2016 IEEE International Parallel and Distributed Processing Symposium Workshops

A fully parameterized Virtual Coarse Grained Reconfigurable Array for High Performance Computing Applications

Amit Kulkarni, Elias Vansteenkiste and Dirk Stroobandt
ELIS department, Computer Systems Lab, Ghent University,
Sint-Pietersnieuwstraat 41, Ghent B-9000, Belgium
Email:{Amit.Kulkarni, Elias.Vansteenkiste, Dirk.Stroobandt}@UGent.be

Andreas Brokalakis and Antonios Nikitakis
Synelixis Solutions Ltd,
Farmakidou 10, 34100, Chalkida, Greece
Email:{brokalakis, nikitakis}@synelixis.com

***Abstract*—Field Programmable Gate Arrays (FPGAs) have proven their potential in accelerating High Performance Computing (HPC) Applications. Conventionally such accelerators predominantly use, FPGAs that contain fine-grained elements such as LookUp Tables (LUTs), Switch Blocks (SB) and Connection Blocks (CB) as basic programmable logic blocks. However, the conventional implementation suffers from high reconfiguration and development costs. In order to solve this problem, programmable logic components are defined at a virtual higher abstraction level. These components are called Processing Elements (PEs) and the group of PEs along with the inter-connection network form an architecture called a Virtual Coarse-Grained Reconfigurable Array (VCGRA). The abstraction helps to reconfigure the PEs faster at the intermediate level than at the lower-level of an FPGA.**

Conventional VCGRA implementations (built on top of the lower levels of the FPGA) use functional resources such as LUTs to establish required connections (intra-connect) within a PE. In this paper, we propose to use the parameterized reconfiguration technique to implement the intra-connections of each PE with the aim to reduce the FPGA resource utilization (LUTs). The technique is used to parameterize the intra-connections with parameters that only change their value infrequently (whenever a new VCGRA function has to be reconfigured) and that are implemented as constants. Since the design is optimized for these constants at every moment in time, this reduces the resource utilization. Further, inter-connections (network between the multiple PEs) of the VCGRA grid can also be parameterized so that both the inter- and intra-connect network of the VCGRA grid can be mapped onto the physical switch blocks of the FPGA. For every change in parameter values a specialized bitstream is generated on the fly and the FPGA is reconfigured using the parameterized run-time reconfiguration technique. Our results show a drastic reduction in FPGA LUT resource utilization in the PE by at least 30% and in the intra-network of the PE by 31% when implementing an HPC application.

***Keywords*-FPGA; Reconfiguration; DCS; TLUT; TCON; PE;**

I. Introduction

FPGAs have been consistently proving their importance in accelerating high performance computing applications. They are used as auxiliary resource for CPUs replacing GPUs or DSPs for intense data manipulation applications. However, the FPGAs incur a heavy development cost compared to the processing units (CPUs, GPUs and DSPs). The programming of the FPGAs (called configuration) is usually done starting from the RTL that describes a circuit in the lower abstraction level (gate level) and therefore the compilation time is high, thus resulting in a slow design cycle. However, this limitation is not present for processing units (CPUs, GPUs and DSPs) since they can be easily programmed at a higher abstraction level and hence they have very low development cost and shorter time-to-market [1].

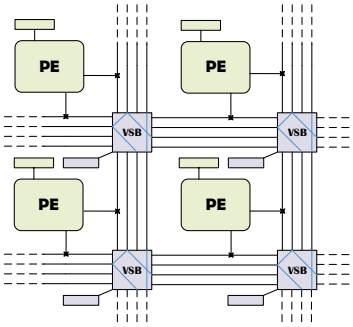

Figure 1. A fragment of VCGRA grid with Processing Elements (PEs), Virtual Switch Blocks (VSB) and corresponding settings registers (rectangles)

In order to overcome this limitation for the FPGAs, VCGRAs are proposed [2]. The programming model for the VCGRAs is different by the fact that the code can be written on a higher abstraction level. This reduces the compilation time by several orders of magnitude as compared to the fine-grained FPGA, thus VCGRAs act as intermediate virtual fabrics [3] to curb the development costs. Figure 1 shows a fragment of a VCGRA. The architecture consists of groups of coarse-grained processing elements connected using virtual connection blocks and switch blocks forming a communication network (inter-connect). The processing elements are powerful and more complex than a LUT and are defined at a higher abstraction level. The complexity of the processing elements can range from a simple ALU to a fully capable RISC processor.

Each PE has a settings register used to configure a function of the PE. With the proper connection settings (configured in the settings register of the VSB - Virtual Switch Block), every application that uses these PEs can be implemented. The settings registers are updated using a dedicated bus that enables us to reconfigure the settings of the PEs and VSBs.

Authors in [2] have shown that conventional VCGRA

DOI 10.1109/IPDPSW.2016.13

implementations use LUTs of the FPGA to realize reconfigurable virtual switch blocks, virtual registers and other virtual components. With the help of parameterized configuration, the authors were able to map the virtual components to the lower level physical resources (settings registers were mapped onto configuration memory and VSBs were mapped onto switch blocks) of an FPGA. This saves a significant amount of functional resources (LUTs) of an FPGA.

However, the work in [2] was limited to parameterize the LUTs of the PEs and inter-connect (VSBs) of the VCGRA only, thus resulting in a semi-parameterized VCGRA. In this paper, our main contribution is to parameterize not only the PEs and VSBs, but also the connections within each PE (intra-connect) using Tunable Connections (TCONs) [4]. The parameterized intra-connections can be automatically mapped to the lower level physical connection blocks and switch blocks of the FPGAs, thus avoiding the use of LUTs to implement the intra-connect. Therefore, we save an additional amount of physical LUT resources of the FPGA.

The rest of the paper is organized as follows: in Section II, we describe the VCGRA tool flow and how it makes use of TLUTs and TCONs. In Section III, we describe a fully parameterized VCGRA that contains TLUTs in the PEs and TCONs for the inter- and intra-connections of the VCGRA. We evaluate our VCGRA on a high performance computing medical application called Retinal Vessel Segmentation which is described in Section IV. In Section V, we present our results and discuss the improvements in the FPGA resource gain. Finally, we conclude in Section VI.

II. Parameterized Configuration for VCGRAs

The main contribution of introducing the VCGRA concept is explained next. It results in splitting up the tool flow in a high level tool flow to be used by the application designer (Section II-A) and a low level tool flow that is only used to generate the proper VCGRA components in an automatic and reconfigurable way (explained in Section II-B).

A. VCGRA tool flow

Figure 2, explains the tool flow for VCGRA design. The VCGRA tool flow makes use of a VCGRA architecture that defines the granularity as well as the possible functionality of the PEs and describes the possible ways the PEs are interconnected. The implementations of the PEs and VSBs is performed in the parameterized reconfiguration tool flow (left hand side of Figure 2) and is explained in Section II-B.

In the VCGRA tool flow (right hand side of Figure 2), the user will determine the VCGRA settings that will configure the required configurable components of the VCGRA to realize the desired application.

The higher level VCGRA tool flow that produces these VCGRA settings consists of a synthesis and a mapping tool in which the textual description of the application design is parsed and converted into a netlist of Processing Elements

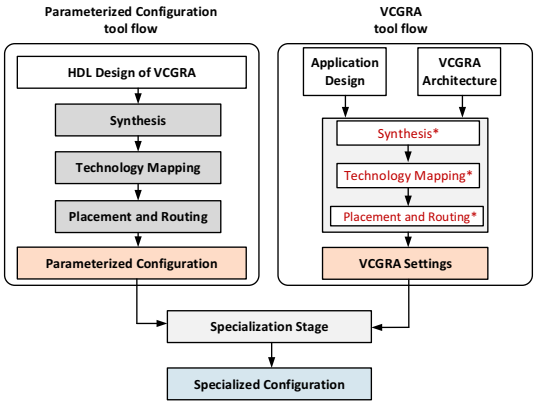

Figure 2. Implementation of applications on a VCGRA using the parameterized configuration tool flow
Note: * indicates steps considering PEs as a basic programmable component

(PEs). Next, we perform placement of the synthesized netlist of PEs on to the virtual PEs of the VCGRA architecture. The tool flow has to take care that all the inter- and intra-connect links of the VCGRA are implemented on the VCGRA architecture's communication network. We make use of a router to establish optimal connections between the placed elements of the VCGRA architecture. The Place and Route (PaR) result determines what the functionality of each PE is and how the communication network is exactly used and that this is reflected in the VCGRA settings.

Because the basic programmable element in the VCGRA tool flow is a PE, the tools (synthesis, mapper, place and route) need considerably less complexity and time to generate the settings values than the standard FPGA compilation would. This is because the higher abstraction level reduces the problem size and therefore the tools are faster. If the application design specification changes with the same VCGRA platform then we can generate the settings values much faster than processing the new design with the standard FPGA tool flow.

Using the parameterized reconfiguration flow gives a set of parameterized VCGRA components. The settings values are then combined with these parameterized components in the specialization stage and this results in the final reconfiguration bitstreams automatically.

B. Tool flow for parameterized configuration

The VCGRA tool flow builds on top of the parameterized reconfiguration tool flow, which provides a parameterized version of the VCGRA building blocks (left side of Figure 2). Figure 3 shows more details on the tool flow for generating a parameterized configuration for a general parameterized application. An application is said to be parameterized if some of its input values change infrequently compared to the rest. The infrequently changing input values are called parameters.

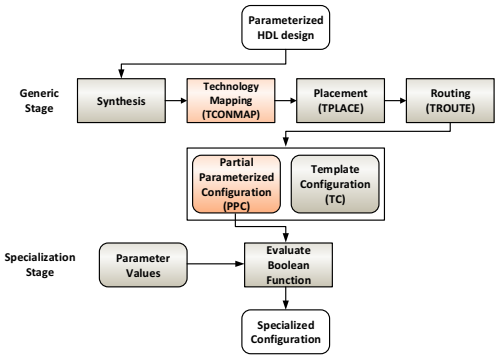

Figure 3. Tool flow for parameterized configuration

There are two stages: a *generic* stage and a *specialization* stage. In the *generic* stage, a HDL design with parameterized inputs (annotated with –PARAM) describes the application. The application is processed to yield a Template Configuration (TC) and a Partial Parameterized Configuration (PPC). A detailed explanation on each step of the generic stage is explained in [5].

The final output of the generic stage is the TC and PPC. The TC contains static bits (0's and 1's) which are the non-reconfigurable parts of the problem. The PPC contains multi-port Boolean functions of the parameter inputs. In order to generate specialized bitstreams, the PPC has to undergo the specialization stage.

In the *specialization* stage, for every change in parameter values, the Boolean functions are evaluated by a Specialized Configuration Generator (SCG) to generate specialized bitstreams. The SCG takes a specific parameter value and evaluates the Boolean functions to produce specialized bits. The SCG can be implemented on an embedded processor (PowerPC, ARM or MicroBlaze) present in the FPGA. With the help of a configuration interface such as HWICAP or MiCAP [6] [7], the FPGA is reconfigured with the specialized bitstream. The technique is called Dynamic Circuit Specialization (DCS).

Upon a value change in parameter inputs, the configuration bits of the TLUTs and TCONs are reconfigured with specialized bits that are thus generated after evaluation of the Boolean functions for a specific set of parameter values.

This means that in the parameterized VCGRA approach, the settings registers are mapped onto the configuration memory of the FPGA which is in contrast with conventional implementation that maps the registers to the flip-flops of a logic cell present in the FPGA. Therefore, a significant amount of flip-flops can be saved.

As long as we are only reconfiguring LUTs, we are able to parameterize LUTs on commercial Xilinx FPGAs. However, parameterization of other primitives such as routing switches can be done only on a hypothetical FPGA but not on the commercial FPGAs (because we do not have access to the low level routing reconfiguration infrastructure). Hence, the tool flow explained above is used to generate the parameterized configuration for a hypothetical FPGA. Therefore, the implementation is limited up to the place and route steps only.

C. Limitation of parameterized VCGRAs

In a conventional VCGRA implementation, the settings registers are updated using a dedicated bus. However, in the parameterized VCGRA implementation, the settings registers of each PE and the routing switches are updated by reconfiguring each frame of the FPGA that contains setting bits of the VCGRA. This is usually accomplished by read-modify and write back frames of the FPGA (*micro-reconfiguration* [5]).

For a VCGRA application that contains dynamic Network-On-Chips or PEs that require cycle-by-cycle context switching, we cannot afford the cost of reconfiguration time so often and therefore, such applications may not be suitable to be handled by a parameterized VCGRA.

However, in the case of much less frequent reconfiguration needs, the parameterized reconfiguration reduces the overhead of the conventional VCGRA as follows:

- The settings registers of the VCGRA are mapped on the configuration memory and therefore, the need of a dedicated bus to update the settings register is avoided.
- The PEs of the VCGRA are optimized by symbolic constant propagation that is integrated within the parameterized configuration tool flow.
- Each VCGRA intra- and inter-connection is mapped onto lower level reconfigurable routing switches (TCONs). Therefore, we reduce the utilization of the LUTs for implementing the connection network.

Even with the limitations of VCGRAs we believe that a large number of VCGRA implementations [3] [8] [9] [10] in which cycle-by-cycle context switching is not needed, can benefit from the above advantages of the fully parameterized VCGRA implementation.

III. Fully Parameterized VCGRA

In the previous work on parameterized VCGRA implementations, authors of [2] parameterized the LUTs and interconnects of the VCGRA. They were able to save 50% of the LUT resources. However, they did not parameterize the intra-connect of the VCGRA (connections within a single PE) since they had no automatic technology mapper to generate TCONs for the intra-connections. In order to overcome this limitation we use an automatic technology mapper called TCONMAP [4] that can generate TCONs and TLUTs simultaneously for a given parameterized application.

In Figure 4, a fully parameterized PE is depicted. Each PE in the VCGRA grid contains Tunable LookUp Tables (TLUTs) optimized to implement the required functions of the PE. The optimization is achieved by the method for optimization for constant parameters as a result of

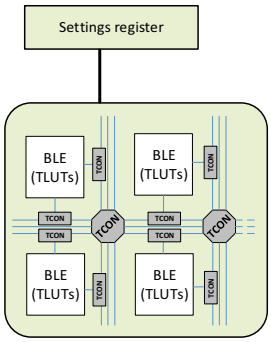

Figure 4. A fully parameterized Processing Element (PE) containing Tunable LUTs (TLUTs) and Tunable Connections (TCONs) within a single PE

parameterization of the LUTs. In this approach, instead of implementing the parameter inputs of the application as regular inputs, they are implemented as constants and the functions of each TLUT are specialized for these constants. For every change in parameter input values, the function of the TLUT is re-optimized for new constant values by reconfiguring the configuration of the TLUT. A group of TLUTs form a Basic Logic Element (BLE) of the PE.

The BLEs of the PE are connected using virtual routing switches (connection blocks and switch blocks) within the PE that form an intra-connect. The PE also contains a virtual routing network composed of wires that is responsible of carrying required signals between the BLEs. A virtual routing switch consists of connection multiplexers with configuration memory. The routing switch connects the wires between BLEs within the routing network depending on the configuration values stored in the configuration memory of the switch and therefore providing an opportunity to parameterize the routing network.

TCONMAP replaces the virtual routing switches with the TCONs. A TCON consists of configuration memory that can be reconfigured depending on the parameter inputs. Therefore, a connection between two BLEs can be made or broken depending on the values of the parameter inputs of a VCGRA application. Further, with the help of TPLACE and TROUTE, these connections can be placed and routed on to the lower-level physical routing switches thereby reducing the PE intra-connect overhead on the physical LUTs of the FPGA. With the help of TCONs, a significant reduction in routing resource consumption (at least by 40%) has been observed in the experiments of [11]. We aim at similar improvements by using the TCON concept on a VCGRA implementation of a retinal vessel segmentation application.

IV. Retinal Vessel Segmentation Application

In this Section, we present an HPC application that is used to investigate the benefits of the fully parameterized VCGRA approach. We have designed the PEs of the VCGRA based on the HPC application. Only those parts of the application that will be implemented on the reconfigurable logic (hardware modules) for the performance acceleration will be used for the VCGRA implementation.

In computer vision, segmentation refers to the process of partitioning a digital image in multiple segments in order to extract prominent features and locate objects and/or boundaries. The particular application of interest - retinal vessel segmentation - refers to the extraction of the vessel structure from the background in fundus images. Vessel segmentation enables the extraction of morphological attributes of retinal blood vessels, such as length, width and branching pattern, that assist the diagnosis, screening, treatment and evaluation of various cardiovascular and ophthalmologic diseases such as diabetes, hypertension, arteriosclerosis and choroidal neovascularization.

The Retinal Vessel Segmentation application that we have implemented is based on the concept of matched filters [12] and is presented in Figure 5. From an initial 2D input retinal image (RGB image), the green channel is retained as it contains most of the information. A preprocessing step is then applied in order to provide a more suitable and clear image for the main filtering operations. The preprocessing involves histogram equalization, optic disc removal and outer region removal.

The resulting image goes through a denoising function by means of a Gaussian filter to reduce the effect of high frequency noise (applying 2 set of coefficients of 5×5 and 9×9 respectively). The main vessel detection function that follows, involves filtering and thresholding the denoised image. Since the cross-section of a vessel can be modeled as a Gaussian function, a series of Gaussian-shaped filters can be used to "match" the vessels for detection. Steerable filters are used (in the current implementation, seven directions are considered) to separate the pixels with the strongest responses (7 different sets of 16×16 coefficients). The problem with this approach is that not only vessels but also non-vessel edges can be identified in the response image. To minimize this effect, a third processing step that involves texture filtering is applied so as to retain in the final image only lines of certain thickness and above. The vessels filtering is also applied in the form of a modified filter applied many times in the image with different sizes depending on the desired filtering effect (e.g 5×5, 9×9, 16×6). In general the number of filters applied in the pipeline is a tunable parameter which depends on the quality of the images and/or imaging technology is used.

Figure 5 presents an overview of the application. It should be noted that the preprocessing steps are implemented in software, while all filtering operations are implemented as hardware modules. All hardware modules employ the same interfaces and are virtually the same in principle: they all share the same core architecture and what changes is size and coefficients of the filter kernels. The orientation of the filter is defined from the coefficients itself.

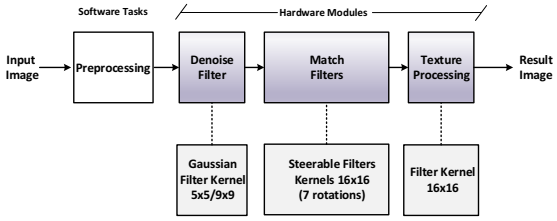

Figure 5. High level presentation of the processing steps for the retinal vessel segmentation application

VCGRA for the HPC application

The filters (hardware modules) of the HPC application described above need to be accelerated by realizing the filter actions on the reconfigurable logic. We use our fully parameterized VCGRA approach to implement the filters on an FPGA. We used a floating point Multiply-Accumulate (MAC) operator as a processing element. We have used the "FloPoCo" [13] floating point library to build the floating point addition and multiplication and thus, we use the FloPoCo floating point format with a 6-bit exponent and a 26-bit mantissa. We have not used any dedicated multipliers or adders while generating the floating point operators using the "FloPoCo" library.

In the MAC operation, the image samples are multiplied with the filter coefficients. Later, they are added to the previously accumulated values after the multiplication. The coefficients of the filter determine the filter configurations such as the noise level of the denoise filter of the vessel segmentation application.

The floating point multiplication is parameterized with the coefficient as a parameter input. The value of the coefficient input changes infrequently. For each infrequent change in the coefficient value, a specialized bitstream is generated and the multiplication is reconfigured accordingly. The settings register for each MAC operator (PE) holds an integer value for the counter that decides the number of iterations the MAC operation should perform with a fixed coefficient value. Therefore, in order to change the filter coefficients and counter values, each PE (MAC operator) needs to be reconfigured.

V. Results and Discussion

The Processing Element (PE) of the filter application (MAC operator) was described using VHDL with annotated parameter inputs (–PARAM). The annotation helps to differentiate between the regular inputs and the parameterized inputs. With the help of Quartus II (v10.0), the PE was synthesized and later subjected to logic optimization by using the ABC tool [14]. We used the TCONMAP mapper [4] to generate TLUTs and TCONs.

The LUT resource utilization of a single PE is tabulated in Table I. Clearly, the total number of 4-input LUTs utilized by the PE with our VCGRA approach shows a significant reduction by $\approx$ 30%. We also observe a difference in the logic depth level by 3 and hence it contributes to the improvement in the performance of the PE.

Table I
Resource utilization and PaR results of a PE

VCGRA	LUTs (TLUTs)	TCONs	Logic Depth level	WL	CW
Conventional	2522(0)	0	36	27242	10
Fully Parameterized	1802(526)	568	33	16824	10

All the TCONs (568) can be implemented on the physical switch blocks and connection blocks, instead of LUTs, thus saving a significant amount of LUT resources of the FPGA. In the conventional VCGRA implementation, these TCONs are realized on the LUTs which is an overhead of $\approx$31% of the total LUTs of the parameterized VCGRA. Therefore, we reduce an overhead of $\approx$31% of LUT resources of the intra-network of each PE.

The synthesized PE was subjected to a Place and Route (PaR) tool using the TPaR CAD tool [11]. The PaR was performed using the *4LUT_sanitized* FPGA architecture from the VPR [15]. The results of PaR for a single PE (MAC operator) are tabulated in Table I. Clearly, the proposed method (fully parameterized VCGRA) has the total wire length (WL) decreased by $\approx$ 31% as compared to the conventional VCGRA implementation, thus saving a significant amount of routing wire resources of the FPGA.

The minimum channel width (CW) of the experiments presented in [2] [11] show an increase in the minimum channel width when using TCONs. However, our results show that the minimum routing channel width of both implementations are the same. We observe no overhead on the minimum channel width of the FPGA after using TCONs for the inter- and intra-connections of the VCGRA.

A fully parameterized 4x4 VCGRA grid:

The resources utilized by a 4x4 VCGRA grid are tabulated in Table II. The resource utilized by the grid contains 16 PEs and 9 VSBs, each of them has a settings register and therefore, the conventional VCGRA would consume twenty five 32-bit registers. In the conventional implementation, these registers are realized using the FPGA's logic-cell flip-flops. However, with the parameterized VCGRA tool flow we map them to the configuration memory of the FPGA and hence we reduce the flip-flops utilization to zero.

Also, in the conventional VCGRA implementation the routing switches (connection blocks + switch blocks) are needed to realize a 4x4 VCGRA grid is 41 (9 VSBs and 32 Virtual Connection blocks) and again these would have to be realized on the LUTs of the FPGA. However, with our fully parameterized VCGRA implementation we can target physical routing resources and thereby reducing the functional resource utilization (LUTs) to zero.

Table II
RESOURCE UTILIZATION OF A 4X4 VCGRA GRID

VCGRA	Inter-Network	Settings register
Conventional	41	25
Fully Parameterized	0	0

With the use of parameterized VCGRA configuration, a significant reduction in FPGA resource utilization is observed. However, this gain does not come for free. There exist a reconfiguration overhead such as reconfiguration time, Boolean function evaluation time and the PPC memory [5]. The estimated reconfiguration time depending on the number of TLUTs and TCONs for one PE is 251 ms. The reconfiguration speed can be improved using the techniques described in [16].

In the vessel segmentation application, the coefficients of the Gaussian filter and the texture processing filter change infrequently and is user configurable. Therefore, the reconfiguration time cost for these two filters is minimal. For example, for 1000 images (of same size) can be denoised and its texture is processed at the reconfiguration time cost of 251 ms per PE per 1000 images. Therefore, the two filters benefit from the parameterized reconfiguration technique along with the advantage of the VCGRA tool flow.

VI. CONCLUSION

In this paper, we presented an extended, complete method to parameterize the intra-network of the Processing Elements (PEs) of a parameterized virtual coarse-grained overlay architecture for FPGAs. The virtual architecture was fully parameterized (using the parameterized configuration tool flow) in the sense that the processing elements containing both intra- and inter-connections of the VCGRA grid were parameterized. The results showed an improvement in the optimization of the PE by 30%. The logic depth level was also reduced by ≈9% which helps in improving the performance of the PE. The intra-connects (TCONs) are realized on the physical routing switches, hence we reduced ≈31% of the LUTs for each PE. The results of PaR have proved that by using the proposed method, we saved 31% of wire length and no overheads in the routing resource utilization were observed. With the help of parameterized configuration we were able to avoid excess LUT utilization for the implementation of the VCGRA's intra- and inter-network by targeting physical routing switches of the FPGA. The main advantage of using the proposed VCGRA approach is: for any changes in specification of the VCGRA design with the same VCGRA architecture, the VCGRA tool flow can be used to overcome the limitations of the compilation time of the standard FPGA tool flow.

ACKNOWLEDGMENT

This work was supported by the European Commission in the context of the H2020 FETHPC EXTRA project (#671653).

REFERENCES

[1] M. Hubner, P. Figuli, R. Girardey, D. Soudris, K. Siozios, and J. Becker, "A Heterogeneous Multicore System on Chip with Run-Time Reconfigurable Virtual FPGA Architecture," in *Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW), 2011 IEEE International Symposium on*, May 2011, pp. 143–149.

[2] K. Heyse, T. Davidson, E. Vansteenkiste, K. Bruneel, and D. Stroobandt, "Efficient implementation of virtual coarse grained reconfigurable arrays on FPGAs," in *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications*. Piscataway, NJ, USA: IEEE, 2013, pp. 1–8.

[3] J. Coole and G. Stitt, "Intermediate fabrics: Virtual architectures for circuit portability and fast placement and routing," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, Oct 2010, pp. 13–22.

[4] K. Heyse, B. Al Farisi, K. Bruneel, and D. Stroobandt, "TCONMAP: Technology Mapping for Parameterised FPGA Configurations," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 4, pp. 48:1–48:27, Sep. 2015. [Online]. Available: http://doi.acm.org/10.1145/2751558

[5] A. Kulkarni, K. Heyse, T. Davidson, and D. Stroobandt, "Performance evaluation of Dynamic Circuit Specialization on Xilinx FPGAs," in *FPGAworld Conference 2014, Proceedings*. Stockholm, Sweden: Association for Computing Machinery, 2014, pp. 1–6.

[6] A. Kulkarni, V. Kizheppatt, and D. Stroobandt, "MiCAP: A custom Reconfiguration Controller for Dynamic Circuit Specialization," in *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*, Dec 2015, pp. 1–6.

[7] A. Kulkarni and D. Stroobandt, "How to efficiently reconfigure Tunable LookUp Tables for Dynamic Circuit Specialization," *INTERNATIONAL JOURNAL OF RECONFIGURABLE COMPUTING*, vol. 2016, pp. 1–11, 2016.

[8] J. Divyasree, H. Rajashekar, and K. Varghese, "Dynamically reconfigurable regular expression matching architecture," in *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, July 2008, pp. 120–125.

[9] L. Sekanina, "Virtual Reconfigurable Circuits for Real-world Applications of Evolvable Hardware," in *Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware*, ser. ICES'03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 186–197. [Online]. Available: http://dl.acm.org/citation.cfm?id=1766731.1766753

[10] T. Miyoshi, H. Kawashima, Y. Terada, and T. Yoshinaga, "A Coarse Grain Reconfigurable Processor Architecture for Stream Processing Engine," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 490–495.

[11] E. Vansteenkiste, B. Al Farisi, K. Bruneel, and D. Stroobandt, "TPaR: Place and Route Tools for the Dynamic Reconfiguration of the FPGA's Interconnect Network," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 3, pp. 370–383, March 2014.

[12] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum, "Detection of blood vessels in retinal images using two-dimensional matched filters," *IEEE Transactions on medical imaging*, vol. 8, no. 3, pp. 263–269, 1989.

[13] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *Design Test of Computers, IEEE*, vol. 28, no. 4, pp. 18–27, July 2011.

[14] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-strength Verification Tool," in *Proceedings of the 22Nd International Conference on Computer Aided Verification*, ser. CAV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 24–40. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14295-6_5

[15] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.

[16] A. Kulkarni, T. Davidson, K. Heyse, and D. Stroobandt, "Improving reconfiguration speed for Dynamic Circuit Specialization using Placement Constraints," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, Dec 2014, pp. 1–6.