High Level Quantitative Hardware Prediction Modeling using Statistical methods

Conference Paper · July 2011

DOI: 10.1109/SAMOS.2011.6045455 · Source: DBLP

3 authors:

Roel Meeuws
Delft University of Technology
26 PUBLICATIONS **262** CITATIONS
SEE PROFILE

Carlo Galuzzi
Swinburne University of Technology
45 PUBLICATIONS **417** CITATIONS
SEE PROFILE

Koen Bertels
Delft University of Technology
319 PUBLICATIONS **3,650** CITATIONS
SEE PROFILE

High Level Quantitative Hardware Prediction Modeling using Statistical methods

Roel Meeuws, Carlo Galuzzi, Koen Bertels
Computer Engineering Lab, Delft University of Technology, The Netherlands
Email: {r.j.meeuws,c.galuzzi,k.l.m.bertels}@tudelft.nl

***Abstract*—With the increasing proliferation of heterogeneous and reconfigurable computing, it has become essential to have efficient prediction models to drive early HW-SW partitioning and co-design. In this paper, we present a high level quantitative prediction modeling approach that accurately models the relation between hardware and software metrics, based on several statistical techniques. The proposed approach generates models that predict hardware performance indicators for reconfigurable components, such as the number of slices, the number of flip-flops, and the number of wires. It utilizes automatic model selection, artificial neural networks, (logistic) regression, and data transformations. These models take a high-level language description as input, enabling hardware prediction in the early design stages. We calibrate the models for two sets of tools targeting Xilinx and Altera FPGAs, where we report, for example, and error of 14% for the number of multipliers in case of Xilinx and an error of only 18% for the number of wires in case of Altera. To provide a realistic evaluation, we validate the approach using 181 kernels, contrary to the majority of the existing techniques, which use libraries of tens of kernels at most.**

I. Introduction

Over the years, we have witnessed an increasing proliferation of heterogeneous and reconfigurable platforms in many computing system domains. By using specialized hardware for computationally intensive tasks, these platforms provide improvements in processing performance, power consumption and, in the end, costs. Additionally, reconfigurable components provide the flexibility needed in today's ever-changing markets by removing the need to develop new platforms for each application. However, the wide variety of components implemented in these platforms, together with their increasing complexity, create the necessity for tools that help the design and partitioning of applications over the different kinds of available components. Furthermore, with a long tradition and a broad user base of software development, these tools need to fill in the gap between software development and HW-SW co-design.

Given an application composed of different kernels, in order to map one of these kernels on the available resources, it is necessary to obtain information on factors, such as the power consumption, the speed-up, or the hardware resource consumption of the kernel for each individual processing element. At the early design stages, however, no implementations exist yet for these kernels. As a result, the designer may determine the necessary information by creating all the possible implementations for each kernel. *This can be exceedingly time-consuming, especially in the presence of reconfigurable hardware, where the hardware generation of many different kernels may take a few hours to a number of days.*

As a result, there is a clear need for tool support characterizing a functional description for different components in a heterogeneous platform. Quipu [15], a quantitative prediction modeling approach for early design space exploration, provides models that address this need. These models are able to predict the performance of kernels on reconfigurable components. They take a High Level Language (HLL) description (C code) as input and estimate area, interconnect, static power, clock period, and other FPGA-related measures. Although estimates are less accurate than the actual values, the time to obtain the former is several orders of magnitude smaller than the time to obtain the latter. As a result, the designers can quickly spot the overall effects of changes in their design, saving hours or even days per design iteration.

As a proof of concept, we integrated Quipu models into the Delft Workbench (DWB) [23], a semi-automatic tool platform supporting integrated HW-SW co-design targeting heterogeneous platforms containing reconfigurable components. The DWB addresses the entire design process from profiling, optimization, VHDL generation and compilation, up to the final evaluation. The produced Quipu models predict the hardware characteristics of the FPGA configurations generated by the DWB.

In this paper, we present a new high level quantitative prediction modeling scheme that accurately models the relation between hardware and software metrics, based on a number of statistical techniques. We added this scheme to the existing Quipu modeling approach. The main contributions proposed in this paper are the following:

- The demonstration of the ability of the proposed approach to generate comparable and appropriate prediction models for two independent tool-chains and platforms.
- The introduction of artificial neural networks, model selection, logistic regression, and data transformations in the prediction methodology. As a result, the proposed methodology exhibits an error ranging from 15% to 34%, which is an improvement of up to 45% compared to the results presented in [17] and [15].

This research is partially supported by the Artemisia iFEST project (grant 100203), the Artemisia SMECY project (grant 100230), and the FP7 Reflect project (grant 248976).

978-1-4577-0801-5/11/$26.00 ©2011 IEEE

- The detailed prediction of the interconnect resources distinguishing between different parameters, such as clock wires, logic wires, and power wires.
- The validation of the produced models by using a set of 181 kernels, contrary to the majority of similar techniques, which use libraries of tens of kernels at most. This is also an improvement over our previous work, where we used a set of 127 kernels only.

The remainder of the paper is organized as follows. First, in Section II, we review the related work and we establish the need for the Quipu modeling approach. In Section III, we discuss the Quipu modeling framework, its key components, and the statistical methods it employs. An discussion of model evaluation is presented in Section IV. Section V presents the experimental setup and methodology. The evaluation of the experimental results is presented in Section VI. Finally, Section VII concludes the paper and presents future work.

II. Related Work

Over the years, many hardware performance estimation schemes have become available. Part of these schemes drive low-level design processes, for example, in the placement and route phases. Others, including our work, operate on a HLL description such as C. Most of the high-level methods focus either on a specific application domain or on a specific kind of design. Other high-level methods are only applicable to certain types of platforms.

In [8], for example, the authors performed area-time estimation for the controller of a design. The input of their technique was a H-CDFG generated from C-code using the Trimaran compiler infrastructure. The authors reported prediction errors of 3.8% for flip-flops and 10.3% for slices. They validated their technique by using a set of *only* 10 kernels. Additionally, this estimation scheme did not provide estimations for the complete design, and it was strictly targeting the Trimaran compiler, which makes it not applicable in different contexts.

In the industry, we also see efforts to provide early design estimates. For example, Xilinx presented a resource estimation algorithm from a HDL specification, based on an estimation scheme that mimics the actual synthesis tool-chain [19]. The authors reported an average prediction error of 14.2% for flip-flops and 21.9% for slices. They verify their model with a set of 90 VHDL designs. The main problem with this scheme is its limitation to the Xilinx synthesis tool-flow. Furthermore, the scheme operates on VHDL code, and, as such, does not take into account the effects of C-to-VHDL translation. As a result, it is difficult to apply this approach in a broader spectrum of tools and platforms.

In [11], we find another estimation scheme targeting regular tasks without extensive control circuits. This scheme requires a DFG description of an algorithm, which is then summarized using various parameters, such as the number of certain operators and their bit-widths. The authors presented a set of custom equations for area and timing and they validate their scheme using *only* 6 kernels. Although the scheme exhibited a prediction error of approximately 12% for area and 29% for frequency, the small number of kernels makes this claim weak at best.

In [14], the authors presented an approach for FPGA area estimation from *SA-C*. This language is a C-dialect adapted for image processing. The idea of the paper was to build prediction models for each type of DFG node. Linear Regression (LR) was performed on a set of DFG nodes with varying parameters to determine the coefficients of these models. The authors reported an error of 5.3% based on *only* 4 kernels from the image processing domain. The estimation method that the authors presented is tightly coupled to the SA-C compiler that they use and, as such, cannot be easily recalibrated for another set of tools or platforms. As a result of the small validation set and the requirement of the SA-C language, this approach is not suitable for more general use during HW-SW co-design.

By using the popular Polyhedral model, [9] presented an estimation scheme targeting loop controllers. The authors established the effect of the number of statements and the nesting depth of a loop on their hardware generation scheme. Based on these values, they performed LR to predict the number of slices and the frequency. By using a set of *only* 12 kernels, they reported an error of 7.14%. As this paper focuses on loop controllers in a polyhedral model, the prediction model has a relatively narrow scope and cannot be used in a more general context.

In [10], the authors presented an area and power prediction method based on linear regression. The approach generates specific estimation models for each component, based on the parameters of the design. These models are targeted to the estimation of IP cores in larger designs. The paper uses a dataset of *only* 21 data points based on 5 IP cores to validate their model, although the models are generated using 50 different configurations for each IP-core. The authors report an error of 8% for the number of slices. The main difference between this approach and our approach is that we provide a generic model applicable to many kernels in an application, while the method proposed in [10] provides specific component models for individual IP cores only.

Another approach using statistical methods can be found in [6]. The authors present an area estimation scheme from SystemC descriptions. Their approach splits the modeling into two parts: the translation to VHDL and the low level synthesis. The final model is based on design parameters extracted from the SystemC description and the VHDL description. Principal Component Analysis is performed to prune the number of parameters. Subsequently, classic linear regression is performed on a set of 20 designs to generate the model. The paper validates the model by using *only* 5 designs. The error is measured for each different system component (FSM, MUXs, etc.) and, then, weighted to obtain a total error of 36.8% for the number of LUTs. By using only a small set of kernels to build and validate the models, this approach does not sufficiently substantiate their claims. Furthermore, Quipu differs from this approach as it considers ANSI C code as input instead of SystemC; the latter has far more low-level hardware details incorporated in the language.

Paper Reference	Object	Predicted hardware measures	Input specification	Error	Size of the validation set
[8]	Controller	flip-flops, slices, LUTs, delay	HLL (C)	10.3%	10
[11]	Entire design	slices, frequency	DFG (RTL)	12%	6
[14]	Entire design	LUTs	HLL (SA-C)	5.3%	4
[10]	IP-core specific	slices, power	Matlab	8%	21
[9]	Loop Controller	slices, frequency	Polyhedral model	7.14%	12
[6]	Entire Design	LUTs, flip-flops	SystemC specification	36.8%	5
[18]	Entire Design	unspecified area metric	FIR specification	19%	250
[19]	Entire design	flip-flops, slices, LUTs, etc.	(V)HDL	21.9%	90
Quipu	**Entire design**	**all above and more**	**HLL (C)**	**26**%	**181**

TABLE I
COMPARISON OF PREDICTION SCHEMES IN ERROR, VALIDATION, AND MODELED HARDWARE MEASURES.

As shown in Table I, all these works based their claims on the quality of their models in terms of error on validation sets containing between 4 to 12 kernels. *However, if the target is to report errors that are not biased to a small data set, a larger set of validation data, such as the one used in this paper, becomes vital.* Let's suppose, for example, one validates a model using a set of only 12 kernels. It is very unlikely that these kernels can represent the whole spectrum of possible kernels. For one, with such a limited set of kernels, the possibility of cherry-picking your validation set increases. Secondly, an anomalous kernel that is not well modeled by a certain model can adversely affect the validation error when one uses a small set of kernels.

In [18], we find one method in this direction. The authors introduce an estimation scheme to predict resource consumption of systems composed of FIR filters. These systems are composed of a set of 7 basic modules. For each of these modules a neural network predicts the resource consumption based on the module parameters. Also, a neural network is used to predict the system level resource consumption, based on the number of modules and on the sum of all estimated modules. The system estimates are based on 250 combinations of basic components. By using this data set an error of 19% for an unspecified area metric is obtained. These results are different from our approach in two main aspects. First, their approach is very specific to their FIR structures and cannot be used for a high level language description as our approach does. Furthermore, the dataset is formed by varying the different parameters of *only* a few basic components, instead of using actual different kernels.

In this paper, we present the Quipu modeling approach for prediction of hardware resource consumption targeting reconfigurable components. This method targets HLLs such as C, in order to drive early design space exploration and to provide models for different reconfigurable platforms and tool-chains. Contrary to existing similar approaches, we validate our approach with 181 *real* software kernels using models produced for the DWB tool-chain as a case study. In our previous work, we validated other Quipu models using 127 kernels. As a result, this earlier library was not well balanced over the different application domains and, as such, lead to models biased to the application domains that were over-represented. Finally, we validate the general applicability of our approach by providing models for different tools and platforms

III. The Quipu Modeling Framework

The work in this paper is presented in the context of the Quipu modeling approach, which we have developed and presented in [16], [17], and [15]. The approach is generic and not limited to any particular platform or tool-chain by allowing the generated models to be automatically recalibrated for different tools and platforms. This is in contrast with the majority of the existing techniques, which require manual reimplementation for different contexts. In this paper, we employ this approach to generate models for a combination of the DWARV C-to-VHDL compiler [25] and the Xilinx ISE Synthesizer for the Virtex4 platform [1]. Notwithstanding, Quipu can provide models for other combinations as well, such as, the Altera Stratix IV FPGA and the C-to-Verilog compiler from the Haifa University [3], [4]. Each specific model instance is calibrated using the output data of such a combination of tools and platform. Note that this set of calibration data can vary also depending on certain tool options. For example, area measurements will be much lower when optimizing for area compared to when optimizing for speed. Most options concerning high level optimizations can be accounted for by the *Quipu Metrication tool* (refer to Section III-B). In other cases, the user may want to consider generating models for each option value. It is obvious that this only applies to options that have a significant effect, as minor differences have no significant effect on the relatively large error at the early stages of development. Another approach would be to use some common option values as model parameters instead.

There were several limitations in the previous iterations of our approach. Firstly, the reported error was still relatively large in some cases. Secondly, the interconnect prediction was not able to distinguish between the different types of wires that co-exist in a design. Furthermore, the kernel library was not properly balanced, with some application domains represented by only two kernels, and others by tens of kernels. Finally, the portability of our approach to different sets of platforms and tools had not been shown.

In the following, we first define the models and the criteria. Subsequently, we discuss the key components of the Quipu approach. Last but not least, the newly developed techniques that constitute our approach are discussed in detail.

A. The Models and the Criteria

It is essential to quantify the characteristic aspects of the software description at hand, when we consider the modeling of hardware from software descriptions. In [17] and [15], we have introduced Software Complexity Metrics (SCMs) as a way to address this. SCMs are indicators of specific characteristics of the software code. Examples of SCMs are the number of operators, the number of loops, or the cyclomatic complexity, but also more complex metrics involving data-flow analysis or grouping operations in different types of functional units. Currently, we use a set of 58 SCMs as a base for our model.

To characterize hardware performance in terms of area, interconnect, power, and other parameters, we measure and predict 49 different hardware performance indicators. For instance, the number of slices, the number of wires, the number of LUTs, and the number of clock wires. Most of these parameters are related to interconnect and provide specific information on wires, nets, route-throughs, or switch-boxes, further subdivided for logic, power, or clock resources. Note that the power data was obtained by the Xilinx XPower Analyzer [24] and did not use simulation output for enhanced accuracy.

Given these criteria, we look for a model that describes the relation between hardware and software:

$$y_{HW} = g(\vec{\mathbf{x}}_{\mathbf{SCM}}) + \epsilon. \quad (1)$$

This is the theoretical optimal model relating some hardware metric y_{HW} to the SCMs $\vec{\mathbf{x}}_{\mathbf{SCM}}$ [1)] with the ideal relation $g(\cdot)$ and some error ϵ inherent to the problem at hand. In practice, an ideal model cannot be found. Instead, any modeling scheme is an approximation to some level. Therefore, we model the relation $g(\cdot)$ with an approximated relation $\hat{g}(\cdot)$ [2)]. This results in the introduction of some error $\hat{\epsilon}$ inherent to our approximation scheme:

$$\hat{y}_{HW} = \hat{g}(\vec{\mathbf{x}}_{\mathbf{SCM}}) + \hat{\epsilon}. \quad (2)$$

The approximation $\hat{g}(\cdot)$ can be, for example, an ad-hoc model, a Linear Regression Model (LRM), or an Artificial Neural Network (ANN). In case of LR techniques, $\hat{g}(\cdot)$ is a linear equation:

$$\hat{y}_{HW} = \hat{\mathbf{a}}\vec{\mathbf{x}}_{\mathbf{SCM}} + \hat{b} + \hat{\epsilon}, \quad (3)$$

where $\hat{a}$ is a vector of coefficients $\hat{a}_i$ corresponding to the element x_i of the set of SCMs $\vec{\mathbf{x}}_{\mathbf{SCM}}$ obtained for some kernel that correspond to the hardware metric $\hat{y}$, and $\hat{b}$ is the offset of the linear model. Note that these variables are stochastic variables. *This means that reporting a simple percentage error is not enough. As a result, the characterization of the error distribution must be addressed as well.*

B. The Tools and the Kernel Library

Quipu consists of a set of tools and a kernel library, as depicted in Fig. 1. In the modeling flow, Quipu extracts SCMs and hardware performance indicators from a kernel library. *This is a library of 181 kernels from a wide variety of application domains, contrary to many existing techniques, which use libraries of tens of kernels at most.* This allows us to build models that are generally applicable. It is also possible to build domain-specific models by using, for example, only the 57 Cryptography-related kernels out of the 181 kernels considered. An overview of the kernels in this library is provided in Table II. The 58 different SCMs can be extracted using the *Quipu Metrication tool*, which produces an XML file containing SCM measurements for each kernel. The metrication tool can measure metrics at source code level and also after performing different optimizations. In this way, we can provide useable input data for tool-chains that perform different levels of optimization.

Domain	Kernels	Size[a]	Bit-Based	Streaming	Control
Compression	12	47.6 (14-95)	x		x
Cryptography	57	192.2 (15-1107)	x	x	some
DSP	12	32.3 (10-110)	x	x	some
ECC	13	74.8 (10-496)	x	x	x
Mathematics	29	33.9 (5-100)			
Multimedia	42	81.8 (6-1107)	some	x	x
General	13	72.9 (22-163)			x
Total	181	102.6 (5-1107)			

[a]avg. size in number of statements (range).

TABLE II
OVERVIEW OF THE KERNEL LIBRARY, THE NUMBER OF KERNELS AND THEIR MAIN ALGORITHMIC CHARACTERISTICS IN EACH APPLICATION DOMAIN.

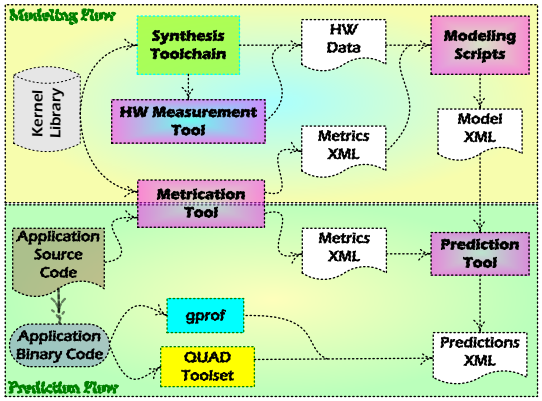

Fig. 1. An overview of the Quipu modeling approach, its tools (the boxes with thick border), and the accompanying tools (the boxes with dashed borders).

Another tool is the *Quipu Hardware Measurement tool*, which measures 49 different hardware parameters, such as area and interconnect, from synthesized hardware targeted at Xilinx FPGAs. The tool keeps track of all nets and components in the design by processing the XDL file generated by the Xilinx synthesis tool-chain. It provides detailed interconnect measurements, such as the number of clock wires or the number of power nets. For Altera FPGAs, such a tool is not

[1)]The arrow in $\vec{x}$ signifies a vector.
[2)]The hat in $\hat{g}(\cdot)$ signifies an approximation.

needed, as the report files generate detailed numbers on the interconnect usage automatically.

The gathered SCMs and hardware measurements are run through a set of modeling scripts that automatically evaluate different modeling techniques. The output model XML file can be used in the prediction flow, where, based on SCM inputs, the *Quipu Prediction tool* provides estimates of any required hardware aspects. All intermediate files in the prediction flow are saved in XML format for an easy integration. For example, the results of execution and memory profiling tools might be integrated as depicted in Fig. 1.

C. The Regression Techniques

In our previous works [17] and [15], we introduced Linear Regression Models (LRM), Generalized Linear regression Models (GLM), and Partial Least Squares Regression (PLSR) as the core techniques for the Quipu modeling approach. In the following, we discuss the regression techniques employed in this paper. In Section III-C1, we discuss the problem of collinearity and we address this problem by using stepwise model selection. Then, in Section III-C2, we show that non-linear relations between some SCMs and hardware parameters require data transformations to be applied effectively in regression analysis. After that, in Section III-C3, we introduce the use of advanced non-linear machine-learning methods for our prediction modeling approach. Finally, in Section III-C4, we address the specific problem with modeling small count values, like the number of multipliers, and suggest the use of logistic regression to improve the prediction of such values.

1) ***Stepwise Model Selection****:* The collinearity between the different SCMs in our model poses a problem. Due to the collinearity, some metrics measure more or less the same aspect of the code. This is problematic for regression analysis as certain aspects are now overrepresented. In [17], we proposed, by using Principal Component Analysis (PCA), to reduce the number of metrics. The drawback is that we don't control how the resulting metrics are composed. An original metric might partially influence the value of principal component values, although it does not affect the modeled dependent variable at all.

Another common approach to reduce the number of predictors (SCMs) is Stepwise Model Selection (SMS). Starting from a preliminary model, SMS successively adds and removes metrics step-by-step. At each step, the significance of each metric, given the current model instance, is calculated and the most significant variable is added to the model. After that, any variable that can be removed without increasing the error is removed. This procedure continues as long as there are possible steps that improve some quality criterion. When there are no more single variables that could be added or removed to improve this criterion the process stops. The resulting model is a local optimum. Better models might be found by using a brute-force algorithm, or a heuristic, such as simulated annealing. For our purposes, we used a stepwise heuristic using *Akaike's An Information Criterion (AIC)* [2] as the quality criterion. This criterion rewards goodness of fit (R^2) and penalizes overfitting.

In this paper, we perform the model selection only with respect to GLM models. The neural networks, which we also present, are evaluated separately. The model selection we employed, performs linear regression O(n) times and, thus, takes more time. However, the model calibration is performed only once, where the models themselves can be used indefinitely.

2) ***Data Transformations****:* During our analysis, we found out that in addition to the collinearity problem, many metrics also do not have a clear linear relation with the predicted hardware characteristics. Consequently, it is useful to transform the metrics, where possible, to better relate to the hardware characteristics. The *Box-Cox power transform* [5] is a widely used transformation of the data. This transformation preprocesses the data in order to reduce the variance of the dataset as well as to make the sample distribution more similar to the normal distribution. The transformation is defined as follows:

$$x_i^{(\lambda)} = \begin{cases} \frac{x_i^{\lambda}-1}{\lambda}, & \text{for } \lambda \neq 0 \\ log(x_i), & \text{for } \lambda = 0 \end{cases} \quad (4)$$

where x_i are the observations of the independent variables, λ is the estimator used to determine the transformation, and $x_i^{(\lambda)}$ is the transformed observation. By using a Box-Cox estimator in the *R statistical computing environment*, we determined that λ is close to zero for many predictors. Therefore, we considered log-transformations for all metrics in our model selection procedure. The transformed metrics were considered in all different modeling techniques employed in this paper.

3) ***Artificial Neural Networks****:* In the previous section, we have addressed the non-linear relation between individual hardware and software parameters. Nevertheless, the combined problem at hand remains non-linear to some degree. There are several techniques that tackle such problems. One well-known method of non-linear regression analysis is to fit Artificial Neural Networks (ANNs) [7]. An ANN is composed of a set of nodes that are arranged in layers. Each layer uses the outputs from the previous layers as inputs. Each node is a weighted linear combination of inputs. The value of this combination is transformed using an activation function passed on to the next layer:

$$net_j = f(\sum_{i=0}^{n} x_i w_{ij}), \quad (5)$$

where net_j is the output of node j, $f(\cdot)$ is the activation function, $x_i (i \in 0, \ldots, n)$ are the outputs of n nodes in the previous layers, and w_{ij} is the weight for the input from node i to node j. We use the ANN training package *nnet* from the *R statistical computing environment*. This package trains networks with a single hidden layer with possible skip-layer connections and nodes with a sigmoid activation function. Neural networks are prone to overfitting, when there are too many nodes in the network or the learning rate is too high. In

this case, the fitted model will be extremely accurate for the fitted data points, i.e. training data, but will not be usable for prediction purposes. In order to prevent this, we investigated many different configurations of these two parameters to find the model with the smallest prediction error.

*4) **Logistic Regression and Count Regression**:* Apart from collinearity and non-linearity, we found also that some of the hardware parameters that we consider, contain only non-negative values. In these cases, it makes sense to divert from the regular LR techniques and use GLMs. GLMs allow us to use error distributions other than the normal distribution. In case of discrete non-negative values, for instance, we might better model the data as a count. The error of such a model is more accurately modeled using a Poisson or Negative Binomial distribution. In [15], we have applied this technique to Quipu. However, count regression is not possible if a dataset contains many zeroes. For this reason, a decision model is needed that predicts whether a zero or a count is expected. Such a model can be fitted using Logistic Regression (LogR), where the odds for a one are modeled instead of the count. Assume y_i is binomially distributed as $B(n_i, p_i)$, i.e. there exists a bound set of values n_i, in this case zero and one. Then, LogR models the logit of the probability p_i for y_i to be 1 as follows:

$$logit(p_i) = log(\frac{p_i}{1-p_i}) = \hat{g}(\tilde{\mathbf{x}}_{\mathbf{SCM}}) + \hat{\epsilon}. \quad (6)$$

This logistic function of the probability of encountering a 1 for y_i is assumed to be linearly dependent on the SCMs. By using the inverse logit, we can predict the probability of each possible value. In our case, we can predict a zero in case the probability is closer to one and a count when it is closer to zero. In case a count is expected, we can use a count regression model based on only the count data, by using the Poisson or Negative Binomial distributions. A dataset where all zero values are removed, is called a *zero-truncated dataset*.

IV. Model Evaluation

In order to validate the predictive quality of statistical models, we need to perform cross-validation. This is the process of using part of the available data to calibrate the model and another part to determine the prediction error. The simplest form of cross-validation is the Holdout cross-validation [13], where the data are split into two sets: a training set and a validation set. When using a relatively small set of data points, this method has the disadvantage that the training set may not be representable for the calibration set. This leads to huge variance in the reported error depending on the data points that are selected for each set.

Another cross-validation method is the K-fold cross-validation [13]. In this method, the data is split over K subsets. Each subset is then used as validation set once, each time the remaining K-1 subsets are used collectively as training set. The average of the resulting error statistics can then be used as cross-validation error. Common instances of K-fold cross-validation are 10-fold cross-validation (K=10) and Leave-One-Out cross-validation (K=n). In this paper, we use 10-Fold cross-validation to validate our models, as suggested in [13] and [20]. The most common error summary statistic used in this type of cross-validation is the 10-fold cross-validated Relative Rooted Mean Square Error of prediction (RMSEp%):

$$RMSEp\% = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{\frac{1}{n}\sum_{i=1}^{n} y_i}. \quad (7)$$

This summary statistic captures the overall predictive performance of our model. Also the *goodness of fit* was determined by using the cross-validated coefficient of determination (R^2):

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \quad (8)$$

where y_i are the observed values, $\hat{y}_i$ are the predicted values, and $\bar{y}$ is the mean observed value. R^2 shows how much of the variance of the original dataset is explained by the model and not contributing to the error.

V. Methodology and Experimental Setup

A. Measurement

We compiled a kernel library of 181 C-kernels from many different application domains in order to evaluate our modeling approach. *The goal of using a diverse set of kernels is to build models that are generally applicable, and not restricted to a specific architecture as with existing approaches.* We collected 58 different SCMs for each of the kernels by using the Quipu Metrication tool. The collection process took 8.8 seconds on a dual-core Intel E8500 running at 3.16GHz. We obtained the VHDL for each kernel by using the DWARV 1.3.4 C-to-VHDL compiler. DWARV performs several optimizations on the input code, such as simplified scalar replacement, static single assignment, common sub-expression elimination, and dead code elimination. These optimizations were accounted for by using the correct optimization level in the metrication tool. Additionally, we generated Verilog for each kernel by using the C-to-Verilog compiler developed at the University of Haifa [4] and available via a web interface [22]. This tool uses optimizations such as resource sharing and pipelining.

We synthesized the VHDL designs by using the Xilinx ISE 11.5 targeting the Xilinx Virtex4 LX200 FPGA (xc4vlx200-ff1513-10). We collected 49 different resource measures out of the synthesized designs. These include, among others, slices, flip-flops, logic wires, and clock wires. All interconnect related measures for the Xilinx platform were determined using our *Quipu Hardware Measurement tool*, which is capable of tracking each individual net, wire, and pin in the design. We synthesized the Verilog code for each kernel by using the Quartus II 9.0 Build 235 synthesizer targeting the Altera Stratix IV EP4SE530 FPGA (EP4SE530F43C2ES). We collected 13 different resource measures from the generated bitstreams. These include, among others, ALUTs, registers, and total interconnects.

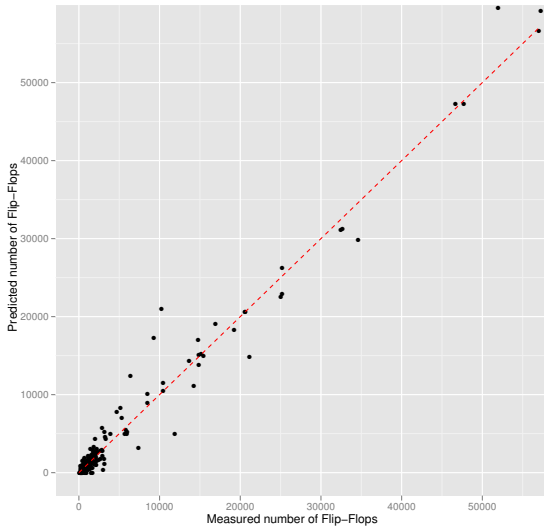

Fig. 2. The actual number of flip-flops vs. the predicted number of flip-flops in the DWARV/Xilinx case.

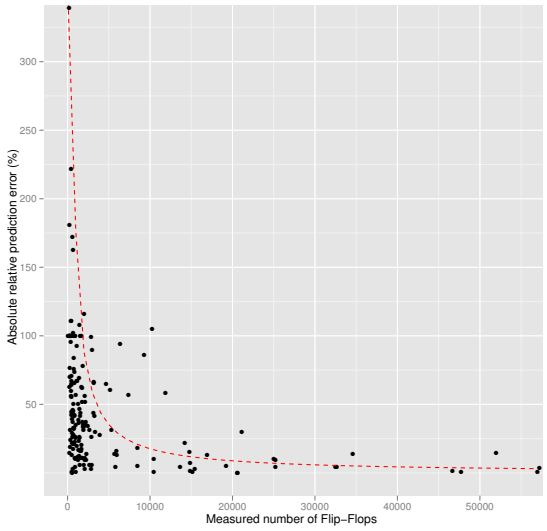

Fig. 3. The actual number of flip-flops vs. the relative prediction error of the total number of flip-flops in the DWARV/Xilinx case.

B. Model Calibration

After obtaining the necessary measurements, we performed stepwise model selection in the statistical analysis suite R by using the *stepAIC* method from the *MASS package* [21]. Different error distributions and data transformations were considered. Additionally, ANNs were trained in R based on the data using the *nnet package* [21]. The neural networks consisted of one hidden layer with varying numbers of neurons. The model with the smallest error was selected. Note that this process was performed automatically. The training of these models is a time-consuming procedure and it took about 6 hours on an 8-core Intel X5550 running at 2.67GHz with 8Mb cache and 32GB memory to calibrate the models for all hardware metrics. When the assessment of ANNs was omitted calibration took only 30 minutes. The model calibration was performed for both the DWARV/Xilinx flow and the C-to-Verilog/Altera flow. After the calibration of the models for a particular platform and tool-chain, the models can be employed efficiently and indefinitely. Note again that the calibration is a one time step.

VI. Results

A. Results for the DWARV/Xilinx case

We have generated models for 49 different hardware characteristics. The 8 most important ones are summarized in Table III. It is worth to note that the Rooted Mean Square Error of Prediction, as a percentage of the mean (RMSEp%), has improved by at least 3.4% for flip-flops and up to 45% for slices. For the models based on neural networks, we observe that the corresponding GLM model (RMSEp% (GLM)) accounts for the largest improvements from the old set of results. This implies that, after the model selection, the training of a neural network has marginally less effect. Still, the use of neural networks decreased the error by at least 0.7% for flip-flops and up to 7% for the number of LUTs.

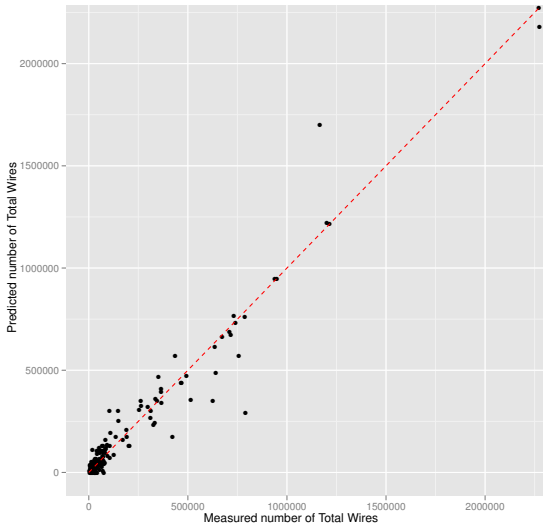

Fig. 4. The actual total number of wires vs. the predicted total number of wires in the DWARV/Xilinx case.

A special case is the model for the number of multipliers (DSP48 blocks). The data for the number of multipliers contained many zeroes, which polluted the regular modeling procedure. We built a binomial LRM on the binary decision variable $Z = Multipliers > 0$. The accuracy of this model was 99.43%. The dataset without the zero values (zero-truncated) was used to build a count model by using the negative binomial distribution. We were able to predict the number of multipliers within an error of *only* 14.76%.

Note that the error of 2.98% for power is also relatively low, although the R^2 indicates that less than 80% of the variance of the data is explained by the model. This model apparently fits the data substantially worse than the other models in Table III. Indeed, the percentage error is mainly small for these measures because there is a large constant offset in the model, which makes the percentual contributions of the error small.

Toolchain	Response	RMSEp%[15]	RMSEp%	MAPE	R^2	Model type	Topology [a]	RMSEp% (GLM)
Xilinx/ DWARV	Slices	71%	26.53%	42.87%	96%	Neural Network	29-11-1	31%
	Flip-flops	28%	24.64%	38.48%	98%	Neural Network	28-7-1	29%
	LUTs	67%	33.07%	57.78%	94%	Neural Network	30-5-1	40%
	Multipliers [b]	30%	14.76%	36.35%	98%	GLM (negative binomial)	n/a	n/a
	Total Nets	31%	27.54%	34.49%	92%	Neural Network	29-1-1	30%
	Total Wires	n/a	33.67%	48.44%	92%	Neural Network	31-11-1	37%
	Clock Wires	n/a	23.82%	31.56%	94%	GLM (normal)	n/a	n/a
	Power	n/a	2.98%	2.09%	74%	GLM (normal)	n/a	n/a
Altera/ C-to-Verilog	ALMs	n/a	16.90%	45.05%	99.1%	Neural Network	38-5-1	23%
	Registers	n/a	12.92%	38.21%	99.5%	Neural Network	31-5-1	15.3%
	ALUTs	n/a	16.19%	52.73%	99.3%	Neural Network	38-3-1	17.3%
	Multipliers	n/a	81.78%	Inf%	98%	GLM (negative binomial)	n/a	n/a
	Total Wires	n/a	17.82%	44.46%	98.9%	Neural Network	43-5-1	19.35%

[a] Number of nodes in the input layer - hidden layer - output layer.

[b] This is a zero-truncated count model of the multipliers. The additional zero prediction model has an accuracy of 99.43% based on a binomial logistic regression model.

TABLE III

SUMMARIES OF COMMON HARDWARE PARAMETERS IN ALTERA/XILINX DESIGNS GENERATED BY DWARV/HAIFA'S C-TO-VERILOG COMPILER THAT QUIPU MODELS.

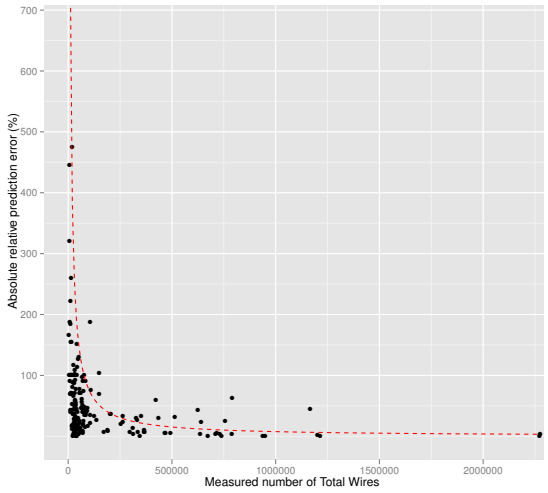

Fig. 5. The actual total number of wires vs. the relative prediction error of the total number of wires in the DWARV/Xilinx case.

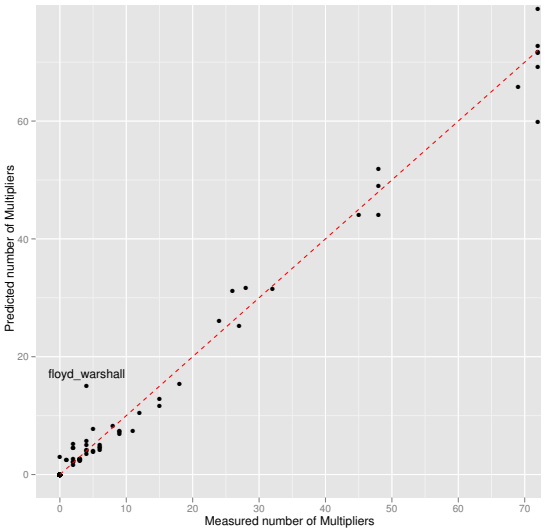

Fig. 6. The actual total number of multipliers vs. the predicted total number of multipliers in the DWARV/Xilinx case.

It is also worth to note, that the predictions on the number of clock wires are substantially better than the predictions on the total number of wires. This may be explained by assuming a more regular layout of clock distribution nets over an FPGA.

In Fig. 2 - Fig. 5, we provide plots of the predicted data and the percentage error for two neural network models. Fig. 2 and Fig. 3 show the results for the Flip-Flops model and Fig. 4 and Fig. 5 show the results for the Total Wires model. These plots make use of cross-validated predictions as data points. We chose these figures as representative illustrations of the behavior of all the factors we estimate. Observe in Fig. 2 that the predictions visually correlate quite well with the observed values. The line $y = x$ represents the ideal case. *There are no noteworthy outliers.* In Fig. 4, we see a similar behavior. Here, *we also find a good visual correlation and no noteworthy outliers.* We can see in this plot that there are a number of zero estimates. Indeed, in 11 cases, the model predicted a negative value which gets truncated to zero.

In Fig. 3, we see the percentage error on each data point. Interestingly, we see that the percentage error is not normally distributed. When we consider that the used regression techniques aim to minimize the absolute error as an amount, this error will be normally distributed when the number of data points is large. Therefore, it makes sense for the percentage errors to have an inverse relation with the measured values. We plotted the theoretical standard error σ as a percentage of the measured values in the plot to compare. Similar behavior can be seen in Fig. 5 for the number of total wires.

Fig. 6 and Fig. 7 show the prediction quality and percentage error for the composed model for multipliers, i.e. both the decision and count model. We observe similar behavior as with the neural network models. The predictions are close to the ideal line. Note in Fig. 6 that the measured values are capped at 72 multipliers. The predictions show a larger error at this point. This can be explained by the fact that some kernels would use more multipliers if available, while synthesis assigned

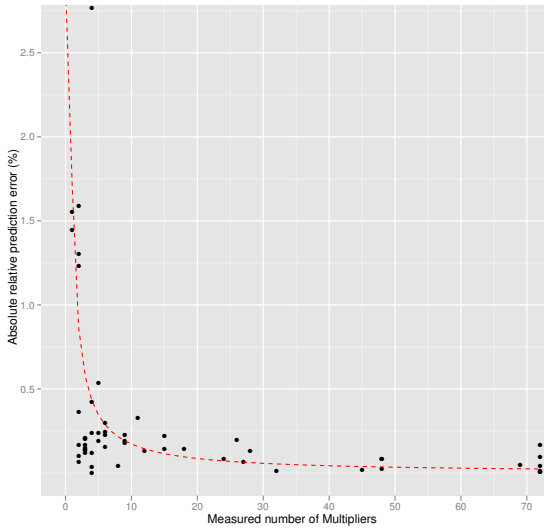

Fig. 7. The actual total number of multipliers vs. the relative prediction error of the total number of multipliers in the DWARV/Xilinx case.

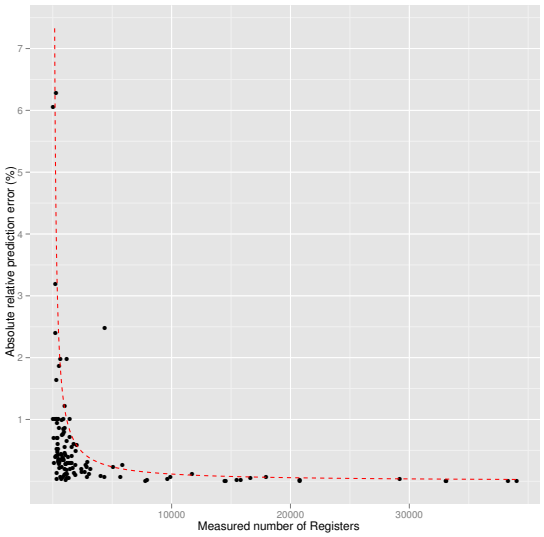

Fig. 9. The actual number of registers vs. the relative prediction error of the total number of registers in the C-to-Verilog/Altera case.

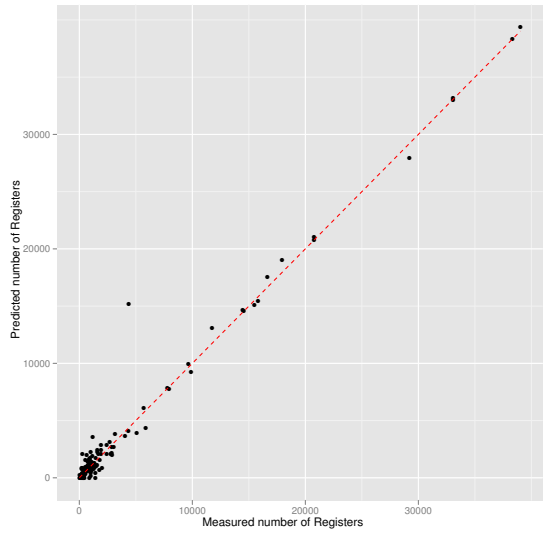

Fig. 8. The actual number of registers vs. the predicted number of registers in the C-to-Verilog/Altera case.

at most 72 multipliers. Furthermore, it's worth to observe, that one kernel is erroneously predicted to have a number of multipliers, while this is not the case. Additionaly, the plot shows one outlier, identified as the *floyd_warshall* kernel, which had a code feature not supported by DWARV. Therefore, the produced VHDL file was not correct. The synthesis tool then removed 3 32-bit multiplications during optimization, each accounting for 4 multiplier blocks (8x18 multiplier). This surmounts to a difference of 12 multipliers in the observed value for this kernel. This discrepancy explains the large offset of this prediction.

B. Results for the C-to-Verilog/Altera case

In this paper, we have validated our claim that the Quipu modeling approach can easily recalibrate models for different combinations of tools and platforms. In Table III, we observe that the reported errors for the number of ALMs, Registers, ALUTs, and wires, all are well below the prediction errors for the models we have generated for the DWARV/Xilinx case. First, note, that this shows that our approach can be recalibrated for another set of tools. Secondly, we observe that the results produced by the combination of C-to-Verilog, the Quartus II synthesis, and the Stratix IV FPGA exhibits a more linear transformation from C. It is not likely that the FPGA structure itself is debit to this fact, as the Stratix IV FPGA also is a fairly complex device. It is more likely that the C-to-Verilog and Quartus II tools produce more regular designs when using no particular optimization flags. It is our suspicion that the C-to-Verilog tool, in particular, does not use complex optimizations. This needs to be further investigated using its source code.

Furthermore, we can also observe similar behavior from Fig. 8 and Fig. 9. In Fig. 8, we observe that the predicted values correspond quite good with the actual values. We do note one outlier on the top-left of the figure. This data point represents the *idct* kernel in the kernel library. This kernel is characterized by having many constant multiplications. Apparently, the C-to-Verilog or Altera tools do not perform the strength reduction optimization in all cases, which makes Quipu underestimate the number of multipliers and overestimate the number of shifts and additions in the design. This outlier clearly shows the need to investigate the different optimizations that are performed in each modeled tool. This misrepresentation of the number of multiplications is also the main reason for the problematic prediction model for multipliers in the Altera case. As can be seen from Table III, this prediction model exhibits an error of 82%. We expect that the accuracy will improve when we have accounted for the correct optimizations employed in the C-to-Verilog tool.

C. Summary of Results

The presented approach provides relevant results that are generally applicable due to our large library of kernels and, as such, it is not characterized by cherry-picking certain kernels. However, we can see from Table I, that we have larger errors than other techniques. For the number of slices, we report an error of 26% compared to 7-10% for techniques that don't take into consideration the entire design, 5% for a method that uses a customized C dialect with regular structure, and 12% compared to a scheme that requires DFG representations. Nonetheless, *the increased error compared with other techniques is expected, as we estimate entire designs at a particularly early level from normal C code*. Moreover, our approach is general and not restricted to any particular platform or tool. In addition, it is very fast, as it takes 8.8 sec to estimate 181 kernels. Most of that time is spent in parsing the source code.

Finally, we would like to address an important issue concerning the usual method of characterizing the error in existing approaches. Almost all techniques calculate the Mean Absolute Percentage Error (MAPE) to characterize the error involved in their prediction method:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|. \quad (9)$$

As we observed in Fig. 3 and Fig. 5, individual percentage errors tend to be larger when closer to the zero. As such, MAPE will be adversely influenced by datasets with more relatively small kernels. This observation has already been made in other fields of research, e.g. [12], [20]. Other error summary statistics, such as the RMSEp or the Mean Absolute Error, can be good alternatives for measuring the error, possibly as percentages of the mean.

VII. Conclusions

In this paper, we proposed a high level quantitative prediction modeling approach with model selection, neural networks, logistic regression, and data transformations. A kernel library of 181 kernels was used to build and validate this method by using 10-Fold cross-validation. We have shown the recalibration of our approach to different tool-chains and problems. For the Xilinx case, we reported errors in the range of 15% and 34%, while for the Altera case errors ranged from 13% to 82%. More specifically, for the Xilinx case, we reported an error of 34% for the number of wires, 26% for the number of slices, and *only* 14% for the number of multipliers. For the Altera case, we reported an error of 18% for the number of wires, 17% for the number of ALMs, but 82% for the number of multipliers, due to the misrepresentation of the specific strength reduction optimization used int the C-to-Verilog tool in the measurement procedure. An important aspect that we want to address in the near future is the validation of our approach in the context of a HW-SW co-design framework. Especially, we need to evaluate whether our approach can help drive early HW-SW partitioning. Anyhow, we leave this to future research.

Acknowledgments

The authors would like to thank dr. Eric Cator for his advise on the statistical analysis of our work.

References

[1] Virtex-4 family overview, August 2010. DS112 (v3.1).

[2] H. Akaike. A new look at the statistical model identification. *IEEE Trans. Autom. Control*, 19(6):716 – 723, dec 1974.

[3] Y. Ben-Asher and N. Rotem. Synthesis for variable pipelined function units. In *Proc. SOC'08.*, pages 1–4, nov. 2008.

[4] Y. Ben-Asher and N. Rotem. Automatic memory partitioning: increasing memory parallelism via data structure partitioning. In *Proc. CODES'10*, pages 155–162, New York, NY, USA, 2010.

[5] G. E. P. Box and D. R. Cox. An analysis of transformations. *J. of the Royal Statistical Society. Series B*, 26(2):211–252, 1964.

[6] C. Brandolese, W. Fornaciari, and F. Salice. An area estimation methodology for fpga based designs at systemc-level. In *Proc. DAC'04*, pages 129–132, New York, NY, USA, 2004.

[7] R. Callan. *Essence of Neural Networks*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[8] L. M. Chuong, S.-K. Lam, and T. Srikanthan. Area-Time Estimation of Controller for Porting C-Based Functions onto FPGA. In *Proc. RSP'09*, pages 145–151, 2009.

[9] T. Degryse, H. Devos, and D. Stroobandt. FPGA Resource Estimation for Loop Controllers. In *Proc. ODES'08*, pages 9–15, Boston, MA, USA, 2008.

[10] L. Deng, K. Sobti, and C. Chakrabarti. Accurate models for estimating area and power of fpga implementations. In *Proc. ICASSP'08*, pages 1417 –1420, apr. 2008.

[11] R. Enzler, T. Jeger, D. Cottet, and G. Tröster. High-Level Area and Performance Estimation of Hardware Building Blocks on FPGAs. In *Proc. FPL'00*, pages 525–534, London, UK, 2000.

[12] W. Guang, M. Baraldo, and M. Furlanut. Calculating percentage prediction error: A user's note. *Pharmacological Research*, 32(4):241 – 248, 1995.

[13] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. IJCAI'95 - Vol. 2*, pages 1137–1143, San Francisco, CA, USA, 1995.

[14] D. Kulkarni, W. A. Najjar, R. Rinker, and F. J. Kurdahi. Compile-time area estimation for LUT-based FPGAs. *ACM Trans. Des. Autom. Electron. Syst.*, 11(1):104–122, 2006.

[15] R. Meeuws, K. Sigdel, Y. Yankova, and K. Bertels. High level quantitative interconnect estimation for early design space exploration. In *Proc. FPT'08*, pages 317 –320, dec 2008.

[16] R. J. Meeuws. A quantitative model for hardware/software partitioning. Master's thesis, Delft University of Technology, Delft, Netherlands, Delft, Netherlands, May 2007.

[17] R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis. A quantitative prediction model for hardware/software partitioning. In *Proc. FPL'07*, pages 317–320, August 2007.

[18] Á. Monostori, H. H. Frühauf, and G. Kókai. Quick estimation of resources of fpgas and asics using neural networks. In *Proc. LWA'05*, pages 210–215. DFKI, 2005.

[19] P. Schumacher and P. Jha. Fast and accurate resource estimation of rtl-based designs targeting fpgas. In *Proc. FPL'08*, pages 59 –64, sep. 2008.

[20] L. B. Sheiner and S. L. Beal. Some suggestions for measuring predictive performance. *J. of Pharmacokinetics and Pharmacodynamics*, 9:503–512, 1981. 10.1007/BF01060893.

[21] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

[22] C-to-verilog. http://www.c-to-verilog.com/.

[23] Delft workbench. http://ce.et.tudelft.nl/DWB/.

[24] S. Wenande and R. Chidester. Xilinx takes power analysis to new levels with XPower. *XCell Journal*, 41(3):26–27, 2001.

[25] Y. Yankova, G. Kuzmanov, K. Bertels, G. Gaydadjiev, Y. Lu, and S. Vassiliadis. Dwarv: Delftworkbench automated reconfigurable vhdl generator. In *Proc. FPL'07*, pages 697 –701, 27-29 2007.