ON DESIGNING SELF-AWARE RECONFIGURABLE PLATFORMS

K. Siozios[1], H. Sidiropoulos[1], D. Diamantopoulos[1], P. Figuli[2], D. Soudris[1], M. Hübner[3] and J. Becker[2]

[1] School of Electrical and Computer Engineering, National Technical University of Athens, Greece
[2] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[3] Ruhr-University of Bochum, Germany

ABSTRACT

Existing application domains exhibit variations in terms of complexity, performance and power consumption, whereas their efficient implementation onto general-purpose FPGAs is not always a viable solution. In this paper we introduce a framework for designing self-aware reconfigurable platforms. Rather than similar approaches, our solution having a template of architectures, instantiates the most suitable FPGA platform at run-time, depending onto the application's requirements. Experimental results with various applications prove the effectiveness of such an approach, as compared to perform application mapping onto a conventional FPGA architecture.

1. INTRODUCTION

Reconfigurable platforms and more specifically Field-Programmable Gate Arrays (FPGAs) have become the implementation medium for the vast majority of modern digital circuits. This make the FPGA paradigm to grow in importance, as there is a continuous demand towards faster, smaller, cheaper and lower-energy devices.

Apart from the flexibility introduced by reconfigurable architectures, usually such devices exhibit limitations posed by application's inherent properties. For this purpose, FPGA vendors provide a variety of devices for each family, each of which has different properties (e.g. LUT size, number of LUTs per slice, amount of routing resources, etc), whereas the selection of proper FPGA device for a specific application becomes a challenging issue. Additionally, it is common that different applications, even from the same domain, might lead to considerable performance variations whenever they are mapped onto a single device.

In order to alleviate the problem of selecting the most suitable FPGA architecture for a given benchmark, throughout this paper we introduce a novel framework for designing self-aware reconfigurable platforms. The architectural properties of these platforms are tunable and customizable at run-time, depending on the constraints posed by the target application.

Different approaches have been used for determining the most suitable parameters for target FPGA. Among others brute force and simulated annealing are two candidate approaches, which are mostly applied at design time, due to their increased computational complexity. However, throughout this paper, we incorporate an approach with significant lower complexity, which is based on neural network.

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. Among others, neural networks provide non-linearity, are universal approximatiors (can approximate input–output functions to any desired degree of accuracy, given an adequate computational complexity) and they are adaptable (adjustable synaptic weights and network topology, can adapt to its operating environment and track statistical variations).

In contrast to relevant approaches aiming to perform architecture-level exploration with the usage of neural networks [1] [2], our framework instantiates also the derived (optimal) Virtual FPGA and then performs application implementation onto this platform under the selected design criteria. In order to support these tasks, apart from the introduced neural network, we incorporate also our former Virtual FPGA architectural [3], as well as the CAD flow for application mapping onto the target architecture [4] [8] [9].

The paper is structured in the following manner: Section 2 describes the target architecture, whereas the proposed methodology for supporting the self-aware reconfigurable platforms is discussed in Section 3. Qualitative and quantitative results that prove the effectiveness of introduced solution can be found in Section 4. Finally, conclusions are summarized in Section 5.

2. TARGET ARCHITECTURE PLATFORM

Fig. 1 gives an abstract view of our target platform consisted of multiple heterogeneous instantiations of Virtual FPGAs (V-FPGA), each of which is based on Xilinx Virtex devices. The term heterogeneous refer to the

possibility each of the employed Virtual FPGAs to have different properties. Typical parameters for differentiation are the size are the LUT size, number of LUTs per slice, number of routing channels, etc. By appropriately handling these parameters, it is possible to design either a high-performance, or low-power, FPGA platform.

Apart from the Virtual FPGA the target platform includes also a microprocessor core, a configuration controller, some of-chip memories and busses for on-chip communication and configuration. While the microprocessor is well suited for control oriented tasks and interfacing, the virtual FPGA cores add the advantage of efficient parallel data processing to the system. On-chip communication is realized by AMBA busses. The microprocessor can access the virtual FPGA cores by the AMBA APB bus, since each core acts as an APB slave. The microprocessor also communicates with the configuration controller by the AMBA APB bus. It specifies which configuration the configuration controller should load into a certain virtual FPGA core. All configurations are stored inside an external non-volatile memory. The configuration controller can access the memory controller by the AMBA AHB bus. This requires a bus arbiter as there are two masters on the AMBA AHB, the microprocessor and the configuration controller. Additional details about the architecture of Virtual FPGA can be found in [3].

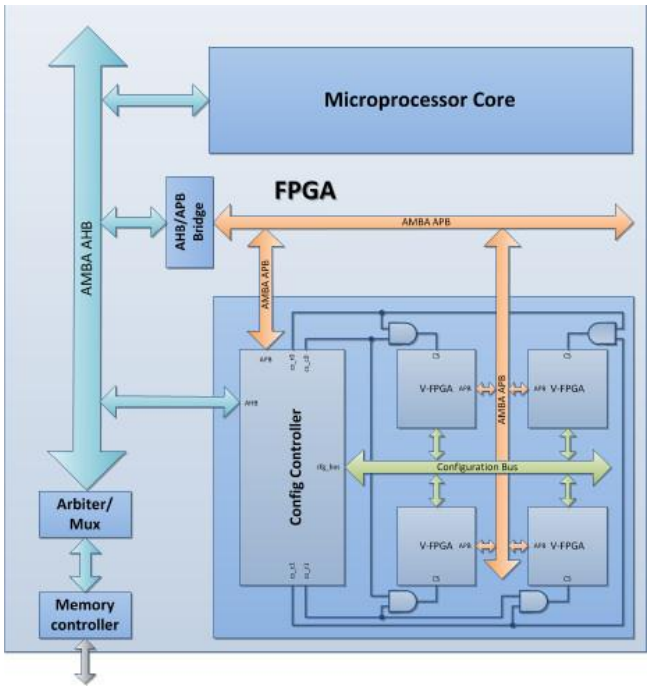

Fig. 1: Schematic view of the system architecture [3].

The platform described in Fig. 1 was developed at generic VHDL and it is mapped onto logic cells within a physical FPGA board. Next section describes in detail the proposed methodology for supporting both the customization of these Virtual FPGAs at run-time based on the application's requirements, as well as the application mapping onto the derived platforms with the usage of a Just-in-Time compilation framework.

3. PROPOSED SOLUTION

The proposed methodology for performing application mapping onto self-aware reconfigurable platforms is depicted in Fig. 2. As input to this framework we use the synthesized application's description.

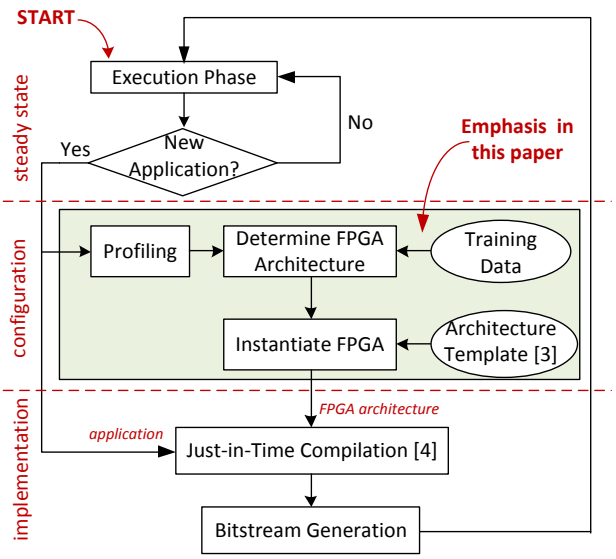

Fig. 2: Proposed methodology for supporting self-aware reconfigurable platforms

Whenever a new application has to be mapped onto the reconfigurable platform, the proper Virtual FPGA is instantiated. In order to derive the architecture's parameters of this device, we profile the application's netlist to extract a number of application-oriented parameters. For the scopes of this paper, the following parameters per benchmark are profiled: (*i*) number of nodes, (*ii*) number of edges, (*iii*) number of primary inputs/outputs, (*iv*) average and maximum fanout.

The profiling results provide a first estimate about the desired architecture for the target FPGA device. Different approaches might be used in order to appropriately fuse these results (e.g. brute force, simulated annealing, genetic algorithms, etc). However, all of them impose mentionable run-time overhead, which is not affordable

especially for systems that have to be executed at run-time.

For this purpose, our methodology incorporates a fast yet accurate multi-objective optimization technique based on neural network. A neural network is a system that learns to map a function from an input vector to an output vector. It consists on a set of simple units which are called artificial neurons. Each neuron has an internal state which depends on its own input vector. From this state the neuron maps an output that is sent to other units through parallel connections. Each connection has a synaptic weight that multiplies the signal travelling through it. So, the final output of the network is a function of the inputs and the synaptic weights of the neural network.

Such a neural network with enough elements, called neurons, can fit any data with arbitrary accuracy. The outcome from neural network is the most suitable architectural parameters in terms of LUT size, number of LUTs per slice, number of routing channels, etc. These parameters are handled by our framework in order to automatically instantiate the desired Virtual FPGA platform. The key differentiation of this solution, as compared to the rest approaches discussed previously, affects the significant lower run-time overhead for deriving architectural parameters of target Virtual FPGA.

A critical task for deriving this optimized architecture affects the proper training of neural network with a representative set of benchmarks. For this purpose, at design-time, we trained the employed neural network with a variety of applications from different benchmark suites (MCNC, LGSynth93, QUIP). Hence, we can almost safely guarantee that the architectural properties derived from our solution are close enough to the optimum device FPGA.

Then, we perform technology mapping, placement and routing (P&R) with the usage of proposed JIT compilation framework. Since a number of additional tasks might be already mapped onto the reconfiguration device, our JIT framework preserves that application's functionalities are mapped onto available (empty) hardware resources. Having as input our framework this information, JIT can perform task implementation (P&R) only with non-utilized resources.

We have to mention that our Virtual FPGA enables such a fine-grain reconfiguration (e.g. routing wires and/or logic blocks inside a single slice). Additionally, it supports a read-back technique for retrieving the current state of configuration data.

The outcome from JIT framework contains all the necessary information in order to compute (*i*) the partial bitstream file for the new task and (*ii*) the resources over the target architecture where this task has to be allocated. Finally, the computed bitstream file configures the Virtual FPGA with the new task. Additional details about how we apply this technique can be found in [3] [6] [7].

The task of customizing Virtual FPGA and generating the properly partial reconfiguration data with the usage of JIT framework is software-supported by an open-source toolset, named 2-D MEANDER [8] [9]. Even though one might expect that running technology mapping and application's P&R will introduce mentionable overheads in execution time and the quality of derived results, we have appropriately tuned these tools in order to improve significantly the run-time overhead without penalties in terms of maximum operation frequency and power consumption, since it is possible to be executed sufficiently even at the embedded processor. Additionally, the JIT framework is expected to introduce the minimum possible fragmentation, since it does not require contiguous area of empty (non-utilized) hardware resources.

4. EXPERIMENTAL RESULTS

For exploration purposes, the employed neural network was modeled in Matlab, whereas after determining the optimal parameters for training (e.g. number of neurons), the network was also developed in C++. This allows integrating the developed network with our MEANDER design framework (depicted in Fig.2).

A critical parameter that affects the efficiency of designed neural network is the regression. Fig. 3 summarizes the metrics of this parameter, as we vary the number of hidden neurons and epochs (determine the maximum number of iterations for training).

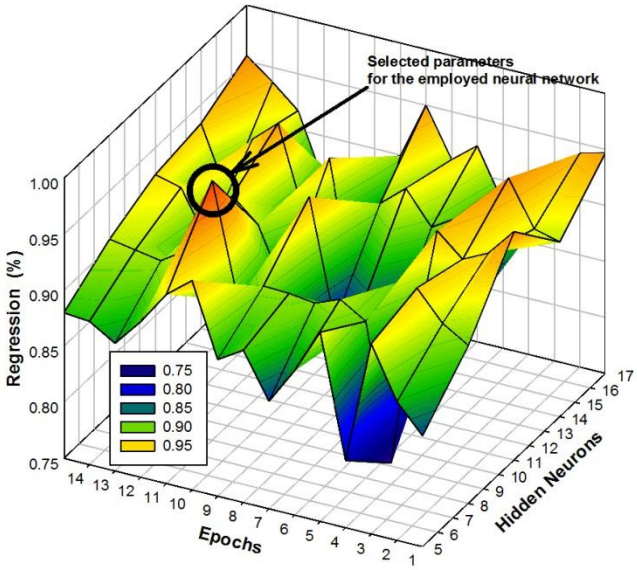

Fig. 3: Exploration results for designing neural network

Based on this graph, we can conclude that mentionable variation in this parameter is possible among the alternative solutions. This also imposes that in order to design an efficient neural network, careful study is required for determining both the number of epochs, as well as the hidden neurons. Specifically, from our exhaustive exploration depicted in Fig. 3, we found that the optimal solution retrieves whenever the number of epochs and hidden neurons are set equal to 12 and 7, respectively.

Another important parameter that quantifies the efficiency of the derived neural network affects its error. This parameter is computed by finding the error between the network's output and the target value over all the example pairs (targets - outputs).

The output of this analysis is summarized in Fig. 4. Specifically, this figure plots the error histogram for the selected neural network. The red color lines denote the optimal solution retrieved after *brute-force* exploration. Regarding our experimental setup, we performed almost 200,000 runs of VPR tool [5] for deriving the results marked with red color.

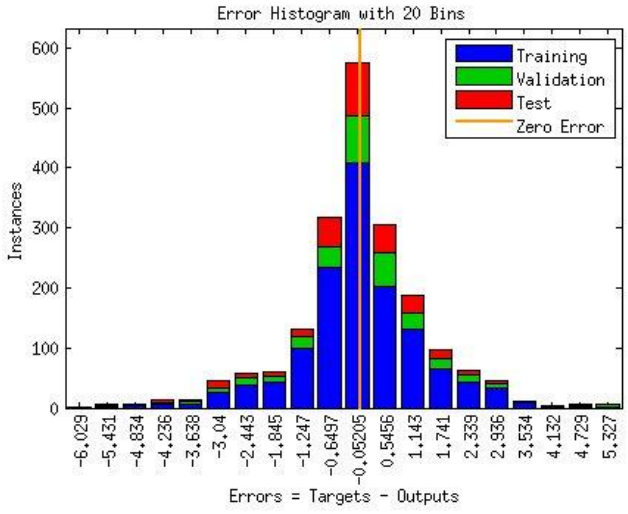

Fig. 4: Error histogram for the employed neural network.

Based on Fig. 4, we can conclude that our proposed network leads to considerable lower execution time compared to the *brute-force* approach, with an almost negligible penalty in selections of architectural parameters.

Note that due to lack of space, the results in this paper affect only the minimization of Power×Delay product. However, apart from this experimental setup, we have already retrieved the corresponding training data for delay, power, area, or any potential combination among them.

5. CONCLUSION

A framework for supporting application mapping onto self-aware reconfigurable platforms was proposed. The introduced solution based on neural networks can achieve sufficient tuning of architectural parameters under delay, power and area metrics.

6. REFERENCES

[1] A. Gomperts, A. Ukil, F. Zurfluh, "Development and Implementation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications" , IEEE Trans. on Industrial Informatics, vol. 7, no. 1, pp. 78 – 89.

[2] Z. Marrakchi, H. Mrabet, U. Farooq and H. Mehrez, "FPGA Interconnect Topologies Exploration," International Journal of Reconfigurable Computing, vol. 2009, Article ID 259837, 13 pages, 2009. doi:10.1155/2009/259837.

[3] M. Hubner, et.al., "A Heterogeneous Multicore System on Chip with Run-Time Reconfigurable Virtual FPGA Architecture", Proc. of Reconfigurable Architectures Workshop (RAW), pp. 143-149, May 2011, USA.

[4] H. Sidiropoulos, K. Siozios, P. Figuli, D. Soudris and M. Hubner, "On Supporting Efficient Partial Reconfiguration with Just-In-Time Compilation", Proc. of Reconfigurable Architectures Workshop (RAW), May 2012, China.

[5] V. Betz, J. Rose, and A. Marquardt, "Architecture and CAD for Deep-Submicron FPGAs", Kluwer Academic Publishers, February 1999.

[6] K. Siozios, D. Soudris and A. Thanailakis, "A Novel Allocation Methodology for Partial and Dynamic Bitstream Generation of FPGA Architectures", Journal of Circuits, Systems, and Computers (JCSC), Vol. 19, No. 3, pp. 701-717, May 2010.

[7] P. Figuli, M. Hübner, R. Girardey, F. Bapp, T. Bruckschlögl, F. Thoma, J. Henkel and J. Becker, "A heterogeneous SoC architecture with embedded virtual FPGA cores and runtime Core Fusion", NASA/ESA Conf. on Adaptive Hardware and Systems (AHS), pp. 96-103, 2011.

[8] Online: http://proteras.microlab.ntua.gr

[9] K. Siozios, K. Tatas, G. Koutroumpezis, D. Soudris, and A. Thanailakis, "An Integrated Framework for Architecture Level Exploration of Reconfigurable Platform", 15th International Conference on Field Programmable Logic and Applications (FPL), pp. 658-661, 2005.