Quick Estimation of Resources of FPGAs and ASICs Using Neural Networks

Ádám Monostori and **Hans Holm Frühauf** and **Gabriella Kókai**
Fraunhofer Institute for Integrated Circuits, Erlangen, Germany
Department of Computer Science 2 University of Erlangen-Nuremberg
{mno,fhf}@iis.fraunhofer.de, kokai@informatik.uni-erlangen.de

Abstract

The redFIR2 project at the Fraunhofer Institute for Integrated Circuits is a tool that provides optimised Finite Impulse Response structures. The generation process of these structures is based on a component library containing seven scalable basismodules. Depending on the chosen Integrated Circuit technology and on the I/O wordlengths the resource utilisation of the modules differ considerably. A fast, a priori estimation of resources during the system-level design is of crucial importance for the generation of resource optimised (adjusted to an Integrated Circuit technology) Intellectual Property cores. The objective of this work is to develop a flexible, adaptive resource estimation methodology.

1 Introduction

For automated design of integrated circuits it is essential to estimate the resource usage of Intellectual Property (IP) cores during the system-level design. In the redFIR2 project at the Fraunhofer Institute for Integrated Circuits the automated generation of Finite Impulse Response (FIR) structure IP cores is based on a library containing scalable basismodules. Depending on the chosen Integrated Circuit (IC) technology (diverse Field-Programmable Gate Array and Application Specific Integrated Circuit technologies) and on the I/O word lengths the resource usage of the modules differ considerably.

A fast estimation of the necessary IC resources is, however, an important basic principle in order to be able to generate optimal (adjusted to a chosen IC technology) IP cores. In the scientific field several methods exist to predict the utilisation of individual IC resources, like routing, area and power consumption. Nevertheless, their estimation methods are based on dedicated algorithms for certain technologies (that is, arbitrarily chosen by the developer of the method, for example Xilinx XC4000E in [Enzler *et al.*, 2000]), and they could only be adapted to different technologies with great effort.

This paper presents our approach, a flexible, adaptive estimation of resource usage. This is a nonlinear optimisation problem as our aim is to determine the (local) extreme of nonlinear functions of many unknowns (number, type and word-length of modules). Our optimisation process is based on neural networks trained by resource measures yielded from automated design flows.

2 Background

This section introduces the basics in order to understand the aim of the paper.

2.1 The redFIR2 system

The **redFIR2 system** (see Figure 1) is a web-based service for generation of optimised FIR Structures. This genetic algorithm-based system generates optimised FIR structures with respect to the user-defined FIR structure- and configuration data. These structures are then translated into a hardware description language as the input for the synthesis and implementation processes.

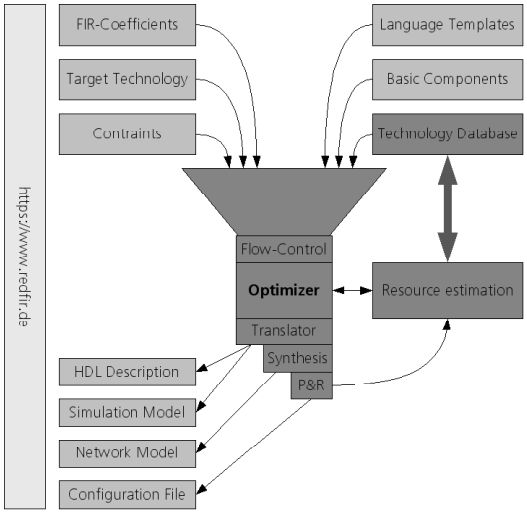

Figure 1: The redFIR2 system

2.2 Characteristics of the desired framework

As already mentioned, the generation of FIR structures is based on user-defined information: besides filter configuration the description of the target technology and optimisation guidelines (area and speed weights) are included, too. These optimisation guidelines provide a balance for the multi-objective optimisation in the redFIR2 engine, but they are not able to indicate measures of individual resources. The web interface should be extended with the possibility of giving hard constraints regarding to the available resources including the following limitations:

- **Area:** Unoccupied surface and technology-dependents resources, like the available embedded multipliers, shift registers, etc.
- **Speed:** The circuitry must be able to run on a specified frequency (if it is reachable anyway).

- **Power consumption:** the circuitry should not consume more power as indicated by the user.

In order to be able to fulfil these constraints the resource usage of FIR structures must be predicted during the generation process. The task of this work is to provide the redFIR2 engine with estimations of the resources mentioned above not only for FIR structures, but for components (add, shift, etc.), too.

If no estimations are made we could not know whether a specific FIR structure meets the requirements until it is generated, synthesised and implemented. After implementation the necessary information about the resource usage are at hand and they can be compared to the requirements. In the case of insufficiency, the whole filter has to be redesigned, resynthesised and reimplemented (design respin). This results in longer computation times and superfluous use of Electronic Design Automation (EDA) tools. These tools are rather expensive and have limited licences, which infers that we might prohibit the other users from using them. Another issue is that while an estimation could be provided in matters of seconds, the implementation of the FIR structures can take hours or even days, depending on their complexity.

To quickly summarise the requirements, this framework must provide a quick estimation of resources on the level of components and structures, too. Furthermore, it has to be able to learn new technologies without requiring great effort from the designer and it must improve itself by validating the already generated, but not measured components and FIR structures during the night-shift, when all the licences are accessible.

2.3 Integrated Circuits Design

In this section the basics of Cell-Based Application Specific Integrated Circuits (CBICs) and Field-Programmable Gate Arrays (FPGAs) are introduced. Furthermore, the resources of integrated circuits will be described highlighting the differences between CBICs and FPGAs.

Cell-Based Application Specific Integrated Circuits

A CBIC [John and Smith, 1997] uses predesigned, precharacterized and pretested logic cells known as standard cells (the generic standard cell layout is shown in Figure 2, where the I/O ports surround the logic core containing the rows of standard cells and their connections). The ASIC designer defines only the placement of the standard cells and the interconnect in a CBIC. The advantage of CBICs (compared to other ASIC approaches) is that designers save time, money, and reduce risk by using standard-cell libraries.

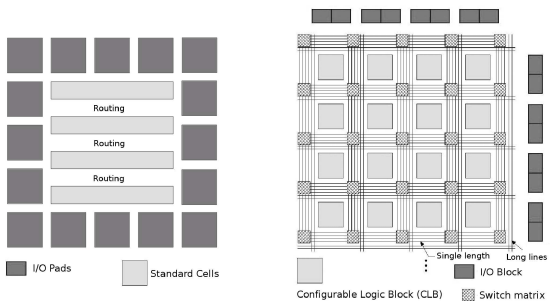

Figure 2: Generic standard cell layout of a CBIC and classical FPGA architecture

Field-Programmable Gate Arrays

A typical FPGA [Oldfield and Dorf, 1995; Wannemacher, 1998] device consists of a prefabricated array of configurable logic blocks (CLBs) surrounded by configurable routing. Each logic block consists of resources which can be configured to define discrete logic, registers, mathematical functions and even Random Access Memory (RAM). A periphery of configurable pads (I/O ports) provides connection to other electronic devices. Figure 2 illustrates the classical FPGA architecture. The function of all of these configurable resources can be defined at any time during the operation of the device to form a large logic circuit. Configurable logic and routing can be formed together to ensure the exact function of a digital processing algorithm. Parallel and pipelined data flows are possible, providing an excellent resource for execution of a signal processing algorithm.

Resources of Integrated Circuits

In general resources are the collectivity of the available facilities solving a certain problem. In case of integrated circuits we focused on the following resources of interest:

1. The term **area** reflects the size of the circuitry but it differs greatly on the chosen technology. The core active area, the periphery active area and interconnect area determine the required chip size (measured in mm^2) for a standard-cell design. In the case of FPGAs area is measured in terms of number of logic blocks and embedded components (like multipliers, memories, shift registers, etc.). Routing, powering and the clock network are excluded, because they are pre-fabricated on the FPGA board.
2. **Propagation delay** is the time required for a digital signal to travel from the input(s) of a component to its output. Propagation delay is important because it has a direct effect on the speed at which a digital device can operate. The frequency f measured in Hertz (which means cycles per second) of an oscillator used to time or synchronise the operations of a circuitry. The higher the clock frequency, the faster the operation of the circuit. The period of the clock (T_{period}) is the time taken to complete one cycle and is the inverse of the frequency f: $T_{period} = \frac{1}{f}$. The **maximum clock frequency** of the circuitry depends on the components' propagation delay and on their routing delay: the slowest component and the wiring delays limit the maximum frequency.
3. **Power consumption** is in general the energy over time ($P = \frac{E}{t}$) that is supplied to a system to maintain its operation. Power consumption can be computed by $P = \frac{1}{2}CU^2 f_{eff}$, that means, that power is proportional to the capacitance (C), operating voltage (U) and effective frequency (f_{eff}). Reducing any one of these variables reduces power dissipation. In the case of integrated circuits there are two main components of power consumption: dynamic and static consumption:
   - Static power is the minimum power required to keep the device 'powered-up' with the clock inputs not switching and the I/Os drawing minimal power.
   - Dynamic power is the power consumed when both the I/Os and logic cells are switching. Dynamic power is a function of the switching frequency and operating temperature.

2.4 Digital Filters

A main field of application of Digital Signal Processing [Porat, 1997] (DSP) is digital filtering. The enormous parallel computation performance of digital filters made DSP so popular. Filtering is commonly used in two basic cases: *signal separation* and *restoration*. Separation is needed if either the signal must be separated from other received data or it is presented together with some signals which are insignificant in the given task. Signal restoration can be a demand if the signal is damaged or distorted.

A digital filter transforms a discrete sequence of numbers (the input) into another discrete sequence of numbers (the output) having a modified frequency domain spectrum.

In the field of DSP two alternatives are know: non-recursive (Finite Impulse Response) and recursive (Infinite Impulse Response) filters [Porat, 1997]. The extensions of FIR filters are the FIR structures which are widely used for the design of digital filters as integrated circuits, especially in case of DSP systems dealing with enormous data rates and a high bandwidth in order to realize series expansion in combination with hysteresis functions.

3 Related work

There already exist several approaches in the field of resource estimation. In [Enzler *et al.*, 2000] a high-level estimation methodology is proposed for area and performance estimation of FIR filters. This methodology is based on very subtle constants acquired by thorough analysis of the target technology as the investigation concerned only one platform-FPGA. For all resources equations have been established using observed constants. The pitfall of this approach is its inflexibility:

- if the basic modules change, then some (or all) constants must be re-investigated, which needs a great effort from the engineers.
- The same holds for the programs used to implement the filter on the FPGA: substituting some of these programs may result in significant changes.

In [Bilavarn *et al.*, 2000] a general performance estimation technique is presented. Here the behavioural description is given in the C language which is then translated into a graph. For each node the functional units are determined and their resource usage is retrieved from a library, which has the characterisation of these units. These resource usages are combined afterwards, in case of area they are simply summed up without correction. Timing estimation is worse: the effect of interconnection delays are not taken into account. This approach is not applicable today, in the deep submicron era, as interconnect has an enormous influence on timing.

These two approaches are not applicable for our problem, as they are neither dynamic nor reasonably accurate. The system must react to the changes of the components without requiring great effort from engineers and it must be capable of learning new technologies by itself.

4 Learning of resource utilisation of Finite Impulse Response Structures

This section clarifies the communication between the redFIR2 engine and the estimator subengine, how the models of resource estimations are provided and the schematic design of the system.

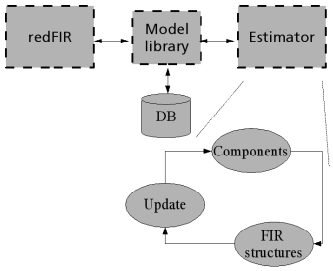

Figure 3: System overview: estimation of components and structures

In order to make the redFIR2 engine capable of generating optimal, resource-limited FIR structure IP-cores an estimator subengine must be developed to support it with fast and accurate models of resource utilisation of components and FIR structures, too (see Figure 3). The subengine maintains a database storing resource measurements and, based on these measurements it creates models of resource utilisation on the component and FIR structure level.

As mentioned, the models must be **fast** and **accurate**. These requirements are opposite to each other, an improvement in one of them generally yields the deterioration of the other. Therefore a balance between them must be found.

Another desired feature of the estimator subengine is cost-efficient **self-improvement**, therefore it has to work in two modes:

- during the normal work-hours it provides the redFIR2 engine with estimations for both components and FIR structures.
- after work-hours it improves its estimation by automatically measuring arbitrary components and FIR structures. Similarly, during this period the changes in the component libraries and the introduction of additional technologies are learnt and adapted. Our system, being an adaptive framework, does not require great effort from its designers: they only have to define the VHDL description of the additional, technology-specific component libraries and implement the design flows for the new technologies.

Without models of the components' resource usage no (reasonably accurate) FIR structure resource prediction can be made. For that reason the estimation process is split up into two separate parts: component and FIR structure estimation (shown in Figure 3).

The basic idea is to characterise the components first, then based on these component models, characterise the FIR structures afterwards reflecting a bottom-up model of estimation.

4.1 Component estimation

The components are the basic building blocks of FIR structures, hence their resource usage characteristics give a guideline for the FIR structures' prediction. The resource utilisation of components is influenced by many circumstances:

- It depends on the chosen technology (FPGA vs. ASIC). Furthermore, the capability of the used EDA tools also affects it.
- Different types of components have different resource utilisation characteristics (for example, a simple delay element utilises less resources like a complex multiplier).

- The different I/O word-lengths and component-specific parameters also make a considerable difference.
- Within a component, many different implementations can be realized leading to discontinuities in the resource utilisation characteristic. For example, one could use a simple adder implementation up to a certain I/O word-length and then a more sophisticated one. These implementations utilise the resources differently, therefore there will be a break in the resource usage characteristic of the component.

The problem is function approximation with finite number of discontinuities. After investigating several Machine Learning algorithms we decided to use neural networks. The structure of the neural networks is built up using:

- two hidden layers containing sigmoid units with the assumption that the extra hidden layer should be capable of extracting the features (behaviour) of the component's resource usage characteristics which are not contained implicitly in the input representation [Cybenko., 1988; Mitchell, 1997]
- an output layer built up by linear units

Given the number of layers, the number of neurons in each layer must be decided:

- The number of neurons in the input layer is component-specific (for example, the input vector of an adder contains the word-length of the two operands and the result) and therefore determined a priori.
- The number of neurons in the output layer is given by the vector-length of the target function: we know in advance, how many resources we want to estimate.
- From the estimators point of view there is only a set of components and the estimator does only know their names and parameters, but not their functionality. Therefore the number of neurons in the hidden layers vary for different components, some components are more difficult to characterise than others, hence they need more neurons in the hidden layers for accurate estimations. Therefore the structure of the networks must be modified dynamically during the training periods: similarly to Cascade-Correlation [Fahlman and Lebiere, 1990], if the starting network's performance does not reach an acceptable level (which is 10^{-3} for area, 10^{-4} for delay and 10^{-3} for power) we keep adding neurons (based on an increment value) to the second hidden layer until the network converges. On the other hand, if the network has converged we try to minimise the number of neurons (to avoid overfitting) in the second hidden layer by removing some ($\frac{increment}{2}$) of them. The training process is then repeated until convergence is obtained.

Instead of Backpropagation, the chosen training method is Resilient Propagation providing faster convergence caused by the direct adaptation of the weight steps.

Another issue is to predict the different resources (area, propagation delay and power) on the component-basis separately, thus for each component we have as many neural networks as the number of resource-classes. The reason for that separation is that different resources require different accuracy in their prediction, which means there are distinct neural networks for area, power and speed estimation for each component with customised performance criterion.

Another practical issue that the training vectors must be also analysed and modified on demand. An obvious example is that an adder never utilises an embedded multiplier or a shift register, therefore these means of area should not be estimated by the neural network and therefore these attributes must be removed from the corresponding training examples. In some cases the results of a neural network have to be post-processed also. In case of area prediction of FPGAs the results must be rounded, as no component can utilise the half of a register, for example.

Component measurement

In order to provide our learning methods with sufficient training data, training structures must be generated both for components and FIR structures.

The generation of component structures is easy as their VHDL [Rajan, 1999] sources are available in the component library. For each structure a VHDL description is generated by instantiating the corresponding component with arbitrary parameters: I/O word-lengths and other, component-specific properties like the number of clock cycles to delay (delay elements) and the shift value (shifters). That means we only have to know the necessary parameters of each component, but nothing about their functionality and how they are implemented.

Having the VHDL description, the training structure must be implemented by technology- and vendor-specific EDA tools. As the result of this implementation flow the reports of each stages are generated containing the necessary information about the resource utilisation of the test structure. These reports must be parsed to acquire this information followed by storing them in the components' database to ensure their availability at training: the parameters of the component serve as the inputs and their measured resource usage constitute the outputs of the neural networks which are supposed to learn the dependency between them.

4.2 System estimation

The problem is solved by using a bottom-up approach based on our background knowledge of physics and FIR structures: we investigate the effect of the individual components resource usage for predicting the resource usage of a structure. The strategy is to "learn" a decision tree (shown in Figure 4). Actually, the structure of the tree is not learnt (and therefore the "decision tree" denotation is a bit strong), but it is based on our intuition:

- at the root we split our tree based on the technology (being FPGA or ASIC),
- afterwards, we descend into the tree, specifying the target platform more precisely at every step and
- in the end, in one of the leaves the corresponding stored neural networks of components and correction terms together with the resource equations are found. Thus the resource usage of a FIR structure can be approximated combining the components' resource utilisation.

To explain our assumptions, the effects between the components' and the FIR structures' resource usage are clarified now for the resource-classes (area, delay and power).
Area of a structure is computed by

$$Area = \sum_{c \in C} A_c * C_a + C_{ao} \quad (1)$$

where *C* is the set of components contained in the current FIR structure. The area terms of the individual components

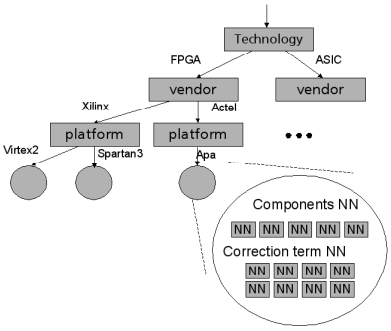

Figure 4: Decision tree of FIR structure estimation

(A_c) are summed up and corrected by C_a representing the effects of the optimisations during synthesis and placing. The correction term C_{a0} represents the area of I/O pads.

Power consumption is given by

$$Power = \sum_{c \in C} P_{dc} * C_p + C_{po} \quad (2)$$

Power consumption has two components, namely, static and dynamic consumption. The power consumption is computed by summing the dynamic power of each component (P_{dc}). The correction factor C_p smooths out the effect of optimisation. C_{p0} is the static power term both for FPGAs and ASICs.

Delay of a structure is determined by

$$Delay = \max_{c \in C}(D_c) * C_d \quad (3)$$

This reflects the fact that the slowest component in the parallel pipeline limits the speed of the structure (maximal clock frequency). The correction factor C_d is the speed-degradation term caused by the delays of the routing wires.

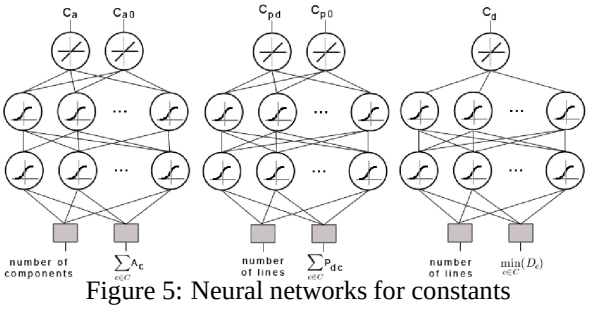

Figure 5: Neural networks for constants

It can be clearly seen that for the estimation of FIR structures we need to determine these correction terms for each technology. The problem is that these correction terms are not really constants but functions. For example, in the related work [Enzler *et al.*, 2000] the areas of the individual components has been summed up and in the evaluation results it can be seen that they observed overestimation effects for little FIR filters and underestimation effect for bigger filters. Therefore for each correction term its value must be approximated using dedicated neural networks (shown in Figure 5). These networks are simpler than the ones for the components, they have also two hidden layer but they have static structure, no dynamic modification of the number of neurons is needed. In order to train these networks, FIR structures have to be implemented and measured. Furthermore, the estimation of components are also needed. Our correction networks need the following structure of training instances:

- Area: The input vector contains the summed up component area estimations and the number of components in the FIR structure. In the output vector the areas of I/O pads (C_{a0}) and the proportions of the measurement of the structures' area and the summed up component estimations. ($\frac{area_{structure}}{\sum_{c \in C} A_c}$).
- Delay: The input vector contains the numbers of lines in the structures and the delay of the slowest components, in the output vector there are the proportions of the structures measured delay and the delays of the slowest component ($\frac{delay_{structure}}{\max_{c \in C}(D_c)}$).
- Power consumption: The input vector contains the summed up component dynamic power consumption estimations and the number of components in the FIR structure. In the output vector there are the proportions of the measurement of the structures' dynamic power consumption and the summed up component power estimations. ($\frac{Power\_dynamic_{structure}}{\sum_{c \in C} P_{dc}}$). For FPGAs the estimation of static power consumption is not needed, because it is constant, but for ASICs it must be also estimated.

Having the neural networks of component estimations, correction term estimations and resource equations (Equations 1 - 3) the estimations of structures can be provided. The estimation flow (shown in Figure 6 for area) starts by parsing the VHDL description in order to retrieve the components of the structure. Based on the technology description the corresponding component and correction term neural networks are retrieved from the decision tree. The component NNs are then activated and their estimations are accumulated defining the first part of the equation. After that, by activating the correction term NNs the whole equation is completed and the structure estimation is provided.

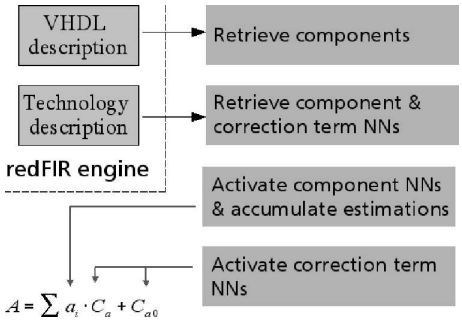

Figure 6: Flow of FIR structure estimation

System measurement

Having the set of components of a given technology, the FIR structures can be created by valid combinations of these building blocks. Actually, these generated structures are not optimised and we are not interested in their functionality. However, creating such "dummy" structures help us to extract the components' impact on the structures resource utilisation: we expect that this impact is the same for "real" FIR structures.

The generation of the training structures must be driven by parameters limiting

- the depth (number of levels, limiting the size of the structure) and
- the width (number of components on a level, reflecting the degree of parallelism)

- word-length of the structure's input signal: we limit either the input or the output word-length, but not both of them

Given a relatively few parameters, the structure is generated and translated into a VHDL description, which is then implemented on the specified technology. As the result of the implementation, the generated reports are parsed and the necessary information about the resource utilisation, together with the VHDL description are stored in a database. These structure measurements are then used to create the training instances for the neural networks of correction constant estimation.

5 Test results

During the tests we used the Xilinx VirtexII [Xil, 2004] platform FPGA as validation technology. For the validation of the correctness of component estimations around 100-120 instances have been implemented for each component. Furthermore, the input word-lengths were limited up to 90 bits, therefore instances having wider inputs might not be estimated accurately. Ninety percent of the instances represent the training set (the set of instances used to train the neural networks) and the remaining ten percent constitute the test set. During the validation, the neural networks are put into operation having the test set as their input. The validation results of components are shown in Table 1: as it can be seen, on the component-level the neural networks provide very good estimations for area, maximum clock frequency and dynamic power consumption (even though that for power the error is significantly larger). For the instances that were constituting the training set of the networks the error is negligible, the estimations and measurements are nearly the same.

	accuracy (mean relative error) [%]	
type of resource	components	structures
Area	3.5	18.99
Maximum Frequency [MHz]	5.5	22.22
Dynamic power consumption [mW]	17	72.1

Table 1: Validation results of the components and FIR structure estimation

For the validation of the FIR structure estimations 250 structure instances have been implemented. The depth has been limited to 7 levels and the maximal width of a level was restricted to maximum 8 components, the input word-lengths were limited up to 20 bits. From the structures ninety percent were used for training and ten percent for the validation. The validation results of FIR structures are shown in Table 1: the estimations of area and maximum clock frequency are quite satisfactory, but for power consumption we encountered a high relative error.

6 Conclusion & Future Work

In this paper we presented our novel approach of IC resource estimation based on Neural Networks. Our aim was to provide estimations with a mean relative error better than 25 percents which is the half of the error that the current redFIR2 engine uses. We developed a bottom-up strategy by reasoning from the resource usage of components to the structures' resource utilisation.

In this work we concerned only the Xilinx VirtexII FPGA technology so far, therefore one of the further development tasks of the system should involve the investigation of other technologies. Another task is to improve the estimation process (at least for power consumption) either by restructuring both the training instances and the neural networks or by investigating and adopting another promising ML methods (like Support Vector Regression and Symbolic Regression). Regarding to self-improvement, the construction of training sets should be done deliberately depending on the performance of the corresponding neural networks. Our approach was validated on FIR structures, but in our opinion it supposed to be a promising way to estimate other, component-based digital structures.

References

[Bilavarn *et al.*, 2000] Sébastien Bilavarn, Guy Gogniat, and Jean Luc Philippe. Area time power estimation for fpga based designs at a behavioral level. In *The 7th IEEE International Conference on Electronics, Circuits and Systems*, pages 524–528. IEEE, 2000.

[Cybenko., 1988] George Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, MA, 1988.

[Enzler *et al.*, 2000] Rolf Enzler, Tobias Jeger, Didier Cottet, and Gerhard Troster. High-level area and performance estimation of hardware building blocks on FPGAs. In *Field-Programmable Logic and Applications*, volume 1896 of *Lecture Notes in Computer Science*, pages 525 – 544, 2000.

[Fahlman and Lebiere, 1990] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. pages 524–532, 1990.

[John and Smith, 1997] Michael John and Sebastian Smith. *Application-Specific Integrated Circuits*. VLSI Design Series. Addison-Wesley Publishing Company, 1997.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill International Editions, New York, 1997.

[Oldfield and Dorf, 1995] John V. Oldfield and Richard C. Dorf. *Field-programmable gate arrays: reconfigurable logic for rapid prototyping and implementation of digital systems*. Wiley-Interscience publication, 1995.

[Porat, 1997] Boaz Porat. *A course in digital signal processing*. John Wiley & Sons, Inc., 1997.

[Rajan, 1999] Sundar Rajan. *Essential VHDL: RTL Synthesis Done Right*. S & G Publishing, 1999.

[Wannemacher, 1998] Markus Wannemacher. *Das FPGA-Kochbuch*. International Thomson Publishing GmbH, Bonn, 1998.

[Xil, 2004] Xilinx, Inc. *VirtexII Platform FPGA User Guide*, April 2004.