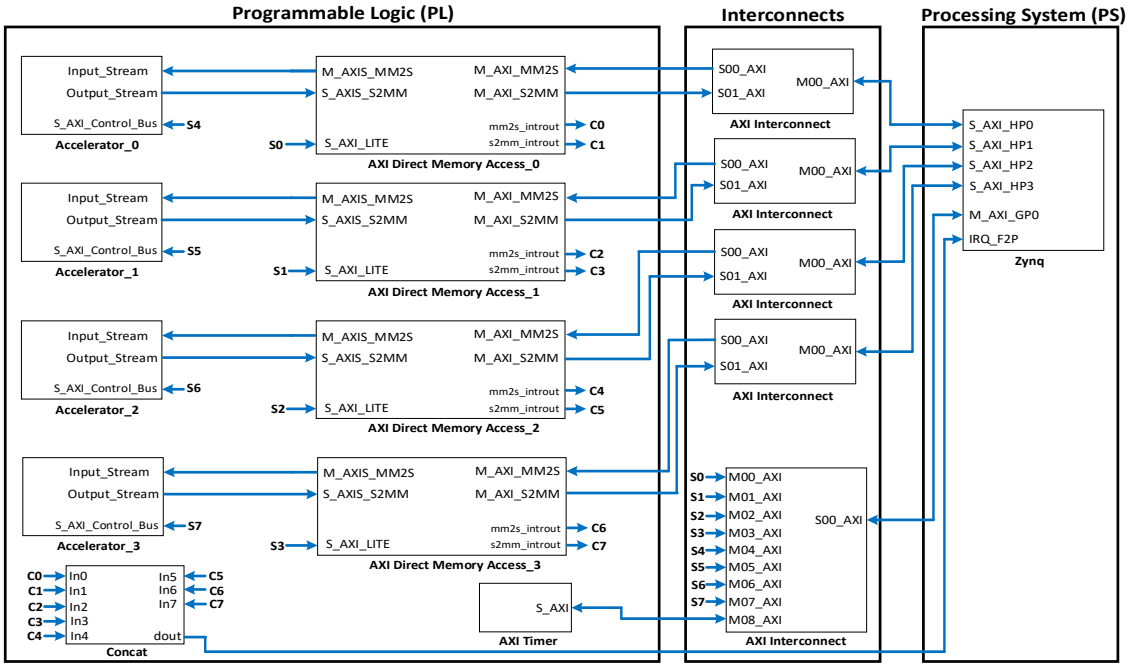

Figure 3.1: Implementation of Parallelized Accelerators [17]

These chosen library functions [29] were optimized for FPGA use with many HLS optimization pragmas for loop unrolling, pipe-lining, etc. Figure 3.2 shows a small subset of the data collected (for a single image size, 250x250 px) compared among the different optimizations. The OpenCV applications were tested using a number of image sizes in addition to the number of processing algorithms used.

While the main body of this work does not experiment with these types of optimizations, it is important to note that it is a promising area for exploration to train a ML model to the benefit of the developer. This would include considerations such as the number and types of HLS optimizations and parallel processing cores that could be added to the training feature set.

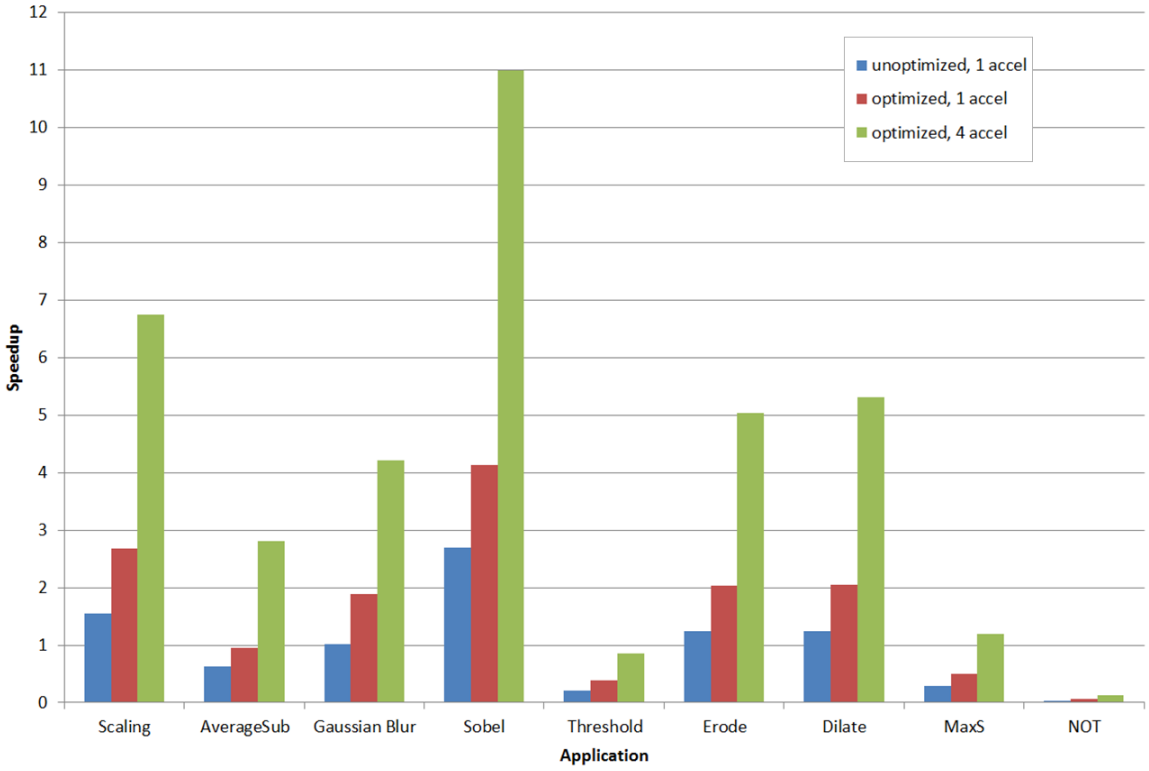

Figure 3.2: FPGA Speedup over ARM A9 Core for OpenCV Applications

3.2 Cross-Platform Design Space Exploration

Another study focused on the gathering of HLS output metrics on a number of different devices as a means to train a ML model, similar to the work presented in the main body of this thesis, as means of performance estimation for selection of devices for a specified task. This work also used a sampling of 10 different FPGA-optimized OpenCV image processing applications on a single image size. A total of 20 devices from three main Xilinx families (Zynq, Artix, and Virtex) were used, including among them 28nm, 20nm (UltraSCALE), and 16nm UltraSCALE+) technologies. Many similar features were used for the training set, such as those associated with

device utilization and performance. However, included was a measure of total estimated execution time of the core and a ratio of how much of that time was spent only in the processing core based on measurements from available tested SoCs, as well as a speedup comparison with the execution time of the same core on an ARM Cortex A9 processor (Figure 3.3).

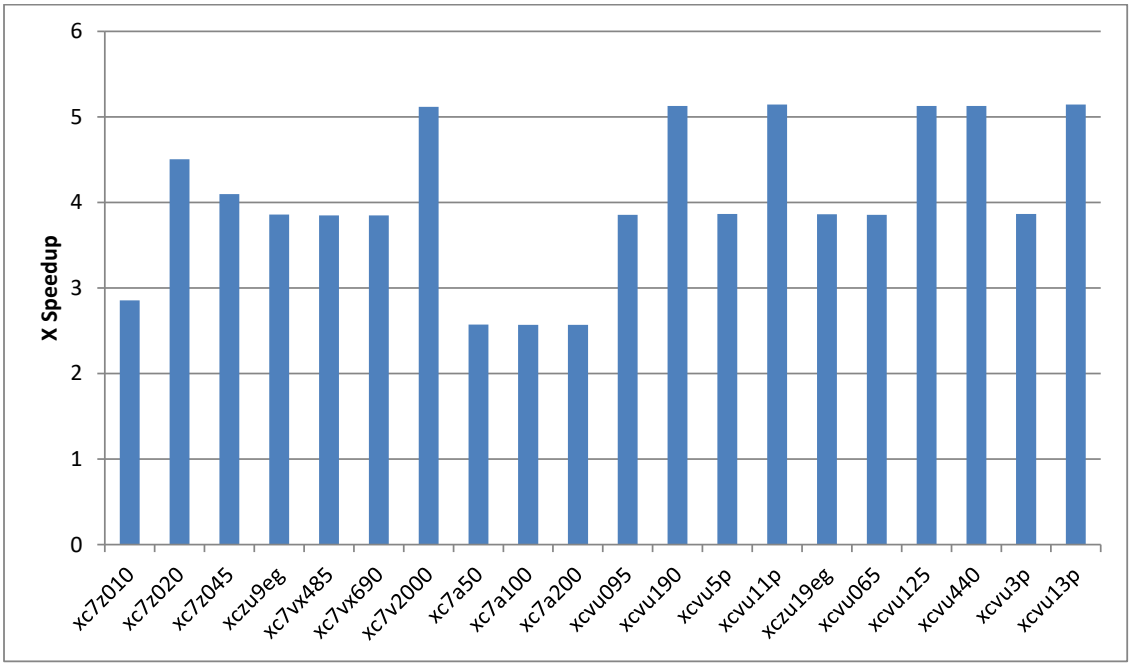

Figure 3.3: Single Accelerator Speedup over ARM for OpenCV Box Filter

Additionally, much like the first study, a theoretical number of parallel processing cores was provided for the ML training set. This took into account the resource utilization of the device under test, such utilization of LUTs, FFs, BRAM, DSPs, and throughput using the maximum DRAM transfer bandwidth. Estimating the number of parallel processing cores allowed for a inclusion of a speedup figure based on either a single core or the total number of theoretical cores. Figure 3.4 shows a

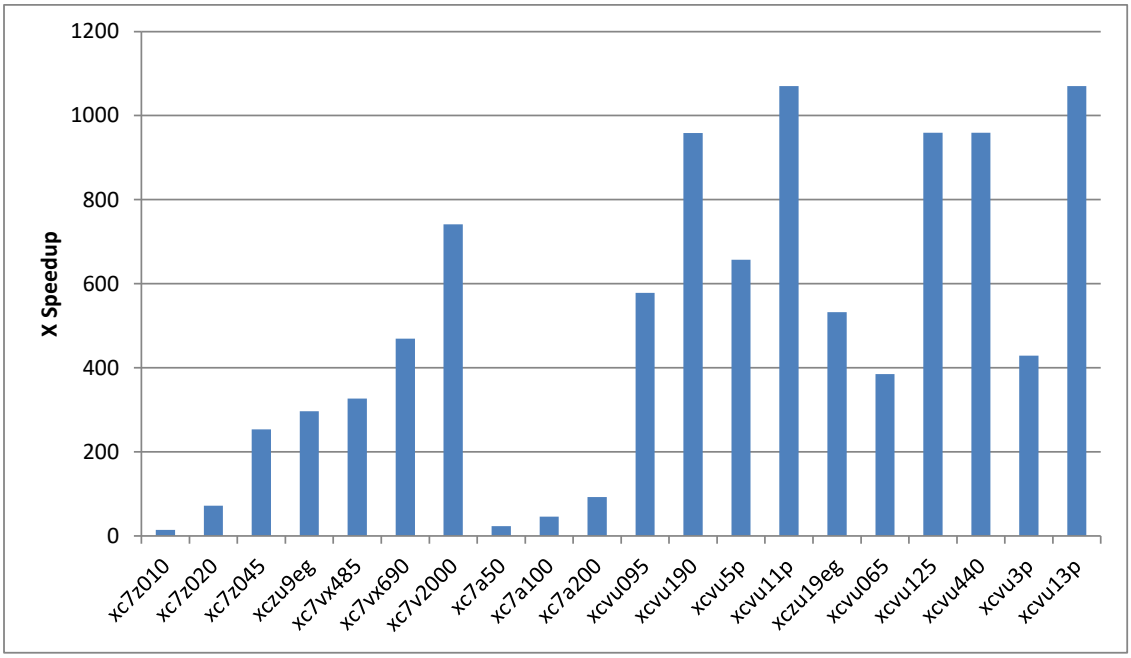

Figure 3.4: Max Accelerator Speedup over ARM for OpenCV Box Filter

sampling of the results obtained in this case. An Artificial Neural Network was used as the ML model, with the end goal of being able to predict device utilization and performance for devices beyond the training set (i.e., future devices that have not yet come to market) for the benefit of designers.

While this work differed significantly in the metrics gathered and their distribution across the selected devices, a major missing component in this work was the inclusion of the post-implementation results. Results from this stage in the design process allow a basis for comparison for the estimations reported by HLS, and without any post-implementation results, it is simply a comparison among HLS reports which still suffer from the inaccuracies of these reports, as discussed in a later section. The main body of this thesis highlights the importance of this data in achieving accurate estimation.

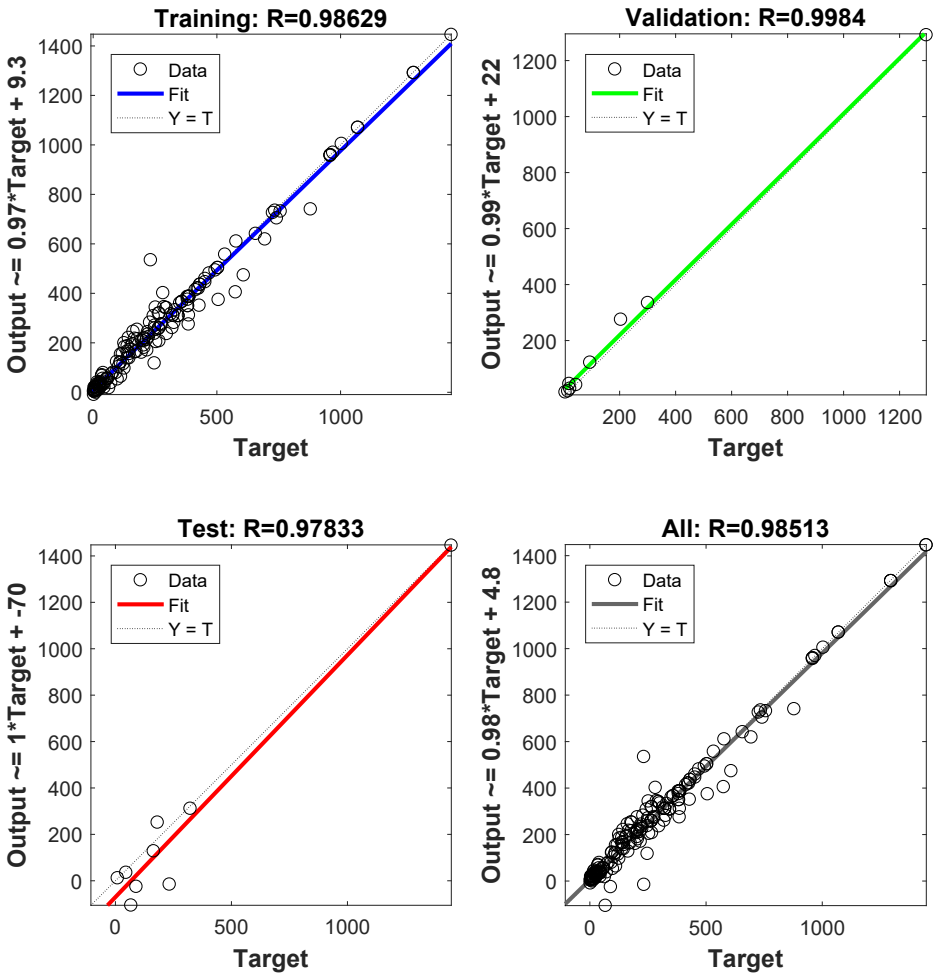

Figure 3.5: Neural Network Performance

Chapter 4: Challenges and Solution

In this section, two major challenges are explained: inaccuracy of HLS report, and evaluation of hardware design that served as motivation to propose an ensemble learning-based framework to improve the accuracy of HLS reports and minimize the time required for finding the optimal resource utilization and timing of the design.

4.1 Inaccuracy of the HLS report

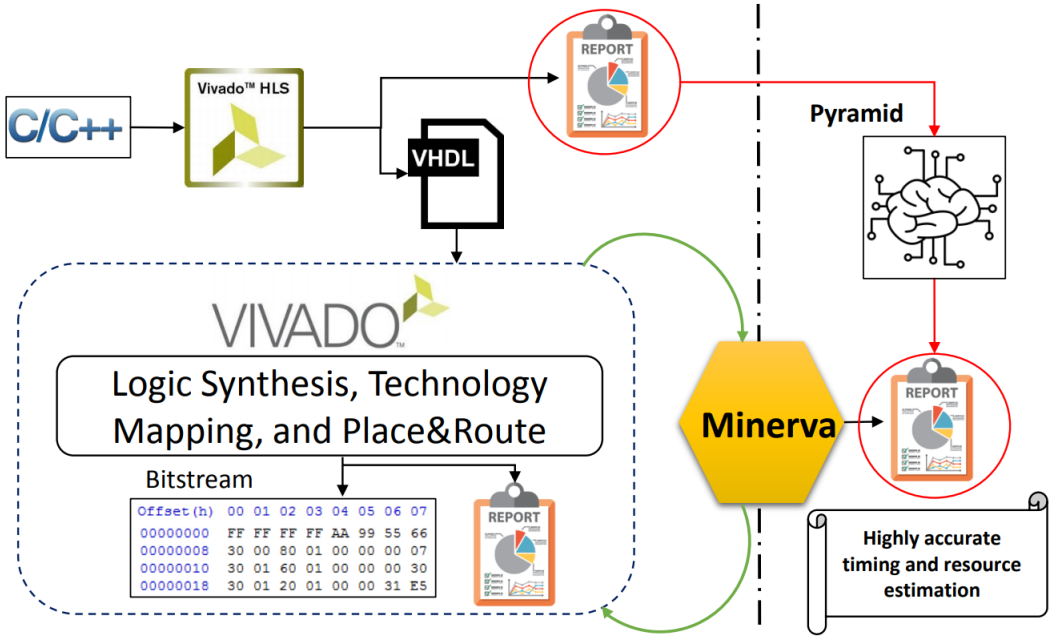

Figure 4.1: HLS design flow + the approach

Figure 4.1 (the left part) shows HLS-based design flow which starts with a high-level software program such as C, C++, or SystemC. The HLS tool translates these high-level language programs into HDL models such as Verilog and VHDL. Additionally, HLS tools report the expected timing and estimated resource usage. At the HLS stage, it is hard to accurately estimate the post-HLS (implementation) results because the implementation process includes many non-trivial steps. Furthermore, resource utilization (such as the number of LUTs, FFs, DSP, and block RAMs) and timing reports depend on the target FPGA specification. HLS tools try to estimate resource and timing of the design by characterizing functional units and instantiated functional for each design. However, such estimation fails to capture the impact of optimization during the implementation. Root Mean Squared Error (RMSE), presented in equation 4.1, was used to evaluate the accuracy of the estimation of HLS's report with respect to the results of post-implementation. This metric is used for reporting of the error throughout this work.

$$RelativeRMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\frac{p_i - a_i}{a_i})^2} \times 100 \tag{4.1}$$

where N is the number of samples, and p_i and a_i are the predicted and actual values of the sample, respectively. Ideally, the percent relative RMSE should be as low as possible. RMSE is a standard metric in regression which is sensitive to scalability. Table 4.1 shows the Relative RMSE for timing and resource utilization for the more than 90 studied benchmarks (details will be discussed later in this paper). The results confirm the large gap between what is reported by HLS and the implementation. This in fact is due to the implementation process, where the HDL

Table 4.1: Average HLS estimation error over 90 benchmarks

Devices	Artix7		Kintex7		Virtex7	
Targets	Resource	Timing	Resource	Timing	Resource	Timing
HLS Estimate	91.7%	23.6%	112.5%	28.1%	88.4%	17.2%

models go through logic synthesis, technology mapping, and placement and routing which are not considered by the HLS tools. Therefore, post-implementation reports, including the actual resource usage and timing on the target FPGA, significantly differs from the HLS report.

4.2 Challenge of hardware evaluation

Throughput and throughput-to-area (Freq./#LUTs) are two of the most common metrics for the evaluation of a hardware design. The calculation of these metrics is complex, as there are several challenges in using static timing analysis for finding the maximum clock frequency of a design. To show the challenge in finding maximum clock frequency, synthesis and implementation were performed for the VHDL code of a CAESAR Round 2 candidate (ICEPOLE). Worst Negative Slack (WNS) results were generated for 128 different target clock frequencies in order to observe the trend. The target clock frequency was set to 333 MHz, and the theoretically achievable frequency (further referred to as the reference frequency) was calculated based on WNS, utilizing the following equation:

$$MinimumClock = TargetClock - WNS \tag{4.2}$$

In the next step, WNS results were generated for the requested clock frequency

varying from -64 to +64 MHz of the reference frequency, with a precision of 1 MHz. Figure 4.2 shows the trend.

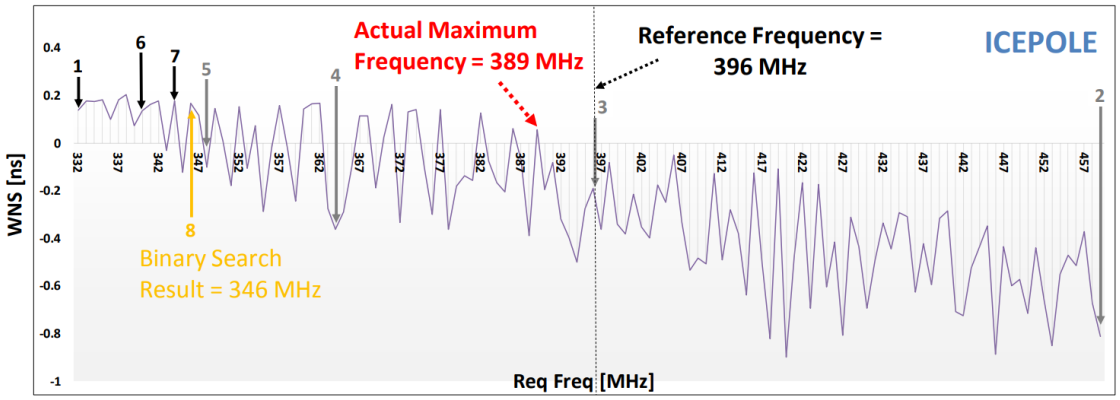

Figure 4.2: Dependence of WNS on the Requested Clock Frequency and graphical representation of the binary search scheme

As observed in this Figure, there are fluctuations around the calculated reference clock frequency. As can be observed, the result of binary search is 346 MHz (number 8 in the figure), which is not the correct maximum frequency. Based on the ICEPOLE graph, the maximum frequency is 389 MHz. Therefore, based on the aforementioned graph, designer cannot only rely on the above equation to calculate the actual maximum clock frequency. It is important to note that these results were obtained using only default options of Vivado. Given the vast optimization strategies that exist in Vivado, calculating the maximum clock frequency becomes more challenging. Another challenge is that Vivado Design Suite offers 25 predefined optimization strategies, which can be used to achieve a higher maximum frequency and a more optimized

design. Hence, incorporating all of these strategies leads to an even more tedious navigation process. Therefore, a designer cannot reasonably be expected to navigate all 25 optimization strategies due to the time-consuming process.

4.3 Solution

This work demonstrated that a commercial HLS tool targeting FPGAs incurs a large error of 97.5% in estimating the resource usage. Similarly, the error for timing estimation was found to be 22.9%. Such inaccurate estimates prevent developers from applying the appropriate set of optimizations, leading to a poor trade-off. Moreover, the static timing analysis of digital systems design provided in state-of-the-art CAD tools is not able to report the maximum frequency achievable. Instead, the user must deal directly with tens of optimization strategies available in the tools, which is time-consuming and tedious. To circumvent such a brute-force approach to find the maximum frequency, a recent work [24] proposed an automated hardware optimization tool called Minerva. Minerva determines the close-to-optimal settings of tools using static timing analysis and a heuristic algorithm developed by the authors, and targets either optimal throughput (TP) or optimal throughput-to-area (TPA) ratio. Minerva is designed to be used to automate the task of finding optimized results. However, based on the size of the design, using Minerva may take a few minutes and up to several hours.

Therefore, as a solution to these challenges, this work proposes a machine learning based framework (called Pyramid) to re-calibrate the HLS reported results and map them to the results of Minerva using the ensemble machine learning method. Figure 4.1 (the right part) shows the overview of Pyramid. In this way, without going

through the time-consuming process of full end-to-end implementation, using Pyramid, developers can have an accurate post-implementation estimation of resource utilization and maximum supported frequency of the design just after getting the report of their HLS design.

Chapter 5: Experimental Setup

For building a machine learning model, first a dataset for training the model is required. For this purpose, the results of HLS tool and its corresponding optimal implementation's results needed to be obtained. Here, this thesis presents the benchmarks, FPGA devices, and the methodology of performing the experiments to collect the required data.

5.1 Benchmarks and FPGA devices

The diversity of benchmarks is important in this study. Therefore, popular HLS benchmark suites were used to make sure that the evaluation is comprehensive. The selected benchmarks are from Machsuite [8], S2CBench [9], CHStone [4], and Rosetta [28]. To increase the diversity and the size of the dataset, CAESAR Round 3 Candidates' HLS-ready codes [30] and a collection of 10 different image processing kernels from Xilinx xfOpenCV [29] were used. A total of 90 benchmarks which include a wide range of domains from simple kernels to machine learning and real-time video processing that reflect the latest application trends wer used. Figure 5.1 shows a detail of the benchmarks used.

The benchmarks are subdivided into four categories, such as machine learning, image/video processing, cryptography, and mathematical applications to validate this work's hypothesis that the results of the machine learning techniques are application dependent. The default version of HLS designs were used without applying

Table 5.1: Benchmarks Used

Suite	Benchmarks	Abbreviations
Machsuite	backprop, gemm, stencil, viterbi, nw	M1, M2, M3, M4, M5
Rosetta	digit-recognition, face-recognition, 3d-rendering, BNN, spam filtering	R1, R2, R3, R4, R5
CAESAR	Tiaoxin, KetjeSr, OCB, NORX, AEGIS	C1, C2, C3, C4, C5
SHA-3	BLAKE, Groestl, Keccak, Skein	H1, H2, H4, H5

any further directive to the designs. However, the dataset can be expanded by synthesizing designs with additional HLS optimization directives. For selecting FPGA devices, three different classes of FPGAs such as Low-end, Medium-end, and High-end were targeted. All chosen devices were selected from Xilinx as follows: Artix7 (xc7a100tfgg484-3), Kintex7 (xc7k420tffv901-3), and Virtex7 (xc7vx980tffg1930-2). The FPGA devices were chosen based on a wide array of available resources and technologies across the spectrum of each family. The software used for the experiments were Xilinx Vivado and Vivado HLS version 2017.2. Figures 5.1 and 5.2 show examples of these outputs reports in the number of LUTs used.

5.2 Data collection

To build a dataset, first it is required to extract all possible features that can be collected from HLS reports. Inputs of final machine learning models consist of HLS-related features. To obtain the maximum achievable clock frequency of the design, Minerva was used. Minerva executes Vivado in batch mode, utilizing the Vivado batch mode TCL scripts provided by Xilinx. An XML-based Python program was used to manage runs. This program launches Vivado with TCL scripts that are

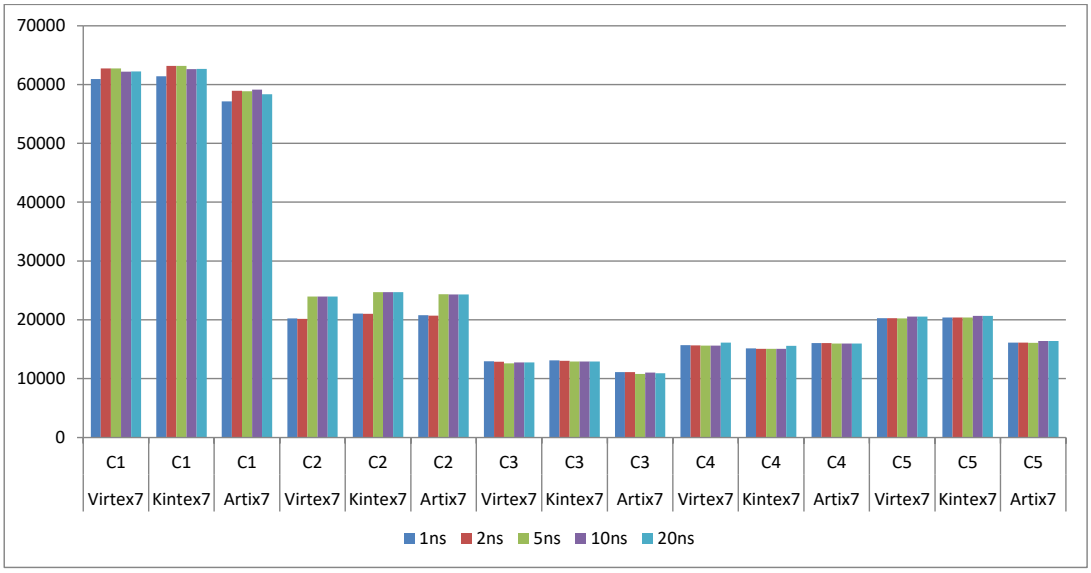

Figure 5.1: LUTs Used as reported by HLS for crypto. applications

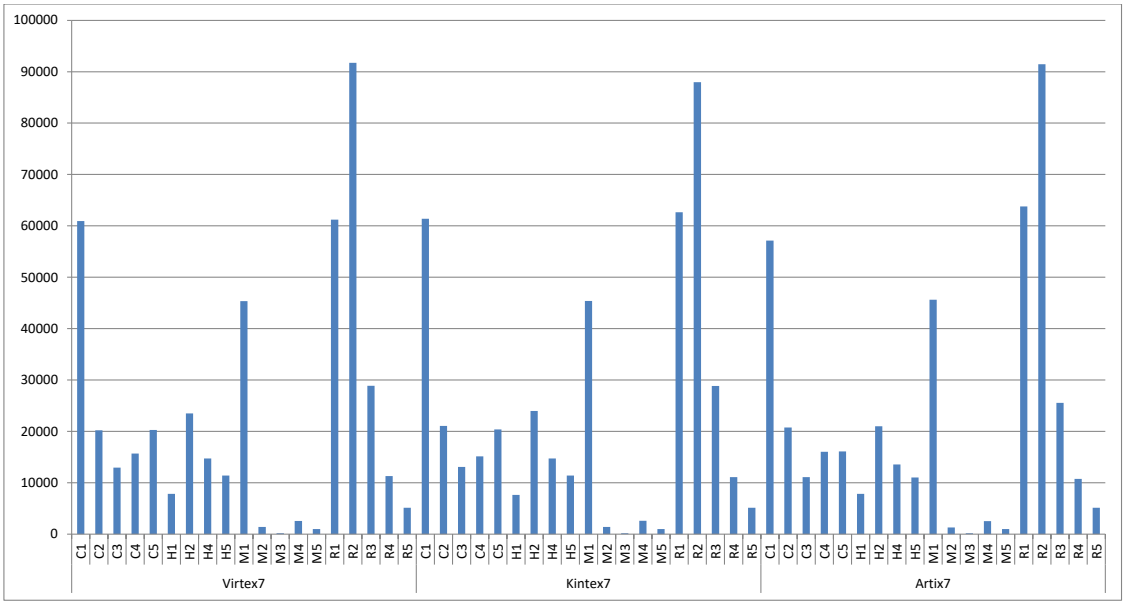

Figure 5.2: Estimated LUTs used for all applications at 1 ns clk period

dynamically created during run-time and later modified to perform each step of the optimization algorithm. Minerva's output includes the maximum achievable clock frequency for the design, the optimization strategy that lead to such a result, and the resource utilization for that implementation. The output of Minerva was also parsed. Minerva's outputs are used as the target values in the dataset. Therefore, the machine learning model will be trained to estimate resource usages for LUT, FF, DSP, and BRAM, as well as maximum clock frequency (totally 5 targets) reported by Minerva.

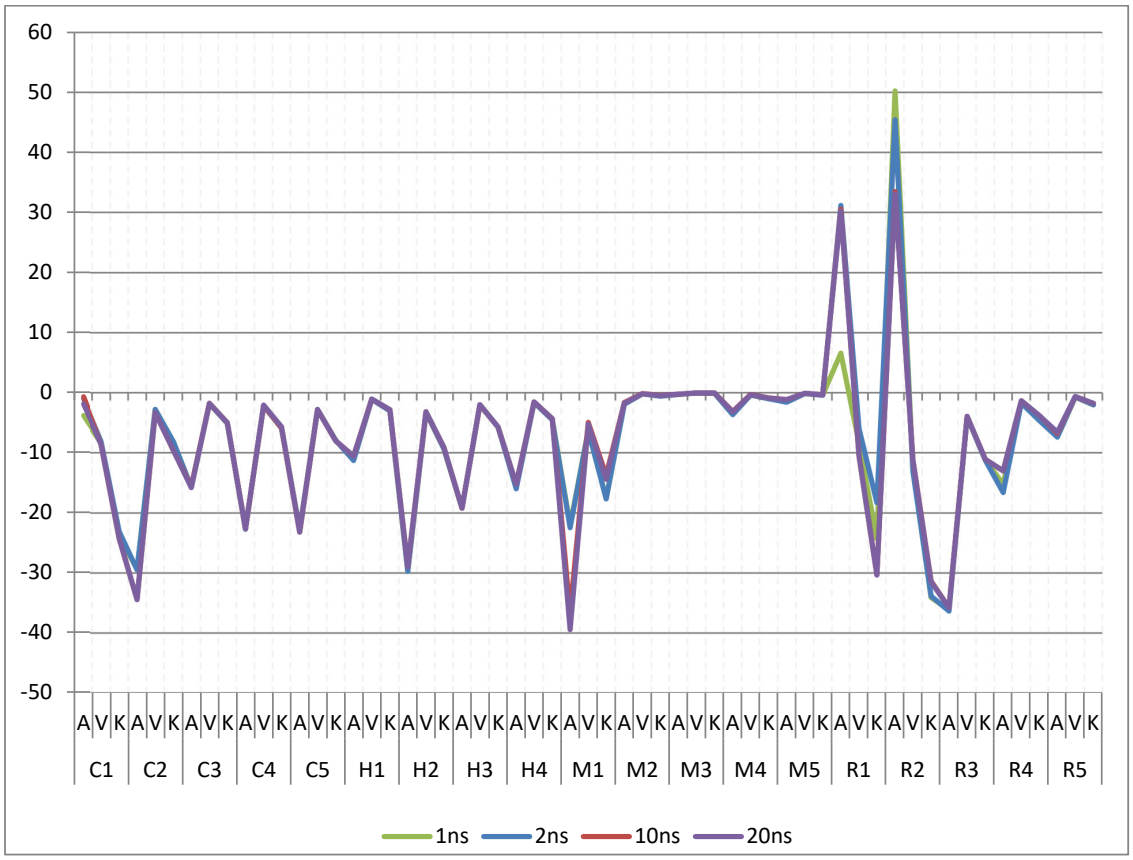

Figure 5.3: Delta of % LUTs Used Reported by HLS vs. Minerva

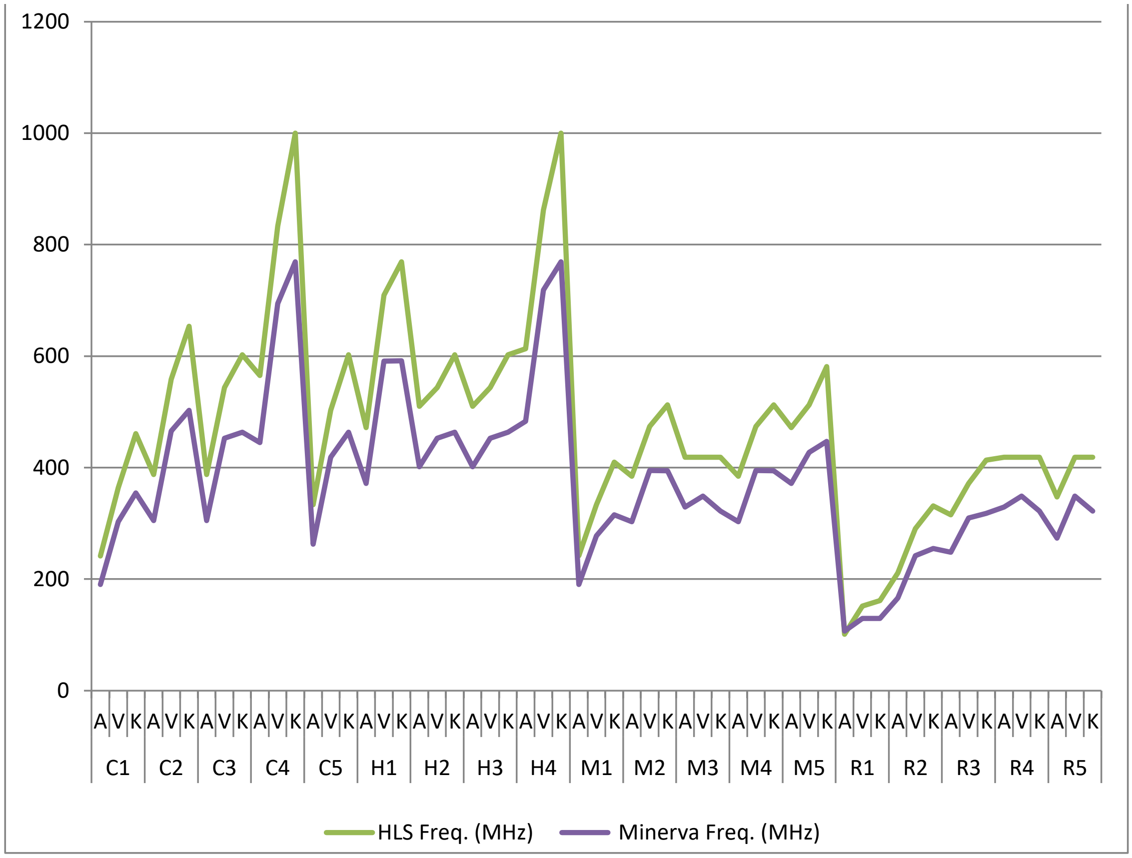

Figure 5.4: Reported Freq. for HLS vs. Minerva at 1 ns clk period

As it is not possible to determine the importance of each feature in advance, this work extracted as many relevant features as possible from HLS reports (total 183 features). The flow of obtaining the results for each design is as follows: 1) Set a timing constraint (clock period) [1, 2, 5, 10, 20ns] and the FPGA device for HLS tool. 2) Run HLS and get the VHDL files of design, and also extract all features from the HLS report. 4) Use Minerva to find the maximum clock frequency and the corresponding resource utilization for two different targets (throughput, and throughput-to-area). Minerva uses out-of-context (OOC) implementation option to run Vivado. 5) Repeat the whole process for the next clock period and also for the remaining FPGA devices.

In this way, this work created a comprehensive dataset ($90 \times 5 \times 3 \times 2 = 2700$ samples) that can be used for the training of machine learning models. The dataset is separated into two parts: training/validation and testing. 20% of the dataset the was randomaly selected for final testing as unseen data and the remaining 80% was considered the training/validation part. 4-fold cross-validation was performed on the training/validation set to train the ML models. This means that in each iteration of training, a random 75% of the data were used for training and 25% were used for validation. It took three months to create this dataset, using 10 servers, each of them equipped with 16-core processors and 128GB memory. While this is the most time consuming part of the solution, the entire training process is only done once. The ML models described in Section 6 were implemented in Python leveraging the scikit-learn [7] and TensorFlow [13].

5.3 Feature reduction

So far, as many relevant features as possible were extracted to build a comprehensive dataset. However, this process produces a very highly dimensional dataset. This high dimensionality may lead to complex models that require longer training time, increase the chance of over-fitting, and are harder to interpret. Therefore, instead of accounting for all extracted features, irrelevant and redundant features were identified and removed and only a subset of features were selected. To remove redundant features, the Pearson's correlation coefficient (eq. 5.1) was computed for each pair of extracted features x and y, and one feature from each group of correlated features was selected.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{5.1}$$

To eliminate irrelevant features, a linear model with L2 regularization (in section 6.1) was used to fit the data. The outcome of L2-regularized linear model is a sparse estimator that zeros out the coefficients of unimportant features. By eliminating unimportant features, the number of features was reduced from 183 to 72. Table 5.2 shows the features that were eventually selected from HLS reports.

Table 5.2: Description of features

Feature Category	Brief Description	# of features
Performance	Requested clock period, estimated clock period by HLS, Uncertainty	3
Resources	Utilization and availability of LUT, FF, DSP, and BRAM	36
Logic and arithmetic operation	Bitwidth/resource statistics of operations	29
Memory	Number of memory words/banks/bits	2
Multiplexer	Multiplexer input size/bitwidth	2

Chapter 6: Solution Framework and Observations

In this section, this thesis presents the Pyramid framework, which uses popular machine learning models to enable fast and accurate resource and timing estimations for HLS designs. Use of machine learning has become popular in design automation, as it provides the means to accurately capture the factors impacting the accuracy of timing and resource estimation. Moreover, analytical modeling [21] and statistic reasoning have been used to construct the estimation models as a function of multiple parameters for the evaluation of hardware design [22], [23]. This work investigated whether the models built by machine learning techniques are sufficiently accurate for timing and resource utilization of a design on a specific FPGA when the HLS tool is used by taking into account 25 different optimization strategies available for the implementation of a design.

6.1 Estimation models

For this purpose, a regression model, artificial neural network (ANN), support vector machine (SVM), and random forest (RF) were employed. Each belongs to a different branch of machine learning to construct a timing model and resource estimation models for a diverse set of benchmarks targeting different FPGA devices. With the appropriate dataset in hand, the complexity of the implementation process as a practical solution to the HLS estimation problem can be modeled.

Linear Regression For a given dataset $\{Y_i, X_{ij}\}$ where Y_i is the target and X_{ij} are features, a regression model is as follows: $\{Y_i = W \times X_{ij}\}$. The training goal of this model is to find W (vector of coefficients) such that the loss between X and Y is minimized. In this work, the Ridge regression model was used. By adding a degree of bias to the regression estimates, Ridge regression reduces the standard errors. Ridge regression solves the multicollinearity problem through the shrinkage parameter λ. Also, it uses the L_2 regularization method.

$$= argmin||Y - XW||_2^2 + \lambda||W||_2^2 \tag{6.1}$$

This model was used to observe how much linearity existed among the chosen features.

Artificial Neural Network Unlike regression and linear models, an artificial neural network (ANN) was selected to create a non-linear model between the target and input features. Neural networks have an input layer and an output layer. The input and output layers are connected to each other through a series of hidden layers. To capture complicated non-linear functions, the depth of neural networks can be increased. ANN has large tuning hyperparameters such as determining the number of layers and neurons per layer. Figure 6.1 shows a simplified example of the algorithm structure.

Support Vector Machine SVM analysis is a popular machine learning tool for nonlinear functions. SVM is considered a nonparametric technique because it relies on kernel functions. Some problems cannot adequately be described using a linear

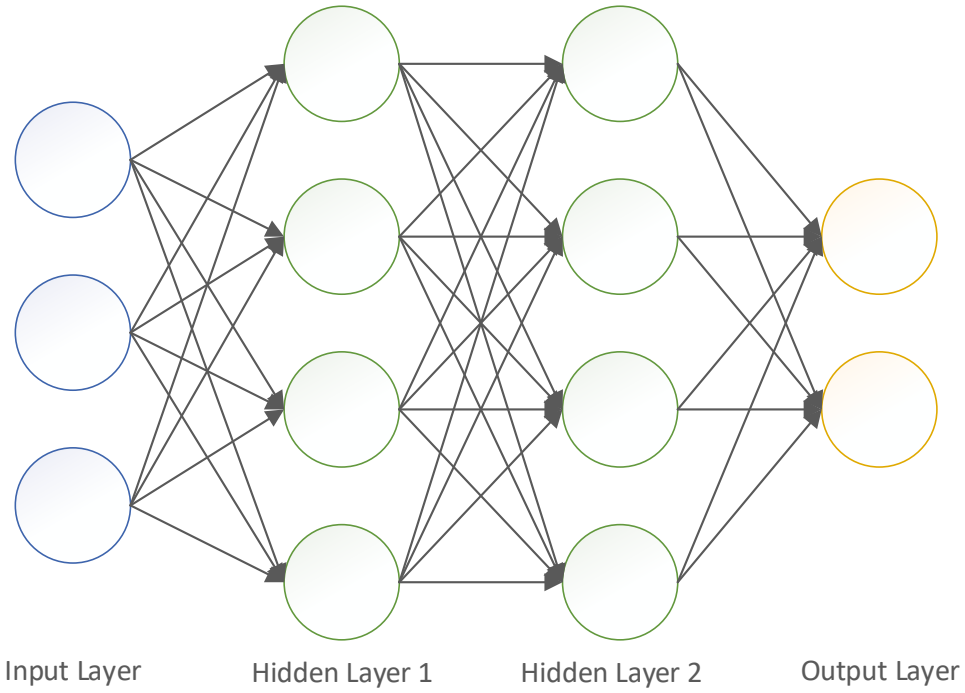

Figure 6.1: ANN Structure

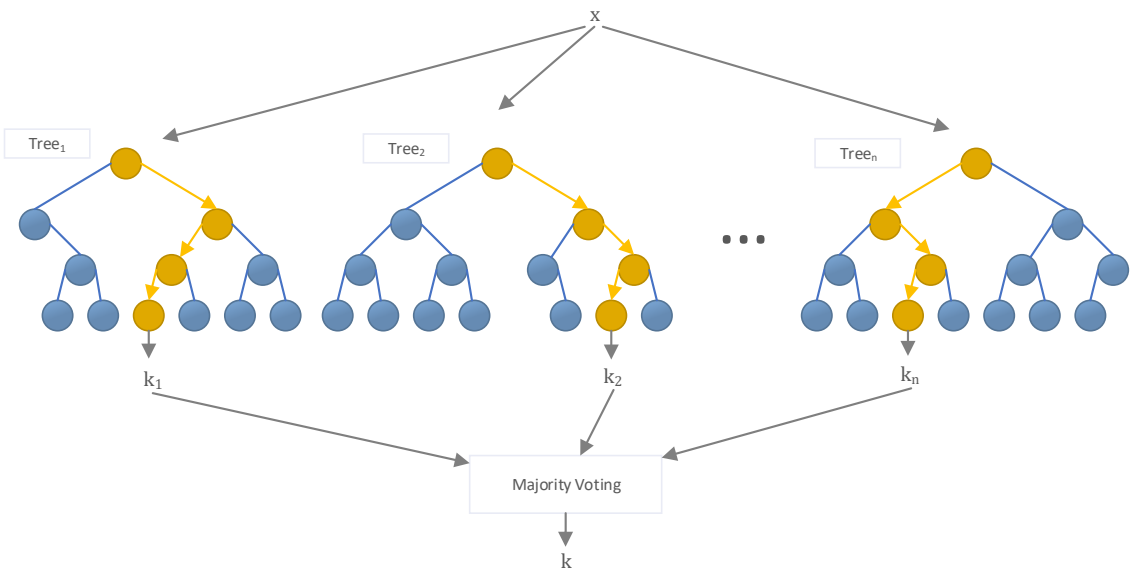

Figure 6.2: Random Forest Structure

model. In such a case, the Lagrange dual formulation (Shown in eq. 6.2) in SVM allows the technique to be extended to nonlinear functions.

$$\begin{aligned} &\text{maximize } f(c_1 ... c_n) = \sum_{i=1}^{n}(c_i) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i c_i (x_i \cdot x_j) y_j c_j, \\ &\text{subject to } \sum_{i=1}^{n} c_i y_i = 0, \text{ and } 0 \leq c_i \leq \frac{1}{2n\lambda} \text{ for all } i. \end{aligned} \tag{6.2}$$

Random Forest Random Forest (Figure 6.2) is a flexible and easy to use machine learning method that most of the time, even without hyperparameter tuning, achieves a high accuracy estimation. Random Forest is a supervised learning algorithm. One of the advantages of the Random Forest model is that it enables the measuring of the relative importance of each feature on the prediction.

For finding the best set of hyperparameters (λ for regression and number of layers and neurons in ANN), Grid search [5] was employed. An ANN with four hidden layers was decided upon, such that the number of neurons in each layer is as follows: 105, 60, 44, and 30. Figure 6.3 shows the error of machine learning models for resource and timing estimation.

The results show that the average errors of models built by LR, ANN, SVM, and RF are 23%, 14%, 19%, and 15%, respectively. For the estimation problem, the number of features is relatively high, and the amount of training data is limited. It is important to note that collecting such a limited amount of data took more than 3 months. Given the tedious and time consuming process, full end-to-end implementation data collection for training ML models is a major challenge in this research, in

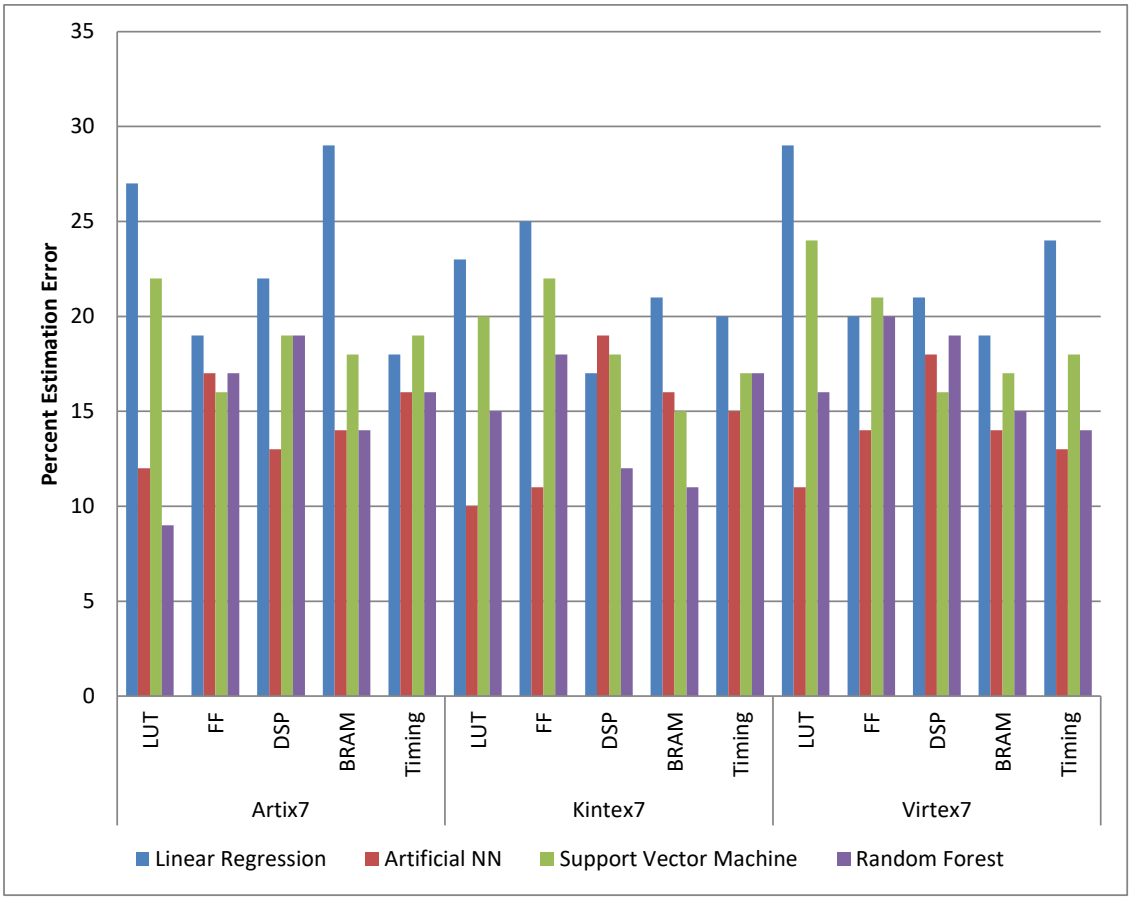

Figure 6.3: ML Estimation Errors

addition to the high error rate when using general ML estimators. Machine learning techniques such as NN, RF, and SVM can highly accurately represent complicated non-linear functions when a large amount of training data is provided for the model to converge. Given the available dataset, and reasonable amount of time for data collection, these experiments demonstrate that merely using general machine learning techniques fail to build accurate models. Therefore, this motivated this work to seek an ensemble learning approach to improve the accuracy of estimation.

6.1.1 Ensemble model

Ensemble learning is a branch of machine learning which is used to improve the accuracy and performance of general ML models by generating a set of base learners and combining their outputs for final decision. It fully exploits complementary information of different estimators to improve the decision accuracy and performance. This work used the stacked regression [3] method, where a number of first level estimators are combined using a second-level estimator. The key idea is to train a second-level estimator based on the output of the first level estimators via cross-validation. It is critical to ensure that the base estimators are formed using a batch of training datasets that are different from the one used to form the new dataset. The second step is to treat the new dataset as a new problem, and employ a learning algorithm to solve it.

To use the stacking approach, two parameters must be determined: the threshold for accuracy and the maximum number of iterations. After each stage, the accuracy must be checked. If the model meets the target estimation accuracy, then the process of model creation is stopped. Otherwise, the iterations are continued until reaching the desired accuracy or the maximum number of iterations. When the parameters of the first-order model are determined, the accuracy of the timing and resource estimation can be evaluated. If the target accuracy is met, sub-models are no longer created. Otherwise, a higher order model is required to further be created until the threshold of iterations is reached.

Table 6.1 shows the estimation errors of timing and utilization models created by Pyramid for two different optimization goals such as throughput (TP) and throughput-to-area (TPA) using the stacking approach. Later in this section, this thesis

Table 6.1: Average error of Pyramid estimations

Devices	Artix7		Kintex7		Virtex7	
Targets	Resource	Timing	Resource	Timing	Resource	Timing
Pyramid-TP	6.3%	3.8%	5.2%	4.1%	4.9%	4.4%
Pyramid-TPA	4.8%	3.5%	4.7%	4.6%	4.8%	4.9%

describes how the model is constructed. Results show that the average error of the model is only 4.7%. Table 6.2 presents the estimation results of ML techniques with regard to the benchmarks' categories. The interesting observation is that the accuracy of estimations for mathematical benchmarks are high even with linear regression. On the other hand, the resource and timing estimation of machine learning and cryptography benchmarks are lower with general ML techniques. However, ensemble learning shows a good accuracy for all benchmarks.

Table 6.2: Average error of ML techniques for different benchmarks

Benchmark category	**Machine Learning**		**Img/Vid Processing**		**Crypto.**		**Mathe.**	
Targets	Res	Tim	Res	Tim	Res	Tim	Res	Tim
LR	29%	25%	17%	16%	38%	22%	11%	8%
ANN	17%	14%	13%	11%	19%	14%	8%	7%
SVM	22%	19%	18%	17%	23%	18%	10%	7%
RF	16%	16%	14%	12%	20%	15%	9%	7%
Ensemble	6%	5%	4%	3%	5%	4%	4%	3%

While solely using a general machine learning technique incurs a large error, the Pyramid framework helps to create an accurate model for estimating the optimal throughput and throughput-to-area of an HLS design while also taking into account

the size of the dataset and high dimensionality of the features.

One of the issues in using the stacking approach is actually finding the right weights to combine the models, as the topology and hyperparameters of the model are the keys for making the ensemble method works best. To address this challenge and avoid putting the burden on the end-user, this work suggests the following guidelines to be followed by developers in order to both facilitate adopting this framework and make it possible to reproduce the results presented here.

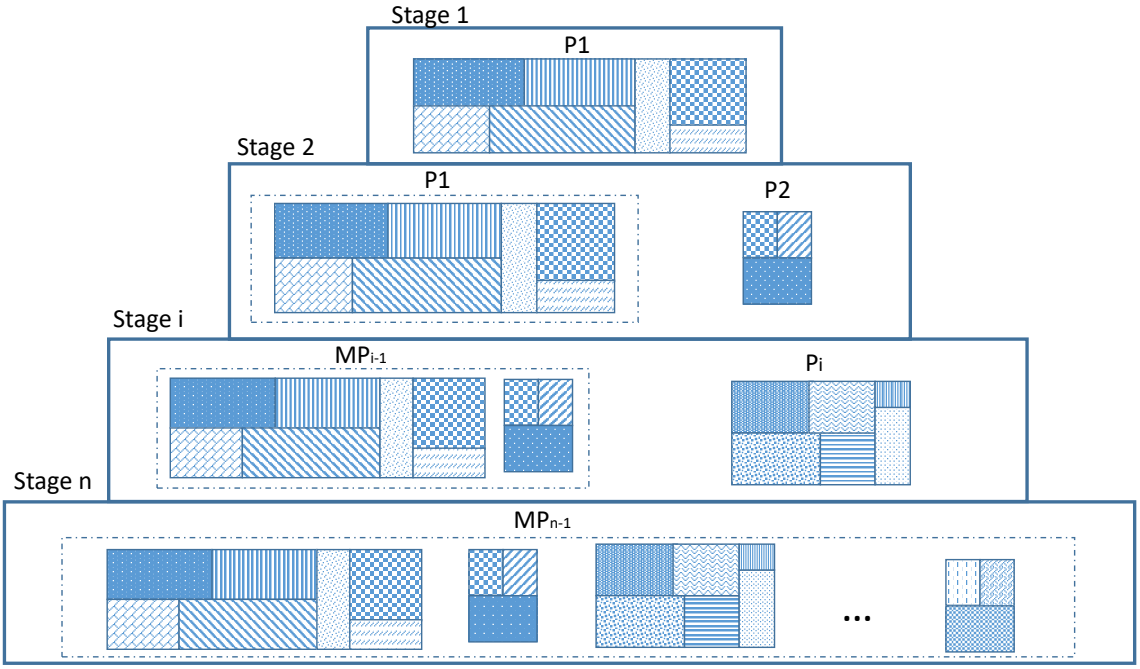

Figure 6.4: Overview of the stacking approach

6.2 Guidelines

Figure 6.4 shows the overview of the stacking approach. In each stage, a sub-model can be created by any arbitrary machine learning technique such as LR or NN, and they can be considered as a black-box. As an example, this work employed a simple three-layer fully connected neural network with 20 hidden neurons employed to create the sub-models. The target accuracy and the threshold for the maximum number of iterations were set to 99% and 50 respectively. In the first stage, a single sub-model (P1) was created as a function of the features from Bootstrap samples of the main dataset. Each time, 20% of samples were randomly selected with replacement. Bootstrapping was important here as the dataset is relatively small. For the second stage, another neural network (P2) was employed to model the variation in the estimation of designs that was not modeled by P1. The boxes of different shades in Figure 6.4 signify the parts of dataset which have been modeled. Now, a primary mixed model, MP1, can be built through the combination of the first two sub-models P1 and P2 as follows: $MP1 = \alpha 1P1 + \alpha 2P2$. P1 and P2 represent the estimated timing by sub-models, and αs are the coefficients which corresponded to the learning rate. To simplify the procedure, the α values were set to 0.1. Sub-models were repeatedly created and added to the mixed model.

If the model achieves the desired accuracy before the total number of iteration, the final model is obtained. This is called the first-order model (FM1). If the target accuracy is not reached after the threshold on the number of iterations, the above procedure can be repeated to create another mixed model (FM2), or the user can end the process. In the case of this work, the target accuracy was not met after 50

iterations. However, more than 95% accuracy was reached, which is acceptable for this case and the training procedure was stopped.

It is noteworthy to consider the following rules of thumb: A lower value for the learning rate increases the number of sub-models required to create the first-order model at a specific accuracy. On top of that, changing the parameter of each sub-model affects the total number of sub-models needed to create the first-order model. Moreover, increasing the complexity of sub-models results in fewer sub-models required to achieve the maximum accuracy at a given learning rate.

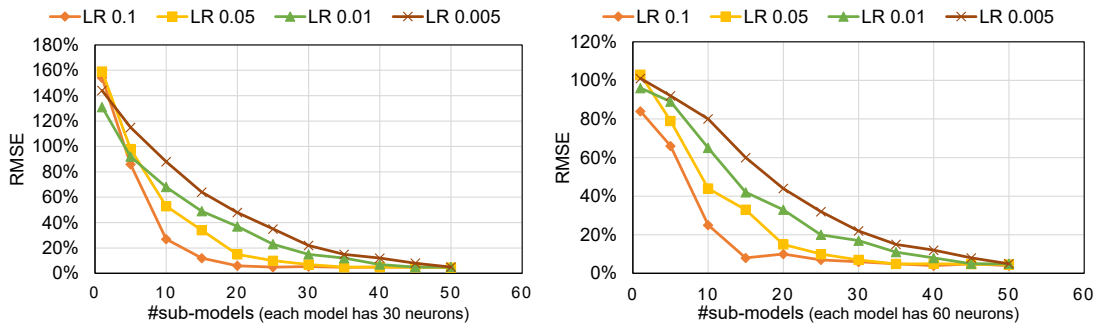

Figure 6.5: The relationship between parameters and errors

In regard to accuracy, Figure 6.5 demonstrates the relation among discussed parameters. It can be observed that when the number of neurons and the number of sub-models is low, the error is always high. By increasing the complexity of the NN or the number of sub-models, the minimum error decreases to under 5%. It can also be observed that the convergence speed is much slower with a small learning rate value. While a larger learning rate can converge faster, a fluctuation in the accuracy is seen.

This approach is a sequential procedure in which the primary model remains unchanged at each step. However, as the approach proceeds, the model becomes more accurate. This approach helps to mitigate the over-fitting issue for a dataset with a limited number of samples and high dimensionality of features.

6.3 Future Work

The most straightforward area for future improvement of this work includes the expansion of the training dataset. This task, however, is non-trivial given the amount of time necessary to properly obtain the results needed for feeding to the ML model proposed in this work. Because of the large disparity in resource/performance estimation based on application dependence, study of an even larger set of benchmarks can only provide benefit above the analysis presented here. Additionally, though the three devices studied in this work provided high diversity in their specification, the inclusion of even more devices with different resource availability and performance specification would aid in a more comprehensively trained ML model for estimation.

Lastly, one of the largest and most interesting areas for exploration in the training dataset is the inclusion of further optimizations available in the tools used to generate results. Though Minerva provides a peak optimization strategy for a given design in Vivado, the default optimization strategies were used in HLS. However, since the nature of these optimization strategies can be far more expansive (i.e., a multitude of different pragma directives at various points throughout the source code, or several instances of a benchmark as described in section 3), this type of exploration adds an additional level of complexity to what is already a time-consuming process. This type of exploration would likely benefit an automated or highly systematic approach.

Chapter 7: Conclusions

This thesis thoroughly studied one of the main challenges of evaluating an HLS design: the inaccuracy of HLS reports in both timing and resource utilization. Moreover, HLS reports lack insights for finding the optimal throughput or throughput-to-area of the generated RTL design. First, several foundational studies were carried out to not only understand the impact of using FPGA core processing relative to other processing architectures, but to more fully investigate the design space exploration that can be accomplished using only HLS outputs. To address these challenges, this work proposed Pyramid, a framework that uses an ensemble learning technique to bridge the accuracy gap between HLS reports and the optimal achievable throughput or throughput-to-area of the HLS design. To achieve this, first Minerva, an automated hardware optimization tool, was used to find the maximum clock frequency and resource utilization of the RTL code of the design generated by HLS tool. Then the stacking approach was used to map the features extracted from HLS reports to Minerva's output. Collecting data for fully implemented HLS designs in order to create a training dataset is an extremely time-consuming task, making it impractical to collect the large data needed to train general machine learning estimators. Therefore, as the dimensionality of features is high and the sample size of the dataset is not significantly large, the obtained accuracy of several studied ML estimators is found to be low. In response, the model created by Pyramid framework using ensemble

learning is shown to have more than 95% accuracy. Several ML techniques were utilized for ensemble learning, including linear regression, ANN, support vector machine, and random forest. Areas for future improvement of this work include a dataset with more applications and a wider array of devices, as well as the exploration of HLS optimizations and their impact on results used for analysis.

Bibliography

[1] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974.

[2] A. Sharma and R. Jain, "Estimating architectural resources and performance for high-level synthesis applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 2, pp. 175–190, Jun. 1993.

[3] L. Breiman, "Stacked regressions," *Mach. Learn.*, vol. 24, no. 1, pp. 49–64, Jul. 1996.

[4] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "Chstone: A benchmark program suite for practical c-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems*, May 2008, pp. 1192–1195.

[5] S. K. Smit and A. E. Eiben, "Comparing parameter tuning methods for evolutionary algorithms," in *2009 IEEE Congress on Evolutionary Computation*, May 2009, pp. 399–406.

[6] B. Holland, A. D. George, H. Lam, and M. C. Smith, "An analytical model for multilevel performance prediction of multi-fpga systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 3, 27:1–27:28, Aug. 2011.

[7] 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 É. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011.

[8] B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 110–119.

[9] B. C. Schafer and A. Mahapatra, "S2cbench: Synthesizable systemc benchmark suite for high-level synthesis," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, Sep. 2014.

[10] D. Jiménez-González, C. Álvarez, A. Filgueras, X. Martorell, J. Langer, J. Noguera, and K. A. Vissers, "Coarse-grain performance estimator for heterogeneous parallel computing architectures like zynq all-programmable soc," *CoRR*, vol. abs/1508.06830, 2015.

[11] 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*, ser. FPGA '15, Monterey, California, USA: ACM, 2015, pp. 23–26.

[12] A. Mametjanov, P. Balaprakash, C. Choudary, P. D. Hovland, S. M. Wild, and G. Sabin, "Autotuning fpga design parameters for performance and power," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2015, pp. 84–91.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016.

[14] Intel. (2016). Intel fpga sdk for opencl, [Online]. Available: `http://www.altera.com/`.

[15] D. Koeplinger, C. Delimitrou, R. Prabhakar, C. Kozyrakis, Y. Zhang, and K. Olukotun, "Automatic generation of efficient accelerators for reconfigurable hardware," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 115–127, Jun. 2016.

[16] D. Liu and B. C. Schafer, "Efficient and reliable high-level synthesis design space explorer for fpgas," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2016, pp. 1–8.

[17] M. Malik, F. Farahmand, P. Otto, N. Akhlaghi, T. Mohsenin, S. Sikdar, and H. Homayoun, "Architecture exploration for energy-efficient embedded vision applications: From general purpose processor to domain specific accelerator," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2016, pp. 559–564.

[18] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on fpgas," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16, Dresden, Germany: EDA Consortium, 2016, pp. 918–923.

[19] Xilinx. (2016). Vivado high-level synthesis ug902, [Online]. Available: `http://www.xilinx.com/`.

[20] Q. Yanghua, N. Kapre, H. Ng, and K. Teo, "Improving classification accuracy of a machine learning approach for fpga timing closure," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 80–83.

[21] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: A high-level performance analysis tool for fpga-based accelerators," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2016, pp. 1–6.

[22] Y. Choi and J. Cong, "Hlscope: High-level performance debugging for fpga designs," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2017, pp. 125–128.

[23] Y.-k. Choi, P. Zhang, P. Li, and J. Cong, "Hlscope+,: Fast and accurate performance estimation for fpga hls," Nov. 2017, pp. 691–698.

[24] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated hardware optimization tool," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec. 2017, pp. 1–8.

[25] M. Makni, M. Baklouti, S. Niar, and M. Abid, "Hardware resource estimation for heterogeneous fpga-based socs," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17, Marrakech, Morocco: ACM, 2017, pp. 1481–1487.

[26] N. Wulf, A. D. George, and A. Gordon-Ross, "Optimizing fpga performance, power, and dependability with linear programming," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 3, 23:1–23:23, Jun. 2017.

[27] C. Xu, G. Liu, R. Zhao, S. Yang, G. Luo, and Z. Zhang, "A parallel bandit-based approach for autotuning fpga compilation," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17, Monterey, California, USA: ACM, 2017, pp. 157–166.

[28] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18, Monterey, CALIFORNIA, USA: ACM, 2018, pp. 269–278.

[29] (), [Online]. Available: `https://github.com/Xilinx/xfopencv`.

[30] (), [Online]. Available: `https://cryptography.gmu.edu/athena/index.php?id=CAESAR_source_codes`.

Curriculum Vitae

Sara Ogburn is currently a candidate for Master of Science in Electrical and Computer Engineering at George Mason university. She obtained her Bachelor of Science in Electrical and Computer Engineering, with a concentration in embedded systems and a minor in computer science, from George Mason University in 2014. She began working towards her master's degree in the fall of 2015. She currently holds a position as an embedded software engineer at Argon ST (A Boeing Company), which she obtained upon graduation with her bachelor's degree and has held full-time throughout the graduate school process.