Quality Assessment of Generated Hardware Designs Using Statistical Analysis and Machine Learning

Lorenzo Servadei[1,2], Elena Zennaro[1,3],
Keerthikumara Devarajegowda[1,4], Wolfgang Ecker[1,3], Robert Wille[2]

[1] Infineon Technologies AG, Am Campeon 1-15, 85579 Munich, Germany
[2] Johannes Kepler University Linz, Altenbergerstraße 69, 4040 Linz, Austria
[3] Technical University of Munich, Arcisstraße 21, 80333 Munich, Germany
[4] Univ. of Kaiserslautern, Erwin-Schrödinger-Str. 1, 67663 Kaiserslautern, Germany
{name.surname}@infineon.com robert.wille@jku.at

Abstract. In order to continuously increase design productivity, engineers and researchers rely on automation frameworks for hardware design purposes. This does not only guarantee an easier implementation of components, but creates a larger margin for improvement by generating design variants. Within this framework, a major problem for optimizing the generated design is retrieving data from which a prediction function (e.g. area, speed, power consumption) could be learned correctly (since a complete generation, i.e. synthesis of the hardware design, is too computationally expensive to be performed for a wide set of variants). In particular, the data used for learning the prediction function should be representative of valid design possibilities and be generated in an efficient way. As one contribution, this paper describes how *Statistical Analysis* (SA) and *Machine Learning* (ML) are used to guarantee the quality of the data. At the same time, its retrieval should avoid time consumption and manual effort. Therefore, this paper also proposes an automatic approach to generate representative and valid configuration samples both to improve the efficiency and to avoid manual effort during the retrieval. To point out this concept, we implement the generation of data for the estimation of the area of a *Register Interface* (RI) component. The proposed methods, implemented through SA and ML, allow to supervise the correctness of the generated data and the learning process itself. As a consequence, given the correctly generated data, the process of learning the RI area through a data-driven ML algorithm guarantees a still accurate ($R^2 = 0.98$) but $600x$ faster estimation.

Keywords: Machine Learning · Statistical Analysis · Data Generation · Design Automation

1 Introduction

In order to cope with the increasing complexity in hardware and system design, a common approach is to rely on higher levels of abstractions. For this reason, model-based design flows utilizing modeling languages such as UML [10] or, more specifically, SysML [2] become of interest. Through multi-layer abstractions and structure modeling, they allow to define dependencies, relations and constraints

for the components. This allows e.g. for verification and validation at early stages of the design flow (using e.g. methods such as [4, 5, 9]), but also to realize the desired system from well defined high-level models. In particular, this approach is very useful for generating hardware designs that differ in some implementation details but rely on a common source (represented in terms of an abstract model).

In the following, we focus on MetaRTL [11], an approach followed at Infineon for design centric modeling of digital hardware. However, the methods proposed in this paper are independent of any specific design flow and shall be applicable to other design flows as well. The MetaRTL design flow allows the automatic generation of hardware designs from a given abstract model (in the following called *meta-model*) and, by this, provides an agile environment which aids designers in the generation of the hardware components and systems.

However, the generation mechanism itself is a challenge, since the generation framework must be resilient, robust and easily maintainable. Indeed, while the meta-model provides a basis for the designs generation, the designer still has to provide a corresponding *configuration*, i.e. a set of parameters guiding the design generation process and defining how the meta-model is instantiated/realized. For example, from an abstract description (meta-model) of the CPU structure, the designer has to select a specific configuration for a CPU instance (e.g. No. of cores, etc.). The complexity of today's hardware components makes it hard to determine a proper set of configurations which indeed yields a design satisfying given objectives e.g. with respect to area, speed, etc. This raises the question how to efficiently determine instances, from a given meta-model, which eventually trigger the desired design generation.

In order to retrieve an optimal design configuration for a given objective, the designer needs to be equipped with a set of different configurations as well as estimates about the resulting area, speed, etc. of the design obtained from each of these configurations. While the same could be achieved manually (implementing a set of different configurations and, through a synthesis tool, obtaining the corresponding area, speed, etc.), this would result in a time consuming process. Moreover, the estimates would be prone to errors since, as long as not a "real" implementation would be created for each configuration, the designer would need to infer those characteristics from the design itself. Hence, an automated and reliable way of exploring the space of design configurations is needed.

In this work, we address these problems. First, we automatically generate values for the design configurations from a given meta-model of a RI, a common hardware component which regulates the data-transfer between CPU and peripheral devices. Then, we utilize methods of *Statistical Analysis* (SA) and *Machine Learning* (ML) for evaluating the structure of the generated data. This is done through an analysis of the features correlation, features clustering and Mutual Information (MI): that captures statistical relatedness and commonalities of correctly as well as incorrectly generated data. By training a ML algorithm on correctly generated data, we are able to maximize the prediction accuracy ($R^2 = 0.98$) [14] and improve $600x$ the speed for obtaining the RI area. Overall, this yields a set of configurations, including corresponding estimates, for the generation of hardware designs – addressing the shortcomings summarized above. As a final remark, this whole process is obtained in an automatic manner.

Experimental evaluations confirm the benefits of the proposed method. While, thus far, designers mainly determined the needed configurations by experience, the proposed approach provides them with a powerful tool which automatically suggests proper configurations, and learn successively a correct objective prediction (e.g. area, speed). Moreover, the SA and ML allows to hint to designers

characteristics of the hardware components which were not obvious in the first place just by manually checking out the single design instances.

The rest of this paper is structured as follows. Section 2 reviews the applied hardware designs generation flow based on MetaRTL and motivates the problem considered in this work. Section 3 then presents the proposed method and outlines how datasets of hardware design configurations are generated, and how methods of SA and ML are applied for analyzing their quality and correctness. Finally, Section 4 summarizes the obtained results and Section 5 concludes the paper.

2 Background and Motivation

With the purpose of keeping this work self-contained, this section briefly reviews the automation framework *MetaRTL*, which is considered in this work as a representative of a model-based approach for generating hardware designs starting from meta-models. Afterwards, we outline the main challenge of the corresponding hardware designs generation flow, namely how to efficiently and correctly obtain hardware configurations compliant to the requirements of the design (w.r.t. the complementary metrics such as area, speed, etc.). Determining which configuration indeed satisfies imposed constraints for the desired objectives is a non-trivial task.

2.1 The MetaRTL Design Flow

The MetaRTL framework (originally proposed in [1]) provides an environment for hardware design generation. MetaRTL allows to define abstractions and properties for each hardware component (e.g. the Register Interface, the Timer etc.), and generates the design based on a chosen configuration. The automation and abstraction of the design process leads to an increase in productivity and to a rapid work-flow.

The MetaRTL generation-flow follows the three level *Model Driven Architecture* (MDA) abstractions, and is optimized for supporting the hardware generation. The first layer, called Model-of-Things (MoT), captures the abstract formalized configurations, which corresponds to the *Computation Independent Model* (CIM) in the original MDA description. The second layer, called Model-of-Design (MoD) corresponds to the *Platform Independent Model* (PIM) and defines the micro-architecture for the given hardware specification. This is the core model for MetaRTL, as it defines the hardware architecture. The third layer, namely the Model-of-View (MoV) corresponds to the *Platform Specific Model* (PSM) and is the least abstract model with a mapping to the target view. It can be conceptualized as a Abstract Syntax Tree (AST), defined by an abstraction of the target low-level design implementation.

2.2 Open Problem: Determining the Desired Configuration

Using the MetaRTL flow described in Section 2.1, different hardware designs can be generated from the same meta-model. To this end, a meta-model is defined first in the MoT layer which provides a *generic* description of the system and/or application to be realized. With this respect, the problem we address in this paper is how to correctly and automatically map configurations of hardware

designs (i.e. instances of the meta-model) to output predictions on a learned function. In order to achieve this, we develop methods for generating potential design configurations and inspect their statistical properties. In the remainder of this work we describe this process, as well as our contributions for improving it, by means of the following running example.

Example 1. We consider a *Register Interface* (RI) application as example. To this end, a meta-model (as shown in Figure 1) is created. This defines dependencies and relations between the RI sub-components. The meta-model is composed of four main classes, namely the *Interface*, the *Unit*, the *Bitfield* and the *Contained*. The *Interface* class contains all the properties of the same, including most importantly the *DataWidth* and the *AddressWidth* of each Interface. Each *Interface* can contain one or more *Units*, i.e. logic entities which group the *Bitfields*. Each *Unit* is indeed related to one or more *Bitfields* by means of the *Contained* component, which are pointers to the Bitfields and allows to decouple them to a single *Unit* usage. The *Bitfields* are the minimal information set that we can read and write at once in the RI, and real center of the meta-model.

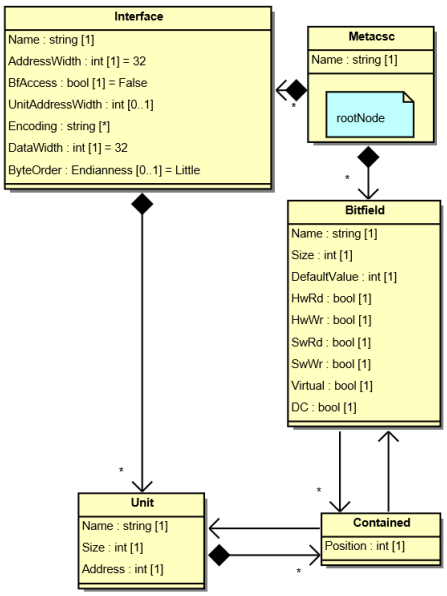

Fig. 1: Dependencies and Relations among RI Components

With the meta-model available, the designer can instantiate different hardware realizations by setting parameters, which we denote as *input values* (or *feature values*) for the prediction. With *features* we indicate instead each attribute to which the input values are referred. The respective parameters are thereby usually restricted through explicit constraints for avoiding completely invalid results. Furthermore, the respectively chosen values will eventually affect the quality of the generated hardware designs e.g. with respect to area, speed, etc.

Table 1: Features from MoT and output parameters.

	Name	Constraints
Features	Containeds Size (X_1)	$0 \leq x_1 \leq x_4 \cdot \mathrm{u}$
	Bitfields Size (X_2)	$0 \leq x_2 \leq x_4 \cdot \mathrm{u}$
	No. Bitfields (X_3)	$0 \leq x_3 \leq x_4 \cdot \mathrm{u}$
	No. Units (X_4)	$0 \leq x_4 \leq \mathrm{m}$
	No. HwRd (X_5)	$0 \leq x_5 \leq x_4 \cdot (\mathrm{u}/2)$
	No. HwWr (X_6)	$0 \leq x_6 \leq x_4 \cdot (\mathrm{u}/2)$
	No. SwRd (X_7)	$0 \leq x_7 \leq x_4 \cdot (\mathrm{u}/2)$
	No. SwWr (X_8)	$0 \leq x_8 \leq x_4 \cdot (\mathrm{u}/2)$
	No. Virtual (X_9)	$0 \leq x_9 \leq x_4 \cdot \mathrm{u}$
	No. DC (X_{10})	$0 \leq x_{10} \leq x_4 \cdot (\mathrm{u}/2)$
Outputs	No. LUTs (Y_1)	
	No. SRs (Y_2)	

Example 2. Consider again the meta-model of the running example shown in Figure 1. Instances of this meta-model are restricted by constraints as shown in Table 1, e.g. the total number of instances of the component *Unit* is limited by the value of the maximum number of Units, denoted with m, while the size of the Unit is identified by u. This allows to reduce the single feature space and adapt the generator to realistic use cases. Each feature may eventually affect the generate designs with respect to area, speed, etc. For the area prediction of the RI component, we enlisted as outputs of the prediction Look Up Tables (*No. LUTs*) and the number of the Slice Registers (*No. SRs*) generated in the synthesis process. These values in fact determine the surface of the implemented RI component.

Now, the challenge is how to properly determine the values for these features. Although, as stated above, the configurations values define the quality of the generated design e.g. with respect to area, speed, etc., their relation is often not obvious. More precisely, the designer is faced with the question, how to determine a set of possible configurations which eventually allow to optimize the desired objectives (e.g. instantiating a realization with a certain area, power, etc.). This problem has often a non-trivial solution: by increasing the number of features, a higher number of combinations has to be considered. Some of these will be considered as valid (e.g. *No. SwRd* + *No. HwRd* > 0 etc.), some of them are falling out of the valid configurations subset (e.g. *No. SwRd* + *No. HwRd* $= 0$, etc.). Finally, enlarging the features space may introduce non-obvious effects, complicating the interdependence in the data.

In the following, we propose a solution to this problem. In order to ease the corresponding descriptions, we thereby focus on determining proper configurations which generate designs satisfying an area prediction task (further objectives such as speed can then be addressed in a similar fashion). As an output for the prediction problem, we forecast the amount of LUTs and SRs generated in the synthesis run. More specifically, in order to optimize objectives of the design configurations (e.g. chip area, cost, speed, etc.), we need a set of generated data, from which the prediction function can be determined. For these data to be a valid input, they should be compliant to defined constraints and present specific statistical relations among their features. In fact, as we generate data sharing

the same constraints for each dataset, these properties could be observed and offer a glimpse on the correctness of the configurations. Furthermore, these conditions allow to learn, through a data driven approach, an accurate, robust and representative approximation of the true design values. This is of high interest for preventing the designer from selecting erroneous settings and carry them to a further implementation process. This could indeed escalate in wrong designs and poorly evaluated manufacturing products, which would lead to high costs and expensive re-design solutions.

3 Proposed Solution

Hardware Synthesis tools are able to take as input the hardware design generated from MoT files, and output the number of Configurable Logic Blocks (CLBs) and the implementation of the intended hardware function. From the synthesis report, the area of the hardware design is identified by number of LUTs and SRs that constitute the CLBs. This approach has two main drawbacks for optimization purposes. First of all, it does not explicitly return the function to be optimized. Second, it requires manual effort and computation power, as the process implies the synthesis of the hardware implementation of the whole platform. ML algorithms can overcome these issues, by learning to map the design configurations to the desired prediction. This approach needs nevertheless correctly generated data to learn from. Without those, it is impossible to have a reliable data-driven algorithm. To this end, in this section we propose ML and SA as viable methods for supervising the automated generation of data, structuring a robust and rapid first step for an optimization work-flow. This approach overcomes the synthesis run for retrieving the area of the design and outlines a prediction function for performing this task.

3.1 The Problem of Area Forecast

In this paper, we generate configurations as input data for a supervised learning problem, in the form of a multiple regression. The output variables are the *No. LUTs* and *No. SRs*, indicated as Y_1 and Y_2. The predictors, usually named *features*, are denoted with $X_1, \ldots, X_{10}$ and are retrieved from the MoT configurations. The dataset is composed of $n = 319$ MoTs data samples and can be represented as $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, where each $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)})^T$ is the vector of feature measurements for the i-th case. The amount of generated data is dependent on the flattening of the learning curve of the algorithm for the area forecast: this shows that additional samples would not enhance the prediction score of the ML model. For an in-depth description of the regression problem and ML algorithms applied, we refer to this paper [14].

3.2 Data Generation and Features Space Exploration

In order to create a robust generator, we establish a range of possible feature values given by the design required implementations. This means that the configurations have to match with the design experience on application use cases. As a consequence, the computational time for generating a representative sampling is reduced. The configuration boundaries used for this paper, referred to the RI design, are shown in the column *Constraints* of the Table 1. Even if the constraints restrict the space of search for the optimal RI area, the number of

possible features combinations leads to a non-trivial problem. Our approach for randomizing the data generator is to sample, within the constraints of the RI, the Units and Bitfields properties. In addition to that, in each generation we apply one out of four mapping functions, inspired by possible design use cases, for setting the Bitfields inside the Units. The algorithms used to map the Units are the following:

- *Compact mapping algorithm*: dense Bitfields mapping inside each Unit;
- *Random mapping algorithm*: random number of Bitfields mapped inside each Unit;
- *Basic mapping algorithm*: one Bitfield mapped inside each Unit;
- *Combined mapping algorithm*: jointly use of Basic and Compact mapping algorithms.

3.3 Analysis of the Generated Data

The automated data generation is a fast and flexible way to retrieve input data for any data-driven algorithms. This process nonetheless has some caveats. In fact, the generation of wrongly constrained data is pernicious for an accurate area forecasting. Further than that, the dataset generated for the learning model, in case of non-trivial problems, should represent extensively the values of the features provided. After building a data generator, in this paper we address thus this problem: we present several approaches for testing the data-generator and evaluating the correctness of the generated dataset. In our case the data generator is implemented through a python script which samples, as aforementioned, from a constrained features space.

The first step of the proposed solutions consists in measuring the correlation among features for the generated datasets. This value indicates how much features are dependent on each other, giving an intuition on how a feature value changes according to other ones. In order to achieve this, we iteratively plot the features values of the dataset, two at times: it results that the correlation among features is mainly monotonic, i.e. either the variables increase in value together, or as one variable value increases, the other variable value decreases. According to these considerations, we compute the Spearman's Rank Correlation Coefficient (SRCC) on the predictors, two at a time. The SRCC, denoted with ρ, expresses the relation between the distance d between the feature components x_i and y_i and the number of samples n. This coefficient provides an overview of the features correlation inside the RI, and can be computed, if the ranks are distinct integers, as

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{1}$$

where:

- d: pairwise distance of the ranks of the observations x_i and y_i of the variables X and Y;
- n: number of samples.

As a second support for the identification of wrong constraints in the data generator, we apply an agglomerative clustering algorithm, a subset of the family of hierarchical clustering. This approach, from a data-driven perspective, reconstructs the context of the given features, by grouping them together. This

procedure allows to match the designer's prior information about the features used with the clusters retrieved in the data. As a first step, we exclude the presence of outliers after performing an analysis on the generated design features. After an evaluation of different linkage methods on our data, we decide to adopt the average linkage clustering. In fact, we aim to find averaged statistics that match the given design knowledge on the meta-model structure and this linkage performs a meaningful clustering on the data. In the average linkage clustering, the distance between clusters is computed as the mean distance among elements (i.e. the mean of the distance $d(x, y)$, corresponding to the modulus of the difference of the two elements x and y) of each pair of clusters in the dataset, as shown in the following equation

$$D(H, G) = \frac{1}{k_G \cdot k_H} \sum_{x \in G, y \in H} d(x, y) \tag{2}$$

where $d(x, y)$ represents the distance between x and y, which belong to clusters G and H, respectively.

- $d(x, y)$: distance between $x \in G$ and $y \in H$;
- G, H: clusters;
- k_G, k_H: No. of elements, respectively in clusters G, H.

Starting from a single cluster per feature, the algorithm iteratively merges them until the desired groups number Q, set previously by a specific hyperparameter, is reached. As a criterion for the agglomeration, the algorithm merges clusters where the mean distance between pair of elements is the least among all clusters at each cycle. The final outcome of the algorithm is a number of clusters Q which include T features, with $Q < T$.

As a further approach to the analysis of the generated data, we introduce a features selection algorithm for measuring how each feature influences the final prediction of number of LUTs and SRs. We use ranking algorithms to highlight the different importance of each feature for predicting the two output responses. The corresponding importance is computed by means of the MI. This metric, indeed, does not assume any monotonic relationship among features, but considers only the degree of their relatedness. The ML algorithm applied computes the MI from the K-Nearest-Neighbors (KNNs) statistics. This value comes from an iterative process applied to the independent variable X (e.g. *No. Units*, X_4) and the dependent output variable Y (e.g. *No. LUTs*, Y_1) in order to quantify the impact of X on the fluctuation of Y. In the paper [6], the KNNs distance for computing the MI among two continuous variables is calculated as the maximum distance between the projected datapoint x_i and y_i and the corresponding next point x'_i and y'_i on the subspaces X and Y. By means of the KNNs distance, and by introducing the hyperparameter k as a result of a grid search which considers the consistency of the model over permutations on the features, we are able to compute the MI $I(X, Y)$, as shown in the following equation

$$\begin{aligned} I(X, Y) = &\psi(k) - 1/k + \\ &- \left\langle \psi(s_x) + \psi(s_y) \right\rangle + \psi(n) \end{aligned} \tag{3}$$

where:

- $I(X, Y)$: MI between the variable X and the variable Y;
- ψ: *digamma function*, defined as $\Gamma(x)^{-1} d\Gamma(x)/dx$;
- $\langle ... \rangle$: average over all $i \in [1, \ldots, n]$;

- s_x: number of datapoints with $||x_i - x_j|| \leq \epsilon_x(i)/2$, where $\epsilon_x(i)/2$ is the distance from $z_i = (x_i, y_i)$ to the k-th datapoint projected in the X subspace;
- s_y: number of datapoints with $||y_i - y_j|| \leq \epsilon_y(i)/2$, where $\epsilon_y(i)/2$ is the distance from $z_i = (x_i, y_i)$ to the k-th datapoint projected in the Y subspace;
- n: number of samples.

3.4 Machine Learning for Area Forecast

We proceed, after the data generation and validation, into the area forecast problem. ML provides a set of algorithms for functions approximation: this has been often exploited in the hardware design literature for the forecast and optimization of power consumption [12], SoC performance [8] and area of chip components [13]. To this end, *parametric* ML algorithms have been often preferred, as they guarantee a fast and inexpensive computation of the predicted value (*inference*), once the approximated function has been learned [3]. Since the restrained dimension of the dataset, after comparing different ML algorithms (e.g. Random Forest, Gradient Boosting, Linear Regression) for the area forecast, we select a *Multilayer Perceptron* (MLP) as the best performing algorithm, as described in [14]. The MLP corresponds to the simplest case of *Feedforward Artificial Neural Network* (FFNN) in which, each node is a neuron that uses a nonlinear activation function [7]. FFNNs provide a general framework for representing nonlinear functional mappings between a set of input variables and a set of output ones. Thanks to the non-high number of data samples, the monotonic variables correlation and the restrained amount of features, we are able to perform a grid search by considering as hyperparameter the *No. Neurons* of the unique *Hidden Layer*, in a interval from 1 to 12. We perform a nested 4-folds Cross-Validation (CV), which provides a fine tuning of the hidden layer size hyperparameter together with a satisfactory and robust model accuracy.

In the algorithm, we average out the prediction and hidden layer size for 30 different models ($e_{models} = 30$). Each model performs a nested 4-folds CV for finding the best test scores (outer 4-folds CV), best parameters (inner 4-folds CV) and grid search over the hidden layers size (inner 4-folds CV). As a solver, after a comparative search, we apply a Quasi-Newton method, called L-BFGS. The algorithm is based on the BFGS recursion for the inverse Hessian matrix H. The L-BFGS method approximates the Hessian with a first order matrix with sparse vectors, in order to limit the usage of memory of the algorithm, as shown in [15]. As a regression score measurement, we adopt the *coefficient of determination* R^2, which provides a value for the *explained variance* of the output variable.

4 Results Evaluation

With the scope of evaluating the SA and ML pipeline, we generate a Baseline Dataset (BD) by implementing the aforementioned mapping algorithms and other three anomalous datasets of 200 samples each. These latter cases contravene the constraints imposed in the original random generator, and they gradually enhance the severity level of the anomalies. We consider the following anomalous datasets:

– ADI: dataset with an additional number of empty Units;

– *ADII*: dataset where all Bitfields created are not readable by hardware devices nor software;
– *ADIII*: dataset where one or more Bitfields are included in the same Unit several times.

The anomalous datasets bypass in an incremental way the set of constraints of the BD dataset: it is important to remark that the anomalies are present on the dataset level (population), as we create each dataset with predefined mapping and constraints. Thus, we focus on dataset statistics, and not on single design (individual) analysis. As a fist approach to the analysis of the generated data, we compute and plot the Spearman's ρ between features. The outcome of the analysis is a symmetric matrix. The SRCC is bounded in the interval $-1 \leq \rho \leq 1$, where $\rho = -1$ means a full-inverse correlation, $\rho = 0$ indicates an absence of correlation, and $\rho = 1$ corresponds to a full-direct correlation. In Figure 2 we plot the correlation values for the BD dataset. For the anomalous datasets, we only provide some comments on the changed coefficients, by observing their correlation matrix.

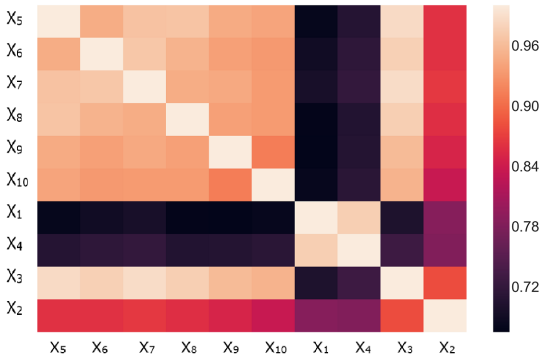

Fig. 2: Correlation Matrix, Spearman's ρ, Baseline dataset

As expected, we can observe here the intrinsic structure of the RI, in a statistical fashion. Indeed, the $\min(\rho^{BD}) = 0.68$ corresponds to the correlation between *Containeds Size* and *No. Bitfields*. This shows the decoupling in the RI between the Contained and the Bitfield component, so that a reference to a same Bitfield can be added in different Units several times. For the anomalous datasets, we point out the following principal changes. In ADI, all the Bitfields properties have a low correlation w.r.t. the *No. Units*, as in the dataset there exists an additional random number of empty Units. In the ADII, we notice that there are no correlation values between the *No. HwRd* and *No. SwRd* properties, whereas the *No. HwWr* and *No. SwWr* have tighter bound to the *No. Bitfields* ($\rho = 0.98$), as either the *HwWr* or the *SwWr* property needs to be True, for the Bitfield to be valid. Last, we observe a correlation interval decrease in ADIII ($0 \leq \rho^{ADIII} \leq 1$). In particular, the *Bitfields Size* and *Containeds Size* are fully uncorrelated with the *No. Units*, since Bitfields and Containeds are redundant in the Units ($\rho^{ADIII}_{(x_1,x_4)} = 0$ and $\rho^{ADIII}_{(x_2,x_4)} = 0$). The correlation between features clearly varies, according to the entity of the anomaly in the data.

As a second step for the features analysis, in order to reconstruct meaningful groups of features in the dataset, we use an agglomerative clustering ML algo-

Table 2: Features from MoT and output parameters.

	BD	**ADI**	**ADII**	**ADIII**
Cluster 1	X_3	X_1	X_6	X_3
	X_5	X_2	X_8	X_6
	X_6	X_3	X_9	X_7
	X_7	X_5	X_{10}	X_9
	X_8	X_7		
		X_8		
Cluster 2	X_1	X_4	X_1	X_5
	X_4		X_4	X_8
Cluster 3	X_2	X_9	X_5	X_4
			X_7	
Cluster 4	X_{10}	X_{10}	X_2	X_1
				X_2
Cluster 5	X_9	X_6	X_3	X_{10}

rithm with average linkage. The table shows the final clustering results of the algorithm. As number of final clusters to be outlined, we select $Q = 5$. The value of Q is chosen based on a features analysis of the RI meta-model since, from design experience, it is possible to retrieve 5 features groups in this component. As show in Table 2, for the BD, all features related to the Bitfield properties compose the $Cluster1^{BD}$. In the same way, other related entities concerning the Units (e.g. *Containeds Size* (X_1) and *No. Units* (X_4)) are kept in the same cluster ($Cluster2^{BD}$), whereas independent properties are set in a separate one-feature cluster. In the ADI, the *No. Units* (X_4) are separated from other clusters ($Cluster2^{ADI}$), whereas most of the Bitfield properties are kept together, as a result of the basic mapping algorithm and the presence of empty Units. With respect to the ADII, it is possible to observe the same cluster including the *No. HwRd* (X_5) and *No. SwRd* (X_7) features, as the corresponding properties *HwRd* and *SwRd* are both constantly set to False ($Cluster3^{ADII}$). In a similar fashion the *No. HwWr* (X_6) and *No. SwWr* (X_8) contribute to the same $Cluster1^{ADII}$, as the properties *HwWr* and *SwWr* are likely to be set to True. Finally, we observe how in the ADIII the *Containeds Size* (X_1) and *Bitfields Size* (X_2) are positioned by the algorithm in a same cluster ($Cluster4^{ADIII}$), whereas the *No. Units* (X_4) does not have any other agglomerated features ($Cluster3^{ADIII}$). This shows how the repetition of Bitfield references in the Containeds lets their relatedness to the Units diminish. The obtained results outline proximity and distances among features for the BD, ADI, ADII and ADIII, and show the different internal structure of the datasets. In particular we observe that, for the BD, the algorithm can retrieve correctly all the assumed clusters of features in the design, differently than for the anomalous datasets case.

As a final step of the features analysis, we evaluate the importance and ranking of each single feature w.r.t. the outputs of our regression by means of the MI score. In Figure 3 it is shown how each single feature is related to *No. LUTs*. The MI between predictors and the response variable *No. LUTs* is very high for the BD, with a $0.75 \leq MI^{BD} \leq 1.77$. In particular, *No. Units* and *Containeds Size* have the highest MI with the *No. LUTs* value. From a design experience point of view, this seems feasible. Observing the MI of the

ADI, the range is $0.60 \leq MI^{ADI} \leq 1.17$. In particular, features with high MI for the BD are not ranked similarly for the ADI (e.g. *No. Units* have the last ranking position for the ADI dataset). Concerning the ADII, it is noticeable, as by constraints, the absence of MI between *No. SwRd*, *No. HwRd* and the *No. LUTs*. Furthermore, the features ranking appears different and the total MI (MI_{tot}) decreases ($MI_{tot}^{BD} = 10.85$, $MI_{tot}^{ADII} = 4.22$). Finally in ADIII, because of the different constraints of the dataset and anomalies produced, the total MI equals $MI_{tot}^{ADIII} = 1.2$, with about 9x reduction from MI_{tot}^{BD} w.r.t. the *No. LUTs*.

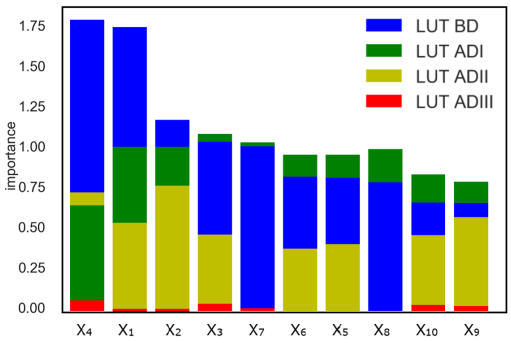

Fig. 3: MI on the No. LUTs

We evaluate as well the MI between each one of the features and the response variable *No. SRs*, as shown in Figure 4. Confirming the hardware design knowledge, the most important feature in the MI^{BD} ranking w.r.t. the *No. SRs* is the *Bitfields Size*, that is the aggregated size of all the bitfields present in the RI. The MI shared between the two variables is in fact $MI^{BD}_{(X_2,Y_2)} = 1.76$. As a second feature for importance, the *No. Bitfields* has a $MI^{BD}_{(X_3,Y_2)} = 1.43$; this ranking position matches as well with the designer knowledge. The $MI^{BD}_{tot} = 12.57$, whereas it results lower for the ADI, where $MI^{ADI}_{tot} = 10.52$. The MI^{ADI}_{tot} is furthermore distributed in a more uniform way, showing that the different degree of relatedness between $MI^{ADI}_{(X_2,Y_2)}$ and between $MI^{ADI}_{(X_3,Y_2)}$ is diminishing the relative importance of X_2 and X_3. The reason behind that is the additive number of empty units and the consequently lower bitfield density in the ADI. The two remaining datasets have a much lower MI_{tot}, respectively $MI^{ADII}_{tot} = 0.54$ and $MI^{ADIII}_{tot} = 0.17$. The lower values of MI w.r.t. the No. SRs are due to the very little information between the single feature and the output. This is particularly evident in the MI^{ADIII}_{tot} where, adding repetitively the same bitfield to a unit, the relatedness of the aggregated bitfield properties towards the units and the final value of No. SRs clearly decreases.

The results obtained show that features ranking and MI^{BD}_{tot} diverge deeply from the anomalous to the representative dataset. This serves as a valuable metric for assessing the quality and representativeness of the generated designs.

After ensuring the quality of the generated dataset, we implement the ML algorithm for the area prediction, as described in detail in [14]. As a first step, we compute a grid search for the MLPs network size over the e_{models}. The results

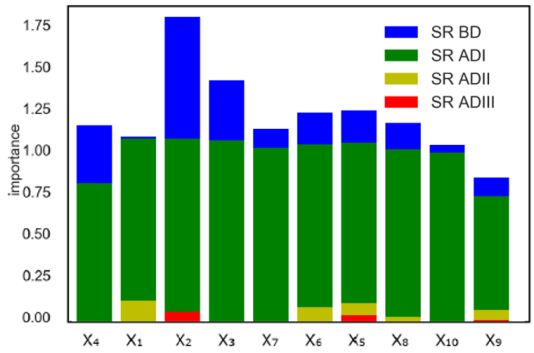

Fig. 4: MI on the No. SRs

of the search show $9 \leq$ *No. Neurons* ≤ 11. The R^2 averaged score of the nested CV (inner k-folds with $k = 4$, outer k-folds with $k = 4$) is $R^2 = 0.98$ in the test phase, which guarantees a satisfactory prediction score, as well as convergence with the training score and robustness. The Root of the Mean Squared Error (RMSE) in terms of LUTs, is 57, where $432 \leq$ *No. LUTs* ≤ 3246 and the mean $\mu_{No.LUTs} = 1456.7$. The RMSE in terms of SRs is 33, where $73 \leq$ *No. SRs* ≤ 857 and the mean $\mu_{No.SRs} = 432.3$. In the inference phase, which consists in the area prediction once the model is trained, the estimation is *600x* faster than the synthesis run for obtaining the RI area value.

5 Conclusion and Future Works

In this paper, we reviewed and analyzed ML and SA algorithms for supporting the process of automated data generation in the design configuration. Through the example of the RI, we could show how SA and ML can help in the correct learning of a mapping function to the RI area in a certain constraints boundary, and express useful metrics for pinpointing the validity and quality of the design settings. Indeed, the algorithms used are able to detect incorrectly generated datasets, which could lead to a skewed or invalid area prediction. This leads to a fundamental contribution to the ML area estimation algorithm which, through representative data, can forecast the RI area with $R^2 = 0.98$ and *600x* faster than the design generation - synthesis cycle. As a future work, we plan to increase the number of hardware components considered and approach further objectives through ML algorithms. This would determine additional dependencies, but also increase the potential of the proposed solutions. As the problem will grow in dimensionality, we think that ML and SA may be even more valuable methods for supporting hardware design configurations, objectives optimization and for understanding complex relations among design features.

6 Acknowledgements

This project was partially funded by BMBF and supported by ITEA under the umbrella of COMPACT.

References

1. W. Ecker and J. Schreiner. Introducing model-of-things (mot) and model-of-design (mod) for simpler and more efficient hardware generators. In *2016 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, Sept 2016.
2. Sanford Friedenthal, Alan Moore, and Rick Steiner. Omg systems modeling language (omg sysml™) tutorial. In *INCOSE international symposium*, volume 18, pages 1731–1862. Wiley Online Library, 2008.
3. Seymour Geisser and Wesley O Johnson. *Modes of parametric statistical inference*, volume 529. John Wiley & Sons, 2006.
4. Martin Gogolla, Fabian Büttner, and Mark Richters. USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*, 69(1-3):27–34, 2007.
5. Frank Hilken, Philipp Niemann, Martin Gogolla, and Robert Wille. Filmstripping and unrolling: A comparison of verification approaches for UML and OCL behavioral models. In *International Conference on Tests and Proofs*, pages 99–116, 2014.
6. Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.
7. Sankar K Pal and Sushmita Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on neural networks*, 1992.
8. 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.
9. Nils Przigoda, Christoph Hilken, Robert Wille, Jan Peleska, and Rolf Drechsler. Checking concurrent behavior in UML/OCL models. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 176–185, 2015.
10. James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.
11. 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.
12. 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.
13. Leonid Yavits, Amir Morad, Ran Ginosar, and U Weiser. Convex optimization of real time soc. *arXiv preprint arXiv:1601.07815*, 2016.
14. Elena Zennaro, Lorenzo Servadei, Keerthikumara Devarajegowda, and Wolfgang Ecker. A Machine Learning Approach for Area Prediction of Hardware Designs from Abstract Specifications. In *Proceedings of the 21st Euromicro Conference on Digital System Design*, 2018.
15. 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.