A Synthesizable VHDL Export for the Custom Architecture Design Tool CustArD

Thomas Fabian Starke, Timm Bostelmann, Helga Karafiat and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: starke.thomas@yahoo.de, {bos,kar,saw}@fh-wedel.de

***Abstract*—The research of reconfigurable architectures usually goes hand in hand with a high amount of non-recurring work for Electronic Design Automation (EDA) tool development or adaption. Therefore, in previous work, a heterogeneous architecture template for application domain specific reconfigurable logic was proposed. The goal of this template is to allow the optimization of a reconfigurable architecture towards a specific application domain and to reduce the effort for tool generation in architecture research. In this work, a method to export the described architecture for synthesis is presented. It can be used for a silicon or Field-Programmable Gate Array (FPGA) overlay implementation and thereby extends the usability of the existing design flow. In the future, this work could even be used do derive a detailed timing-model for the designed architectures.**

Keywords–FPGA; architecture design; VHDL; CustArD

I. INTRODUCTION

Highly flexible FPGAs [1] address the demands of fast product lifecycles perfectly, where the non-recurring engineering costs and the slow development process of Application Specific Integrated Circuits (ASICs) are prohibitive. However, the main disadvantage of such flexible, reconfigurable logic structures lies in the vast amount of configuration and communication overhead and hampers their use for high volume or high performance applications. The overhead is caused by the configuration memory of the logic blocks and the routing resources, as well as by the routing network itself. As shown by Kuon et al. in [2] – compared to a standard cell implementation – this overhead increases the area of the chip by a factor of 40 and the power consumption by a factor of 12. Additionally the delay times are increased because the switch- and connection-boxes used for the flexible routing are much slower than fixed connections, resulting in a 3.2 times slower design. A very detailed analysis of the gap between FPGAs and ASICs is presented by the same authors in [3]. It is also shown, that the overhead can be reduced significantly if the right special function blocks (e.g., multipliers or memory) are included in the FPGA design. The main problem herein is, that the demand for special function blocks varies considerably with the application. Obviously, a high amount of unused special function blocks has a direct negative impact on the area efficiency. Moreover, the clock frequency can also be reduced because of longer signal paths between the actually utilized resources. Simply put, flexibility can be traded for efficiency (in a smaller set of applications) and the other way around (see also [4]).

In fact, modern FPGAs are equipped with coarse-grained logic to make them more competitive. Furthermore, many specialized reconfigurable architectures have been proposed by the scientific community over the last decades. For example, a datapath oriented FPGA architecture – implementing parallel routing – has been introduced by Leijten-Nowak et al. in [5]. It reduces the necessary amount of configuration memory by sharing configuration memory bits between routing resources as proposed by Cherepacha et al. in [6]. Furthermore, a highly hierarchical, heterogeneous architecture (*Tree-Based Heterogeneous FPGA Architecture*) was introduced and evaluated by Farooq et al. in [7]. Both techniques are shown to be beneficial for arithmetic intensive applications like Digital Signal Processing (DSP). At the same time, especially the usage of memory sharing reduces the flexibility of the architecture.

Unfortunately, architecture research often demands the development or at least an adaption of a complete toolchain. This makes the exploration of new architectures very time-consuming and complicates the comparison of architectures that have been developed using different tools. As in the papers quoted above, academic toolchains are usually customized towards a rather fixed architecture. They only allow to configure the proposed architecture to some extent, but not to modify its global structure. To some degree the *Verilog-to-Routing (VTR) Project for FPGAs* which has been introduced by Rose et al. in [8] is an exception in this regard. The VTR project offers a very flexible and sophisticated academic development toolchain for FPGAs. It allows a detailed description of a hypothetical architecture, including timing information. Even an export for synthesis has been proposed by Kim et al. in [9]. The architecture description language grants a very flexible definition of Configurable Logic Blocks (CLB). The CLBs can contain for example fracturable lookup-tables, custom routing resources like bus-multiplexers, custom logic and hierarchical clusters [10]. However, the general structure of the architecture – an island-style grid of logic blocks – is still fixed. Special function blocks and different CLBs have to be placed column-wise in the grid, meaning a column can only contain one type of logic. Considering this, even though VTR is great for EDA and island-style FPGA architecture research, its architecture description language is not fully suited for a wide and rapid exploration of the architecture design space as it is envisioned by the authors of this work.

Therefore, a heterogeneous architecture template for application domain specific reconfigurable logic was proposed in [11] by Bostelmann and Sawitzki. A class of reconfigurable architectures – a meta-architecture – which can be flexibly

 ISBN: 978-1-61208-585-2

optimized towards a specific application domain was introduced. It supports hierarchical, heterogeneous structures, as well as parallel datapath connections. In addition, a concept for a corresponding design flow has been proposed in [12]. It allows a directed exploration of application domain specific architectural optimizations. By the derivation of specialized architectures from a very flexible meta-architecture the design flow allows to:

1) Rapidly optimize an architecture towards a specific application domain
2) Improve the comparability between different derived architecture instances
3) Reduce the effort for tool generation

In this work, based on the previous publications mentioned above, an extension for the export of the described architecture to a Hardware Description Language (HDL) is introduced. It can be used for a silicon or FPGA overlay [13] implementation of the designed architecture. The current implementation supports an export as synthesizable Very high speed integrated circuit HDL (VHDL) code with direct support of the Intel FPGA Quartus tools.

The rest of this work is organized as follows. In Section II, the concepts of the meta-architecture and the corresponding design flow that have been proposed in previous work are summarized. In Section III, the implementation of the HDL export is described. In Section IV, exemplary results of the HDL export are presented. Finally, in Section V, this work is summarized and an outlook to further work is given.

II. Background

This section is split into two parts. First the degrees of freedom and the global structure of the meta-architecture used in this work are described and then the concept of the extended design flow is depicted.

A. Meta-Architecture Description

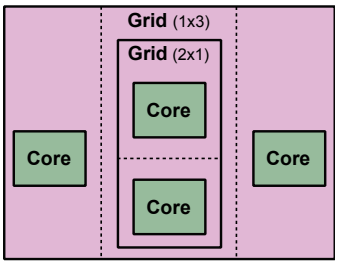
The reconfigurable architecture is described on function block level. A basic set of configurable function blocks is provided, but can also be extended by the user. The basic set consists of the following function blocks:

LUT	lookup-table
REG	register or single flipflop
MUX	multiplexer
MEM	memory
IO	input and output buffer
SB	switchbox (bi- or unidirectional)
CB	connectionbox (bi- or unidirectional)
IP	fixed IP core from a library
GRID	grid of blocks or grids

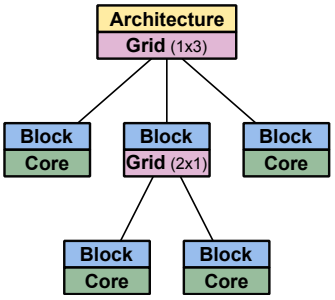
The global structure of the architecture is based on two types of grid. The first one is a 'repeating grid' which repeats one block (or an other grid) $n \times m$ times. The second one is a 'custom grid' which can contain a custom compilation of blocks (or other grids like shown in Figure 1). By supporting recursive grids the design of highly hierarchical, tree-based architectures without a huge top-level grid is encouraged. Of course, the description of flat island-style architectures is possible as well, by simply using a top-level 'repeating grid'.

B. Design Flow

The design flow for the meta-architecture described above is shown in Figure 2 as proposed in [11]. The user creates an architecture in a graphical architecture design tool called Custom Architecture Design Tool (CustArD) or derives it from a template. The design tool was first presented by Sternberg et al. in [14]. CustArD exports an internal Architecture Description File (ADF), as well as an HDL description of the architecture. This feature is the major topic of this work. The user can then select a set of reference or benchmark applications, which are synthesized and mapped to the architecture based on the ADF. After this step, an analysis tool provides the user with an early evaluation of the block utilization. This is especially interesting if the impact of special function blocks for a given set of applications is explored. It allows for example a directed optimization of the provided resources towards a specific application domain. After a complete toolchain iteration the analysis tool provides a detailed evaluation including benchmark results and the utilization of routing resources. This can be used to explore new routing techniques or again to optimize the architecture towards a specific application domain, for example by adapting the width of parallel datapath routing elements.



(a) Flat view of the recursive grid structure



(b) Tree view of the recursive grid structure

Figure 1. An exemplary recursive grid structre, consisting of two grids and four cores

III. Implementation

The VHDL export plugin maps the CustArD representation of an architecture into a VHDL representation. This is achieved by mapping each CustArD architecture component to its corresponding VHDL representation.

A. VHDL Primitives

The architecture in CustArD consists of primitives whose complexity can vary from a simple logic gate to a complex Intellectual Property (IP) core. For the VHDL export the following primitives are used: logic gate (AND, NAND, NOR, NOT, OR, XNOR, XOR), multiplexer, flipflop, Lookup Table (LUT), selector, wire, Static Random-Access Memory (SRAM), Connection-Box (CB) and Switch-Box (SB). Further primitives can be added through storage of their VHDL

 ISBN: 978-1-61208-585-2

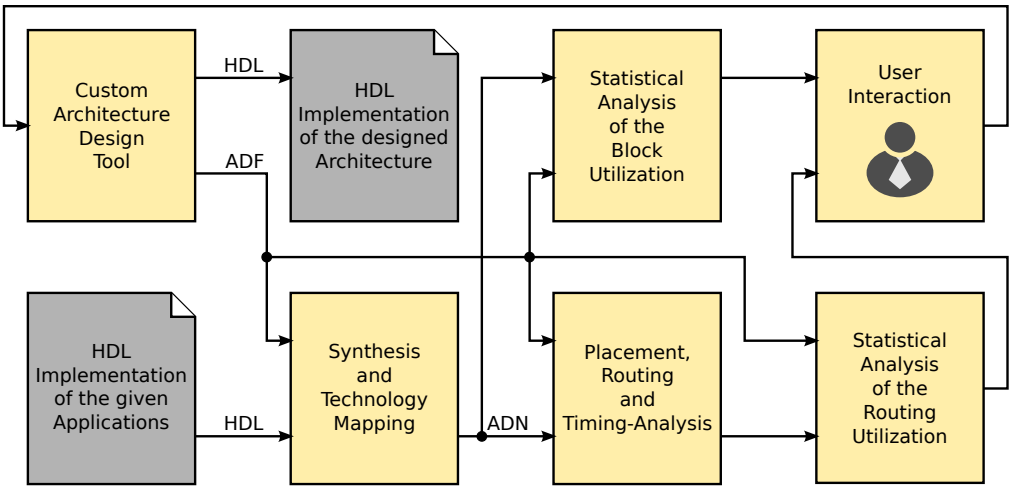


Figure 2. A Flowchart of the described design flow for heterogeneous reconfigurable architectures

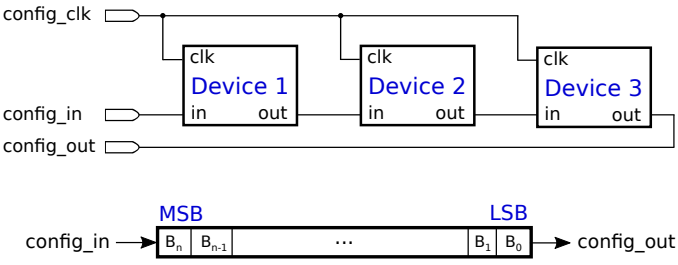


Figure 3. Schematic of the global organization of the configuration memory

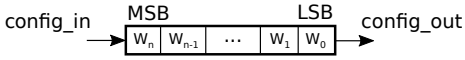


Figure 4. Organization of the configuration memory for a LUT

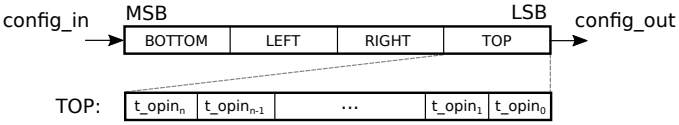


Figure 5. Organization of the configuration memory for a switchbox

representations in the export plugin.

Global Signals: Currently only fully synchronous designs are supported. As a result the signals *clk* and *reset* that are needed by some VHDL primitives like a flipflop are handled globally. In the VHDL export, they are marked with the prefix *global_* in all primitives that make use of them.

Configuration Signals: To use the exported architecture for an application some components like LUT or switchboxes have to be configured. This is done through a JTAG-like configuration mechanism with a shift register. All components that need configuration are serially connected as shown in Figure 3 and store the configuration information internally. The length of the configuration register depends on the information that is needed for each component. The signals used for configuration are marked with the prefix *config_* in the corresponding VHDL primitives.

Wire: A wire connects an input signal directly to an output signal. It is the most simple primitive in CustArD. Its export could have been implemented with a special treatment which would have reduced the complexity of the VHDL code. However, a special treatment would lead to a higher implementation complexity and less consistency. Therefore, even this trivial component is exported as a VHDL primitive.

Lookup Table: A LUT stores precomputed information that is available at runtime. The data is stored in the LUT during the configuration. Therefore the VHDL primitive of a LUT contains a configuration register (see Figure 4). It consists of $n = 2^{asw}$ words, where as *asw* stands for address signal width. The size W of a word corresponds to the output signal width *osw* of the LUT. As a result, the overall size of the configuration register has $W \cdot 2^{asw}$ bits.

SRAM: The SRAM is like a LUT, except that the information can be stored and changed at runtime. It is assumed that the input vector width *isw* is equal to the output vector width *osw* and that the number of words stored in the SRAM is 2^{asw}, where *asw* stands for address signal width. The resulting size of the SRAM is $2^{asw} \cdot isw$ bits.

Selector: The selector is similar to a multiplexer except that the output signal is chosen during configuration time instead of runtime. Therefore this component needs a configuration register of size $\mathrm{ld}(isw)$ bits.

Switchbox: The standard switchbox in CustArD uses bidirectional pins. In order to minimize the hardware complexity, an advanced switchbox has been implemented which has fixed input and output pins on each side. The internal connection structure was implemented as a disjoint switchbox. As a simplification, it is additionally defined that the amount of the vertical input and output pins must be the same as the amount of the horizontal ones. The configuration register (see Figure 5) defines in which way the input pins and output pins are connected, according to the contraints of a unidirectional disjoint switchbox.

Connectionbox: The connectionbox is a special form of a switch box and is used for configurable connections between logic elements and horizontal or vertical connecting structures. In order to provide maximum connectivity, any input pin can be assigned to any output pin of the connection box. Figure 6

 ISBN: 978-1-61208-585-2

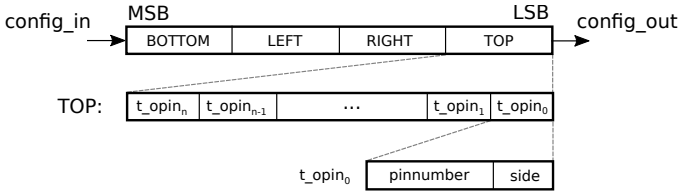


Figure 6. Organization of the configuration memory for a connectionbox

shows the structure of the configuration register within the connectionbox.

B. VHDL-Export-Plugin

The HDL-export traverses the architecture tree and calls the corresponding processing routine for every node, which then passes the calculated information towards the root. The utilzed routines are described below.

Core: The core object represents a leaf of the architecture tree. It is directly translated into a VHDL primitive. Therefore, the entity declaration of a VHDL file is read and the signal names, as well as the generic identifiers are mapped to the identifiers used in CustArD. The generic mapping declaration is designed from the configuration values deposited in CustArD. It is stored in VHDL objects. All related CustArD signals are stored in a dictionary like $\{Pingroup : [(Pinnumber, [(Terminal\_1, Terminal\_2, Net)^*])^*]\}$, where *Terminal_1* corresponds to the source and *Terminal_2* to the sink.

IO-cores which are the in- and output of the architecture, are processed differently. For them no VHDL primitives are created, but the signals of the IO-cores are integrated into a global list that is only processed at the root element of the tree.

Grid: A grid represents a node of the architecture tree and is translated into a VHDL file. For all elements of the grid the following steps are processed.

1) A VHDL file is created for each of the subtrees through a call of the block method.
2) The instantiation of the VHDL file is created and inserted into the grid VHDL file object including the creation and storage of the port and generic map declaration.
3) The signals of the instance are entered into the grid VHDL file as local signals.
4) If there are configuration signals within the port of the instantiated VHDL object, the configuration bus is extended by this object and the configuration action signals within the grid are adjusted.

All detected signals from the VHDL instances are assigned by a matching algorithm. All signals that couldn't be matched, lead to different parts of the architecture tree and are therefore passed up to the next instance.

Block: A block is a container class within the CustArD architecture tree and, in addition to a grid or a core element, contains further meta information, such as a unique block ID and a referenced flag. The processing method of a block, first creates the associated VHDL file object from the grid or core element. If this is not a primitive object, the associated VHDL file is generated from the VHDL file object. The determined signals of the subtrees are used as inputs or outputs of this VHDL entity and are specified in the port declaration. Blocks can also be used several times within the architecture tree. This

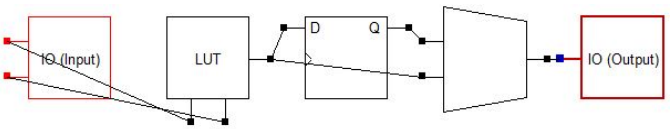


Figure 7. Imlementation of a simple CLB in CustArD

TABLE I. EXPORT TIME FOR AN ARCHITECTURE CONSISTING OF TWO PRIMITIVES IN A 5 X 5 GRID

Number of connections	time / s	used memory / MB
0	0.63	<1.5
320	0.68	<1.5

is indicated by the reference flag within the block. If this is the case, the obtained information about the interface of these subtrees is stored in a global block cache. If such a block is recompiled, the translation can be interrupted directly at this level and replaced by the cached content, which significantly reduces the translation effort and the number of resulting VHDL files.

Architecture: The architecture element represents the root of the architecture tree. Similar to the block, it can contain a core or a grid element. The processing of this node is similar to that of the block, but all internal signals must be mapped at this level so that only the IO-core signals, as well as the $config\_$ and $global\_$ signals are included.

IV. RESULTS

In this section, the runtime and memory usage of the VHDL export plugin are discussed, depending on the number of connections, number of used primitives and the architecture tree depth. Figure 7 shows a simple CLB architecture designed in CustArD which consists of one 1 x 5 grid and five different primitives. The result of the VHDL export plugin consists of three VHDL primitives and one VHDL file representing the grid of the CLB architecture. Figure 8 shows a Register Transfer Level (RTL) plot of the export in Intel FPGA Quartus.

Table I shows the export time for an architecture consisting of two primitives in a 5 x 5 grid, with and without wiring. It shows that the number of connections has only little influence on the runtime of the export plugin. Table II shows the export time for an architecture consisting of ten different primitives in a architecture tree with a depth of four. The runtime of the export plugin depends on the number of primitives and the maximum depth of the architecture tree. This is manageable even for larger structures, since they usually consist of referenced blocks or grids that are used multiple times but are exported only once.

TABLE II. EXPORT TIME FOR AN ARCHITECTURE CONSISTING OF TEN DIFFERENT PRIMITIVES IN A ARCHITECTURE TREE WITH THE DEPTH OF FOUR

Number of connections	time / s	used memory / MB
0	0.50	<1.0
10	0.50	<1.0
30	0.65	<1.0

 ISBN: 978-1-61208-585-2

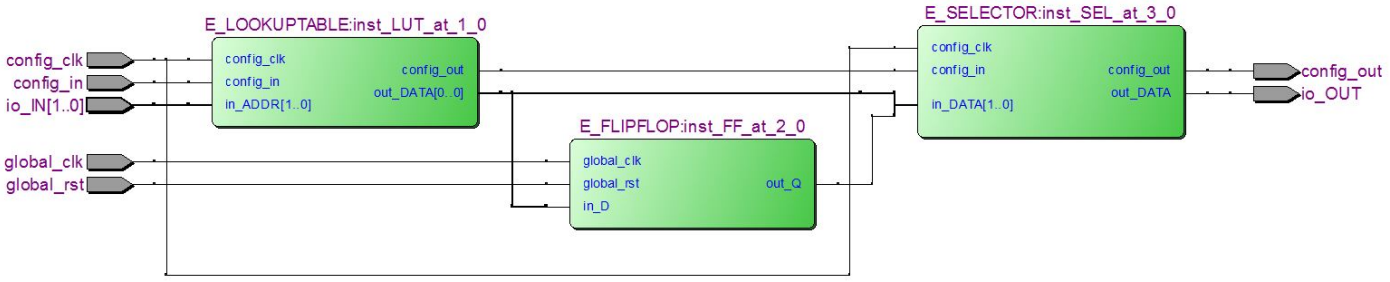


Figure 8. Intel FPGA Quartus RTL view of the exported CLB shown in Figure 7

V. Conclusion and Future Work

In this work, an extension for the custom architecture design tool CustArD was presented. It was shown how a reconfigurable architecture that has been optimized towards a specific application domain can be exported to a synthesizable VHDL format. The results for a simple LUT design were presented. Furthermore, it was shown that the export is reasonably fast for small designs. The automatic generation of project files for the Intel FPGA Quartus tools allow a seamless utilization of the results.

In future work, an export to the Verilog HDL language is planned to establish consistency between the input and output file formats. Since the internal data-structures and concepts are HDL-independent, this should not be very complicated. Furthermore, larger architectures should be benchmarked to show the real-world applicability of this approach. Finally, the usability of this export tool could be increased even further by the creation of more building blocks (e.g., configurable DSP blocks).

References

[1] W. Carter et al., “A user programmable reconfigurable logic array,” in IEEE Custom Integrated Circuits Conference (CICC), 1986, pp. 233–235.

[2] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, Feb 2007, pp. 203–215.

[3] I. Kuon and J. Rose, Quantifying and Exploring the Gap Between FPGAs and ASICs, 1st ed. Springer Publishing Company, Incorporated, 2009.

[4] H. Parvez, Z. Marrakchi, and H. Mehrez, “ASIF: Application specific inflexible FPGA,” in Field-Programmable Technology (FPT), Dec 2009, pp. 112–119.

[5] K. Leijten-Nowak and J. L. van Meerbergen, “An FPGA architecture with enhanced datapath functionality,” in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2003, pp. 195–204.

[6] D. Cherepacha and D. Lewis, “DP-FPGA: An FPGA architecture optimized for datapaths,” VLSI Design, vol. 4, no. 4, 1996, pp. 329–343.

[7] U. Farooq, Z. Marrakchi, and H. Mehrez, Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization. Springer, 2012.

[8] J. Rose et al., “The VTR project: Architecture and CAD for FPGAs from Verilog to routing,” in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2012, pp. 77–86.

[9] J. H. Kim and J. H. Anderson, “Synthesizable FPGA fabrics targetable by the Verilog-to-Routing (VTR) CAD flow,” in 2015 25th International Conference on Field Programmable Logic and Applications (FPL), Sept 2015, pp. 1–8.

[10] J. Luu, J. H. Anderson, and J. Rose, “Architecture description and packing for logic blocks with hierarchy, modes and complex interconnect,” in ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2011, pp. 227–236.

[11] T. Bostelmann and S. Sawitzki, “A heterogeneous architecture template for application domain specific reconfigurable logic,” in 2015 Austrian Workshop on Microelectronics (Austrochip), Sept 2015, pp. 9–14.

[12] T. Bostelmann and S. Sawitzki, “Towards a guided design flow for heterogeneous reconfigurable architectures,” in 2015 25th International Conference on Field Programmable Logic and Applications (FPL), Sept 2015, pp. 1–2.

[13] A. Brant and G. Lemieux, “ZUMA: An open FPGA overlay architecture,” in Field-Programmable Custom Computing Machines (FCCM), April 2012, pp. 93–96.

[14] H. Sternberg, T. Bostelmann, and S. Sawitzki, “CustArD - a custom architecture design tool,” presented at the Technical Demonstrations Session of the International Conference on Reconfigurable Computing and FPGAs (ReConFig), Dec 2014, p. 2.

 ISBN: 978-1-61208-585-2