A Machine Learning Approach for Area Prediction of Hardware Designs from Abstract Specifications

Elena Zennaro[1,3], Lorenzo Servadei[1,4], Keerthikumara Devarajegowda[1,2], Wolfgang Ecker[1,3]
Infineon Technologies AG[1] - University of Kaiserslautern[2] - Technical University of Munich[3]
Johannes Kepler University Linz[4]
Email:<Firstname.Lastname>@infineon.com

***Abstract*—Advancements of Machine Learning (ML) in the field of computer vision have paved the way for its potential application in many other fields. Researchers and hardware domain experts are exploring possible applications of Machine Learning in optimizing many aspects of hardware development process.**

In this paper, we propose a novel approach for predicting the area of hardware components from specifications. The flow uses an existing RTL generation framework, for generating valid data samples that enable ML algorithms to train the learning models. The approach has been successfully employed to predict the area of real-life hardware components such as Control and Status Register (CSR) interfaces that are ubiquitous in embedded systems. With this approach we are able to predict the area with more than 98% accuracy and $600x$ faster than the existing methods. In addition, we are able to rank the features according to their importance in final area estimations.

***Index Terms*—Machine Learning, Design Productivity, Area Estimation, Meta-Modeling, Register Interface, Code Generation, Model-Driven-Architecture**

I. Introduction

The *design gap* in hardware industry is a well-known issue and is the result of continuous scaling in the technology domain. Product design can be largely improved using modern modeling and optimization techniques. The key question to ensure an optimal design is what is desirable about the design. When dealing with SoC design several aspects can be considered as optimization targets, e.g. dimension, speed, cost, power consumption, etc. Ideally, a multi-objective optimization [14] of these features is necessary to ensure a fabrication that is constantly at the forefront and aligned with market requirements.

To fill the productivity gap, advanced computational sciences can be the cutting-edge choice. Machine Learning, which is a branch of Artificial Intelligence (AI), is an important area of research with several promising opportunities for innovation at various level of hardware design. ML algorithms learn from examples and try to identify the structure of a system. In this sense, ML methods play a crucial role in extracting meaningful information from complex structured data and are able to ensure fast and accurate performances.

In order to cope with the complexity of hardware design optimization, we address the problem by starting from the estimation of one of the target features, that is the area of an hardware component. To achieve this goal, we resort to different Machine Learning algorithms to perform the prediction. The novelty here illustrated relies upon an RTL generation framework to collect data samples from abstract specifications, combined with the use of ML algorithms, trained on the collected data, to estimate the area.

The rest of this paper is structured as follows. Section II presents an overview of the state of the art as regards hardware design optimization and Machine Learning approaches in semiconductor design and manufacturing. Section III describes the approach adopted to create data collection of hardware components and the Machine Learning algorithms used to provide an estimation of the area starting from abstract specifications. Section IV delves into a real-life application of the proposed approach. A discussion on the results of our proposed approach on estimating the area for register interfaces and a brief summary concludes the paper.

II. Related Work

SoC design has been widely explored in the last decades. Several approaches can be considered in order to optimize SoC features. In [24], a *convex optimization* approach is used to optimize the performance of Real Time Chip Multiprocessors (CMPs). Convex optimization is shown to be very efficient in an early stage design exploration, in order to deal with the choice of the area of the individual components of the chip.

Machine Learning algorithms are adopted in several scenarios in the semiconductor manufacturing field. For instance, ML and power estimation is the main focus of paper [17]. Through regression trees, a powerful and straight-forward ML algorithm, the research group is able to combine high-level design features and low-level soft-processor parameters, for an accurate prediction over power and performance at the same time. In [23], *Singular Value Decomposition* (SVD), a matrix factorization algorithm for feature reduction, is used to retrieve principal components, a lower number of feature combination which are more meaningful to predict the power usage. *Principal Components Analysis* (PCA), a linear feature reduction method, is exploited also in [16]. The authors use a combined approach of PCA and evolutionary algorithms for a size optimization problem. PCA accelerates the search of an

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00076

optimized design of analog and radio-frequency circuits. By means of PCA, the authors propose a feature selection which provides the necessary and relevant information for a robust design size reduction. The results show that they are able to find a set of parameters leading to an almost 3x reduction of the circuit hypervolume.

ML techniques are often employed to detect faults and anomalies. In [12], several ML methods are exploited to detect faulty wafers in semiconductor manufacturing. To detect hotspot regions in a system, *i.e.,* regions where circuit failures are more likely to happen, in [22] a deep learning approach is addressed. Particular attention is given to the tuning of the hyperparameters of the CNN architecture, in order to establish an efficient hotspot detection. The use of ML techniques to identify intermittent failures in post-silicon validation is proposed in [4]. The authors explore both *supervised* and *unsupervised* ML algorithms and show how a clustering approach can be more efficient to solve the problem of post-silicon bug diagnosis.

With recent advances in ML and its power to extract meaningful information from complex structure, many research works address the use of ML approaches to handle the complexity of semiconductor manufacturing framework. However, the application of ML techniques for hardware design components optimization is still in its early stages.

III. Approach

Our approach for area estimation of hardware designs using *Machine Learning* algorithms is divided into two phases: *data collection* and, *training and testing*.

- *Data collection*: This phase involves collecting data samples from abstract specifications and synthesis reports, required for training the Machine Learning models. In other words, we first generate the RTL code for a given specification through an automation framework; synthesize the RTL design for a given FPGA board and extract the number of configurable logic blocks[1] (CLBs) information from the synthesis report.
- *Training and testing*: In this phase, appropriate parameters from synthesis reports and abstract specifications for training the Machine Learning models are selected. After collecting a reasonable number of data samples, appropriate ML algorithms are applied for area estimation. The training (model learning) and testing (model evaluation) are carried out simultaneously until learning curve flattens out. Finally, we use ML models to estimate the area for new abstract specifications without the need to generate and synthesize RTL code.

[1]Configurable logic blocks are fundamental building blocks of programmable arrays consisting of multiple logic cells. The number of CLBs required to realize the given design function directly represent the area requirements of that design function.

The RTL generation framework is a pre-requisite in order to generate multiple data samples needed for ML. The underlying generation framework allows to generate anomalies, corner cases and hence ensures robustness of the learning models. In III-A, we describe the RTL generation framework used to collect data samples to train ML models. The rationale behind selecting different ML algorithms for area estimation and a brief description of individual approach is outlined in III-B.

A. RTL generation framework for collecting data samples

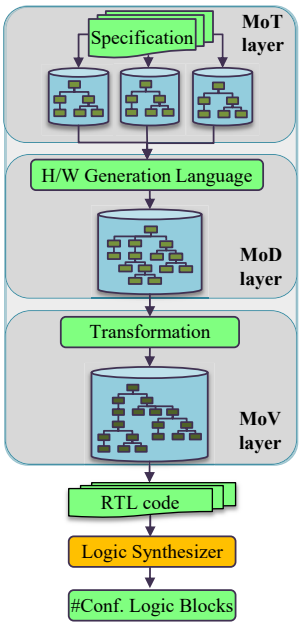

Fig. 1: RTL generation flow for collecting data samples.

At Infineon, we use a well established automation framework based on meta-modeling for code generation [9], [10]. Object Management Group (OMG), a non-profit organization for standard software flows, recently updated a code generation flow called Model-Driven-Architecture (MDA) [20]. The existing meta-modeling framework is being enhanced to adapt OMG's MDA principle for code generation. Although the adapted approach for RTL generation is already published in [18], [19], we provide a brief summary of the same for explaining the need for RTL generation framework.

Our approach for RTL generation relies on series of model-to-model transformations before generating the view files in a specific target language. The approach followed is drawn in Figure 1. The different layers of the flow are described as follows:

- *MoT Layer*: The first layer of our approach is MoT layer and involves capturing the informal specification according to formalized, abstract meta-models called Meta-Model-of-Things (MMoT). The meta-model defines the high level elements, their relations, their properties and in addition, the constraints and semantics for the models.

An instance of the MMoT is Model-of-Things (MoT), which represents the valid abstraction of the hardware object/structure. Valid instances of the MMoT enable architectural exploration of a given hardware function. An example of a MMoT is shown in Figure 3, which represents the meta-model of a register interface. MoTs provide an outside view of the requirements and abstract away the implementation details (micro-architecture).

- *MoD Layer*: The intermediate layer includes transforming MoT into a more concrete model called Model-of-Design (MoD). The transformation is realized through a hardware generation language also referred to as Hardware Domain Specific Language (HDSL), which is coded in Python. We refer to these transformations as Templates-of-Design (ToD). This Python based HDSL extracts the abstract specification from MoT and defines a more concrete model that holds the micro-architecture of the intended hardware function. The MoD in turn is an instance of its meta-model that holds all available components and also outlines the semantics of the MoD. ToD are programmed to MMoT and hence, it is possible to realize MoD for any valid instance of the MMoT.
- *MoV Layer*: A final model-to-model transformation of MoD into Model-of-View (MoV) is realized through another transformation layer. MoVs are platform specific and can be viewed as abstract syntax tree of the specific target language from which the view files (ex: RTL code in VHDL or SV) are generated.

After generating RTL code, logic synthesis tool is used to synthesize RTL code to a FPGA based board. From the synthesis report, we extract number of configurable logic blocks that directly depict the area requirements of the implemented hardware function. In addition, the generated RTL code is functionally verified by following an approach described in [5], [6]. The underlying automation framework allows generation of valid instances of a MMoT with minimal manual efforts. In addition, the framework enables random generation of valid instances of a meta-model. As already mentioned, ToDs are programmed to consider all valid instances of the MMoT and hence, RTL code is generated for any randomly generated MoT. In this fashion, for a given specification of a hardware function, multiple data samples are generated with the help of RTL generation framework. A specific example for collecting data samples, employing the described approach is outlined in IV.

B. Machine Learning Algorithms for Area Estimation

Machine Learning is a form of Artificial Intelligence (AI) that is able to perform a certain task without being specifically programmed for it. To achieve this, a ML model *learns* from previous examples related to the specific task during a process called *training*. In this subsection, we provide a general overview of the ML algorithms that are employed for area estimation. In order to provide an estimation of the area of a hardware component, a *regression problem*[2] has to be considered. The goal of regression is to predict the value of one or more continuous target variables given the value of a multi-dimensional vector of input variables [13].

To obtain robust and accurate results, we implement and compare two *ensemble methods*: *Random Forest* and *Gradient Boosting*. Ensemble methods are meta-algorithms that combine several ML techniques into one predictive model [7]. *Random Forest* (RF) algorithms became of popular interest because of their abilities to capture complex interaction structures in the data. The driving principle of a random forest is to build several estimators independently and then average their predictions [2]. In detail, it builds a large collection of decorrelated trees and then averages them. Finally, the prediction is performed by a voting decision among the ensemble of trees. On the other hand, with *Gradient Boosting* (GB) algorithms, base estimators are built sequentially in order to combine several weak models to produce a powerful ensemble [11]. We adopt a GB approach, named *Gradient Boosting Tree* (GBT), in which decision trees are used as weak-learners. In this way, we are able to actively generate complementary base-learners by training the next learner boosting on the mistakes of the previous learners. The choice of RF and GB is related to the internal structure of ensemble methods. Indeed, algorithms which average out multiple regressors are prone to achieve better generalization performances. Furthermore, the possibility to use decision trees as a feature selector method provides meaningful insights on the importance of features for the prediction.

As an alternative, we consider *Multilayer Perceptron* (MLP), which corresponds to the simplest case of *Feedforward Artificial Neural Network* (FFNN) in which, each node is a neuron that uses a nonlinear activation function [15]. Feedforward neural networks provide a general framework for representing nonlinear functional mappings between a set of input variables and a set of output ones. Indeed, from a theoretical point of view, as demonstrated in the *universal approximation theorem* [21], an FFNN with a single hidden layer is able to approximate continuous functions on compact subsets of $\mathbb{R}^m$. As shown in equations (1) and (2), for any continuous function f and $\epsilon > 0$, there exist a monotonically increasing and bounded continuous function φ, $n \in \mathbb{N}$, number of neurons in the hidden layer, constants $v_i, b_i \in \mathbb{R}$, and parameters $\beta_i \in \mathbb{R}^m$, for $i = 1, \ldots, n$, such that

$$F(x) = \sum_{i=1}^{n} v_i \varphi(\beta_i^T x + b_i) \tag{1}$$

has the property

$$|F(x) - f(x)| < \epsilon, \quad \forall x \in I_m, \tag{2}$$

[2]In Machine Learning, a regression problem is defined as a task in which *quantitative* outputs are predicted, *i.e.,* outputs are real values/continuous variables. Differently, in a *classification problem*, *qualitative* outputs are predicted, *i.e.,* outputs are discrete values/categorical variables.

where I_m is an m-dimensional unit hypercube. This theorem highlights the power of a simple MLP: it is able to represent a wide variety of functions, when using appropriate parameters. Visually, an MLP can be represented by a graph in which the input layer is made of a number of perceptrons equal to the number of input variables, while the output layer has as many neurons as the number of output variables. The *learning rule*, such as *Stochastic Gradient Descent, Nesterov Momentum, etc.*, method that improves the neural network's performance, is a quasi-Newton algorithm. This algorithm updates the weight parameters of the network. In this paper, as learning rule we consider the *Limited-memory BFGS* method, a second-order algorithm that does not require the exact computation of the Hessian matrix, but directly considers a sparse approximation of its inverse [25]. Thanks to this procedure, the L-BFGS algorithm allows to obtain a fast convergence. This method guarantees fewer learning epochs w.r.t. first order methods (e.g. Stochastic Gradient Descent). However, due to computational constraints, it is usually feasible only to small datasets. The selection of MLP as a forecasting algorithm is related to the final purpose of the area prediction, which is a general design optimization. Neural Networks are prone to be used in multi-objectives optimization, since they are provided with a differentiable loss function and offer the possibility to be implemented in a composite fashion, with other Neural Networks models.

In order to ensure a robust and statistically relevant evaluation of the models, we split the model selection step from the model evaluation. The two different processes are performed through a *nested cross-validation* [3]. The cross-validation is performed by splitting the dataset into different folders (*k-folds cross-validation*). After that, one folder is used for the testing phase, while the remaining are involved for the training process. These steps are repeated until each folder, one after the other, has been used as a test set. Results of training and testing validations are then averaged out among all the repetitions done. Additionally, in nested cross-validation, a first outer k-fold cross-validation loop is used to split the data into training and test folds, and an inner loop is considered to select the model via k-fold cross-validation on the training fold. After model selection, the test fold is then used to evaluate the model performance. This approach allows to select the proper hyperparameters during the inner loop phase, as well as the possibility to evaluate the performance for each model in the outer loop phase, keeping the two operations separated.

To ensure a fine tuning of the hyperparameters of a model, two optimization techniques can be considered: the *random search* and the *grid search* [1]. Both approaches explore the same space of parameters. Even if the run time for random search is drastically lower, its performances are slightly worse. On the other side, it would be too computationally expensive to search over the many different parameters simultaneously using grid search. For this reason, we resort to a combination of the two: first a random search to narrow the range of the values for each parameter, afterwards a grid search based on the values provided by the previous phase. We apply this procedure to tune the hyperparameters only for the RF model. Since this algorithm is usually subjected to a consistent number of hyperparameters, a simple grid search would be too computationally expensive by itself. As we show in Section V, this approach allows to obtain the necessary trade-off between the feasibility of the hyperparameters and the accuracy of the model.

IV. Application

The approach described and proposed in Section III is employed for the area estimation of real application hardware components. For the same, we consider a regular component in an embedded system. Embedded systems in general constitute an embedded processor, memory element and a handful of peripheral devices. Timers/counters, interrupt controllers, serial ports to name a few, are regular components in any embedded system. Each of these components interacts with the software/program through Control and Status Registers[3]. Address, size and function/purpose of these CSRs are features of the peripherals that hold them [8].

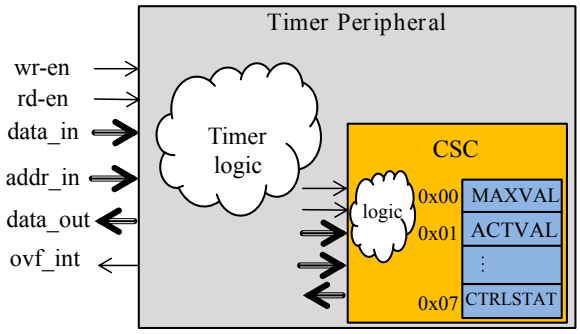

Fig. 2: Timer peripheral with CSR interface.

We implemented a timer component as per the approach described in III. A simplified timer peripheral device is as shown in Figure 2. A timer can be used for diverse applications: as a watchdog timer for resetting CPU; for counting external events; to provide time-base for asynchronous communication devices; for raising interrupts at specific time interval and many more. The state of the timer is represented by the state of its registers such as *MAXVAL*, *ACTVAL*, *CTRLSTAT* and *CCUVAL*. Each register is readable and writable by the program. As the registers are common in all peripheral devices and there needs be a standard mechanism for accessing them, we built a *Control Status Configuration* (CSC) that can be integrated as sub-component into any peripheral device. As shown in Figure 2, CSC sub-component represents the register interface of the timer device. As described in section III, the first task of the RTL generation framework is to create a MMoT that represents the abstract components of a register

[3]CSRs are also referred to as Special Function Registers (SFRs)

interface. The meta-model of the register interface (simplified) is as shown in Figure 3.

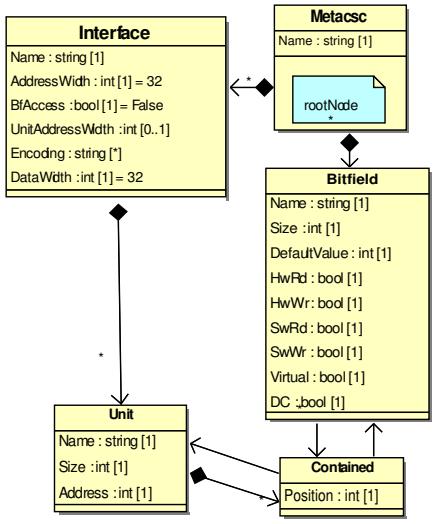

Fig. 3: Meta-model of a register interface (simplified).

Metacsc represents the root node of the meta-model and is composed of a list of bitfields and interface features. *Bitfield* class represents the properties of each bitfield contained in the register interface. Each bitfield has a default value, size and access rules (*HwWr, HwRd, SwWr, SwRd*). In addition, a bitfield can be *virtual* to the interface and DC represents if the bitfield is protected against un-authorized/unnecessary modifications. The *Interface* class in turn defines the properties of the interface such as data-width, address-width and whether each bitfield is addressable individually (*BfAccess*). *Interface* class is composed of *Unit* class that acts as a wrapper for bitfields in which the position of the bitfields is represented by *Contained* class. As all valid instances of this meta-model are valid MoTs, we setup a random MoT generator script[4] that creates valid MMoT instances by considering necessary semantics of the hardware component. RTL code is generated for each MoT, following the 3-layer approach depicted in Figure 1, after which logic synthesis reports are generated. Data samples are collected from MoT (abstract specification) and corresponding synthesis reports for training ML learning models.

Selecting input features and output parameters: We consider a multiple output regression problem where we wish to predict the $Y_1, \ldots, Y_K$ *outputs* (also called *responses*) from the *inputs* (usually named as *features*) $X_1, \ldots, X_p$. To build the dataset we start from the parameters of the MoT. Accordingly, LUTs and Slice Registers of the CSC are the outputs to be predicted, while the set of features is related to bitfield aspects and is reported in Table I. We consider $N = 319$ samples which means that the set of training data is given by $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, where each $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$ is a vector of feature measurements for the i-th case. Therefore, we use a dataset which has $p = 10$ features and $N = 319$ samples. The amount of the generated samples is chosen according to the flattening of the learning curve of the selected algorithms (i.e. adding other samples would not improve the algorithm accuracy with the chosen hyperparameters).

TABLE I: Features from MoT and output parameters.

Dataset	
Features	Total Contained Size
	Total Bitfields Size
	No. of Bitfields
	No. of Units
	No. of HwRd
	No. of HwWr
	No. of SwRd
	No. of SwWr
	No. of Virtual
	No. of DC
Outputs	No. of LUTs
	No. of Slice Registers

The features for the regression task are extrapolated from the MoT. The MoT is structured so that it contains relevant information for the CSC component, which can be used for the LUTs and Slice Registers forecast. Two different groups of features can be identified in the CSC MoT: *register specific features* and *bitfield specific features*. The former considers the CSC structure from a macro-level perspective: the *Total Contained Size* describes the amount of bits as a sum of all bitfield references inside each unit, the *Total Bitfields Size* aggregates the size of all generated bitfields, and finally the *No. of Units* provides the number of units inside the CSC. The latter instead is specific to the single bitfield structure: the *No. of HwRd* captures how many bitfields can be read from peripheral devices, the *No. of HwWr* indentifies how many bitfields can be written from peripheral devices, the *No. of SwRd* describes how many bitfields can be read from CPU instructions and the *No. of SwWr* tells how many bitfields can be written from CPU instructions. Furthermore, the *No. of Virtual* aggregates how many bitfields can be virtual the CSC, whereas the *No. of DC* describes the number of bitfields which have a protection mechanism bit.

V. Results and Discussion

In this section we present results obtained by applying the three ML algorithms described in Section III-B to estimate the area of the CSC in terms of number of LUTs and Slice Registers. The ML algorithms are implemented by means of *Scikit-learn* and *Scipy* Python 3.x libraries. In the first part, we analyze the search of the optimal parameters for each model, while in the second part, we evaluate the performance of each model by means of the *accuracy score*.

[4]Random generator is a Python script that makes heavy use of underlying automation framework for generating valid meta-model instances with minimum manual efforts

Hyperparameters: The tuning of the hyperparameters is performed in the inner loop of the nested cross-validation and by means of a grid search the best model is selected. For the inner loop a 4-fold cross-validation is used for all the three algorithms.

As regards the RF algorithm, the tuning of the following hyperparameters is considered:

- *number of trees* in the forest;
- *maximum depth* of the tree;
- *maximum number of features* to be considered when looking for the best split;
- *minimum number of samples* required to be at a *leaf node*;
- *minimum number of samples* required to split an internal node.

As previously mentioned in Section III-B, as a first step, a random search is carried out to narrow the range of the values for each hyperparameter. Afterwards, the inner loop 4-fold cross-validation provides the optimal hyperparameter values (see Table II) by means of a grid search. This combined optimization procedure allows to obtain the proper trade-off between the number of trees of the forest and the maximum depth of each tree. A fine tuning of these two parameters is necessary to deal with *bias-variance* trade-off [2]: deeper trees reduce the bias, while more trees reduce the variance. As we show in the next subsection, we are able to avoid huge overfitting of the RF model in this way.

TABLE II: Random Forest hyperparameters.

Parameters	Values
No. of Trees	400
Max. Depth	120
No. of Features	3
Min. Samples split	2
Min. Samples leaf	1

To select the proper GBT model, a *multi-output regressor* (MOR) is considered with different GBT as *estimators*. In order to build the parameter grid for the multi-output regressor we consider the following GBT regressors as estimators:

- GBT regressor with *number of trees* $= 1000$ and *learning rate* $= 0.01$;
- GBT regressor with *number of trees* $= 1000$ and *learning rate* $= 0.04$;
- GBT regressor with *number of trees* $= 2000$ and *learning rate* $= 0.01$;
- GBT regressor with *number of trees* $= 10000$ and *learning rate* $= 0.04$.

The *learning rate* parameter represents a weighting factor that shrinks the contribution of each trees that is added sequentially in the series. Since new trees are created to correct the residual errors in the predictions from the existing sequence of trees, the GBT can lead to a quickly overfitting of the training dataset. For this reason, a proper trade-off between number of trees and learning rate is necessary in order to avoid huge overfitting of the model. In the inner loop 4-folds cross-validation, the grid search provides the best GBT estimator (among the 4 proposed) in terms of numbers of tress and learning rate (see Table III).

TABLE III: Gradient Boosting hyperparameters.

Parameters	Values
No. of Trees	1000
Learning Rate	0.04

Since the restrained dimension of the dataset, a simple MLP with one hidden layer is sufficient to achieve satisfactory results. The tuning of parameters is performed only on the *hidden layer size* hyperparameter and the grid search is restricted to hidden layer of dimension that ranges from 1 to 12 neurons. In order to ensure robustness of the algorithm, a n_{iter}-loop is instantiated: at each iteration, the nested 4-folds cross-validation with grid search is performed and the best model, in terms of optimal value of the hidden layer size, is computed. We run the algorithm with $n_{iter} = 30$ and at the end, we average the obtained values for the hidden layer size. With this procedure we obtain an optimal value of 10 neurons in the hidden layer. The MLP model we consider is depicted in Figure 4.

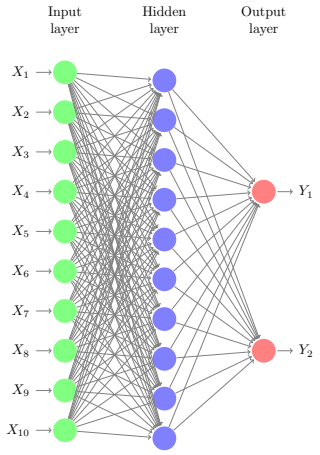

Fig. 4: Multilayer Perceptron.

Models Performance: As previously explained in Section III-B, after the inner cross-validation, an outer k-folds cross-validation is necessary to evaluate the performance of the models obtained from the parameter optimization in the previous phase. Also for the outer cross-validation the number of folds is set to 4 for all the three algorithms. In this second phase, the performance of each model is evaluated by means of the *coefficient of determination* R^2 which is defined as

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{3}$$

where SS_{tot} represents the total sum of square, which is proportional to the variance of the data, while SS_{res} corresponds to the residual sum of squares. For each algorithm the nested cross-validation R^2 score is computed and obtained results are shown in Table IV. As we can notice, all the three algorithms are able to reach a coefficient of determination of at least 0.95.

TABLE IV: Model accuracy.

Models	R^2 scores
Random Forest	0.979
Gradient Boosting	0.951
Multilayer Perceptron	0.983

However, by comparing train and test scores on each split of the cross-validation, substantial differences on the behaviors of the three algorithms can be observed. These results are reported in Figure 5, 6 and 7. From these plots, it is clear that the RF and the GBT algorithms still overfit slightly the dataset, even after several regularization steps. A possible reason to explain these behaviors is related to the tuning of the hyperparameters. For the RF and GBT the search space of the parameters results to be larger and more complex and this leads to a more challenging fitting of the learning function. In addition, more common regularization methods (e.g. depth trees reduction, decrease in the number of features) [2] have failed in decreasing the overfit. On the other side, a simple neural network, characterized by a relatively confined hyperparameter space, is sufficient to ensure accurate prediction with limited computational effort.

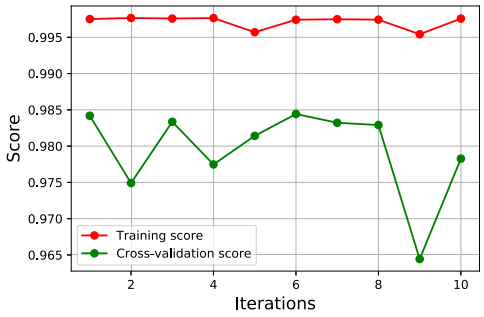

Fig. 5: RF training vs testing scores.

To conclude the result section we provide an analysis of the importance of the features and relate it to hardware design aspects. RF algorithms provide a straightforward method for feature selection, which is named *mean decrease impurity* [2]. Every node in the decision tree represents a condition on a single feature. Thus, when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure. Figure 8 shows the use of the RF algorithm to evaluate the importance of features on the regression task. It is noticeable how the *No. of Units*, the *Total Contained*

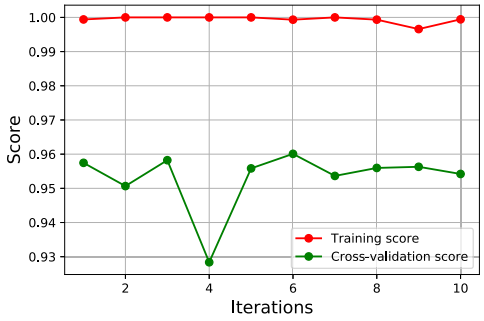

Fig. 6: GBT training vs testing scores.

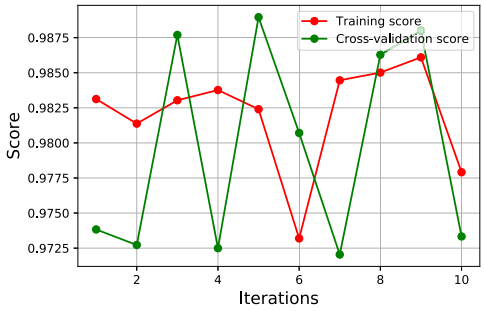

Fig. 7: MLP training vs testing scores.

Size and the *Total Bitfields Size* are the most important factors which affect the area prediction. These results are reasonable since the number of units together with the total dimension of the bitfields are what influence more the final dimension of the register interface. Indeed, if we consider different mapping algorithms to link bitfield to units, a higher number of units is correlated with a larger dimension of the register interface, and this can give a reasonable explanation for the register interface dimension. A similar explanation can be given for the *Contained Size* and the *Total Bitfields Size*: larger CSCs contain a larger contained size, which depends also on the total size of the bitfields. The aggregate size of the bitfields is indeed predominant w.r.t their number. The rest of the features occupies at most one bit per bitfield, for this reason, it does not appear to be too relevant for area estimation purposes (as this is irrelevant to the dimensions of containeds and bitfields).

VI. Conclusion and Future Work

In this paper we introduced a novel approach to estimate the area of hardware components from specifications. In this paper we have proved that Machine Learning techniques and statistically generalizable model evaluations are a promising approach to ensure fast and reliable performances and can help for a better understanding of hardware design features.The novelties of our approach rely upon the combination of an a RTL automation framework to create a data collection from abstract specifications and ML algorithms to perform the prediction. Precisely, we generate the RTL code for a given specification for several alternatives and then synthesize

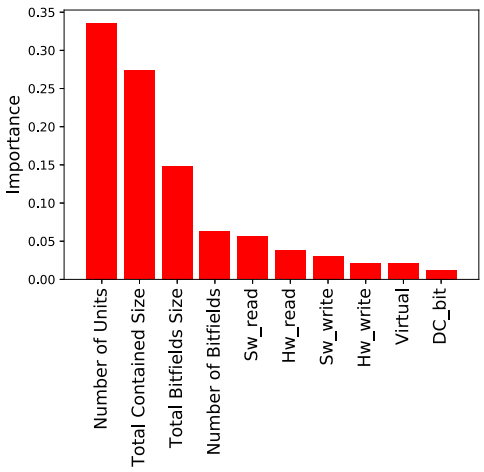

Fig. 8: Features Importance.

the RTL design in order to extract the number of CLBs. Afterwards, an estimation of the area in terms of CLBs is performed by means of ML algorithms. In this way, trade-off analysis of implementation alternatives can be done much faster than doing repeatedly synthesis runs and at a very high level of precision (approx. 99%).

Three ML algorithms have been considered to predict the area of a RI component. In order to ensure robustness of the results for a real and concrete task, particular attention has been given to the model performances. Moreover, we studied the effectiveness of the hyperparameters tuning to make the architecture more feasible for the specific task. From the results obtained, it is noticeable how the MLP model is able to outperform RF and GBT.

As next step, we are undertaking area estimation of several sub-components of a SoC, together with trade-off analysis between different objectives such as area, performance, and power consumption. The increase in complexity requires robust and accurate models able to cope with non-trivial hardware design aspects. In particular, when dealing with a larger parameter space, more complex ML algorithms and optimizers are necessary for further forecasting problems.

VII. Acknowledges

This work has been partially funded by the German "Bundesministerium für Bildung und Forschung (BMBF)" projects: COMPACT grant BMBF FE 01IS17028 and SAVE4I grant BMBF 01IS17032.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[2] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[3] Michael W Browne. Cross-validation methods. *Journal of mathematical psychology*, 44(1):108–132, 2000.

[4] Andrew DeOrio, Qingkun Li, Matthew Burgess, and Valeria Bertacco. Machine learning-based anomaly detection for post-silicon bug diagnosis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 491–496. EDA Consortium, 2013.

[5] K. Devarajegowda and W. Ecker. On generation of properties from specification. In *2017 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 95–98, Oct 2017.

[6] K. Devarajegowda, J. Schreiner, R. Findenig, and W. Ecker. Python based Framework for HDSLs with an underlying Formal Semantics. In *Proceedings of the 36th International Conference on Computer-Aided Design*, ICCAD '17, New York, NY, USA, 2017. ACM.

[7] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[8] Wolfgang Ecker, Volkan Esen, Thomas Steininger, and Michael Velten. *HW/SW Interface*, pages 95–149. Springer Netherlands, Dordrecht, 2009.

[9] Wolfgang Ecker, Michael Velten, Leily Zafari, and Ajay Goyal. Meta-modeling and code generation - the infineon approach. In Wolfgang Mueller and Wolfgang Ecker, editors, *MeCoES - Metamodelling and Code Generation for Embedded Systems: Workshop with ESWEEK*, pages 1–4. http://adt.cs.upb.de/mecoes/MeCoES2012-Proceedings.pdf, 2012.

[10] Wolfgang Ecker, Michael Velten, Leily Zafari, and Ajay Goyal. The metamodeling approach to system level synthesis. In Gerhard Fettweis and Wolfgang Nebel, editors, *DATE*, pages 1–2. European Design and Automation Association, 2014.

[11] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[12] Dongil Kim, Pilsung Kang, Sungzoon Cho, Hyoung-joo Lee, and Seungyong Doh. Machine learning-based novelty detection for faulty wafer detection in semiconductor manufacturing. *Expert Systems with Applications*, 39(4):4075–4083, 2012.

[13] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[14] Patrick Ngatchou, Anahita Zarei, and A El-Sharkawi. Pareto multi objective optimization. In *Intelligent Systems Application to Power Systems, 2005. Proceedings of the 13th International Conference on*, pages 84–91. IEEE, 2005.

[15] Sankar K Pal and Sushmita Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on neural networks*, 3(5):683–697, 1992.

[16] T. Pessoa, N. Lourenço, R. M. Martins, R. Póvoa, and N. Horta. Enhanced analog and rf ic sizing methodology using pca and nsga-ii optimization kernel. pages 1–4, March 2018.

[17] Adam Powell, Christos Savvas-Bouganis, and Peter Y. K. Cheung. High-level power and performance estimation of fpga-based soft processors and its application to design space exploration. *J. Syst. Archit.*, 59(10):1144–1156, November 2013.

[18] Johannes Schreiner and Wolfgang Ecker. Digital hardware design based on metamodels and model transformations. In *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*, pages 83–107. Springer, 2016.

[19] Johannes Schreiner, Rainer Findenig, and Wolfgang Ecker. Design Centric Modeling of Digital Hardware. In *IEEE International High Level Design Validation and Test Workshop, HLDVT 2016, Santa Cruz, CA, USA, October 7-8, 2016*, pages 46–52, 2016.

[20] F. Truyen. The fast Guide to Model Driven Architecture.

[21] Halbert White. *Artificial Neural Networks: Approximation and Learning Theory*. Blackwell Publishers, Inc., Cambridge, MA, USA, 1992.

[22] Haoyu Yang, Luyang Luo, Jing Su, Chenxi Lin, and Bei Yu. Imbalance aware lithography hotspot detection: a deep learning approach. *Journal of Micro/Nanolithography, MEMS, and MOEMS*, 16(3):033504, 2017.

[23] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai, and Tin-Fook Ngai. Early stage real-time SoC power estimation using RTL instrumentation. pages 779–784, Jan 2015.

[24] Leonid Yavits, Amir Morad, Ran Ginosar, and U Weiser. Convex optimization of real time soc. *arXiv preprint arXiv:1601.07815*, 2016.

[25] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.