RTRLib: A High-Level Modeling Tool for the Implementation of Dynamically Partial Reconfigurable System-on-Chips

Regina Marcela Ivo
Department of Mechanical Engineering
University of Brasilia
Brasília, Brazil
Email: regina.ivo29@gmail.com

Daniel M. Muñoz
Department of Mechanical Engineering and
Electronics Engineering, Faculty of Gama
University of Brasilia
Brasília, Brazil
Email: damuz@unb.br

***Abstract*—Partial Reconfiguration allows the time-multiplexing resources to be explored, accomplishing requirements such as adaptability, robustness, power consumption, cost, and also fault tolerance. This paper presents a new partial reconfiguration toolkit called RTRLib that aims to simplify the development of dynamically reconfigurable systems, consequently reducing the development time. The RTRLib is a platform-based high-level modeling tool for run-time reconfigurable systems based on the semi-formal refinement-by-replacement methodology used to simulate and analyze the system behavior during the reconfiguration process. The RTRLib also automatically provides the Vivado and SDK scripts, under some restrictions. Since RTRLib is based on previous characterized IP-cores, it is possible to have an earlier estimation of the reconfiguration time and the reconfigurable partitions resource utilization during the design process. The proposed solution allows the designer to construct the partial reconfiguration solution to be mapped on FPGA-based System-on-Chips by a simple system specification and parametrization.**

***Index Terms*—Partial Reconfiguration; High-level Modeling; System-on-Chip; FPGA**

I. Introduction

Dynamic partial reconfiguration is a powerful technique applied to high-performance embedded systems, allowing requirements such as adaptability, robustness, fault tolerance, low power consumption, and low cost to be achieved, taking into account physical size restrictions [1].

The using of dynamically reconfigurable systems has some advantages, of which it is possible to stands out the increasing of the solution flexibility through the functionality of the time-multiplexing of the configurations. Also, the hardware resources and the consumption of dynamic power are decreased, reflecting on the reduction of the size and costs of the FPGA (Field Programmable Gate Array) [2]. When designing dynamically partial reconfigurable (DPR) systems, it is essential to analyze in advance the following aspects: (a) the impact of the reconfiguration time, since it can increase the total latency of the system, reducing its performance; (b) the area required for each configuration and; (c) the data dependency between reconfigurable regions and; (d) the reconfiguration strategies depends on the adoption of hardware or software triggers and involves the use of specific hardware responsible for controlling the reconfiguration process [3]. These considerations demonstrate the need of a clear vision about the system behavior during the development process.

A DPR system is composed of a static part and one or multiple *Reconfigurable Partitions* (RP), each RP can be loaded with different *Reconfigurable Modules* (RM), where each RM implements a specific functionality of the system. Then, the FPGA resources can be time-multiplexed by the dynamic reconfiguration of the FPGA configuration memory with various hardware functionalities, defined by different design specifications. The Vivado IDE allows the development of DPR systems by generating the bitstream of the static logic as well as partial bitstreams (PBS) related to each RM.

The complexity of developing a DPR system is considerably higher when compared to the one of a static system. From this perspective, it is possible to see the difficulties in designing DPR systems, such as the hardness of exploring the effects of some RMs characteristics, for instance, the impact of the reconfiguration time and latency on the performance and behavior of the overall system.

In this paper, the authors introduce the RTRLib, a new platform-based high-level modeling tool that intends an easy generation of DPR systems, specifically focusing on all the details concerned with the RTRLib *high-level modeling framework*. An example of proof of concept consisting of two RPs (the first one with 3 RMs and the second one with 2 RMs) connected to the ARM processor of a Zynq device was used for validating the proposed tool. The results demonstrate the correctness of the extraction of the design specification as well as the effectiveness of the final implementation, including the partial bitstreams obtained from the created TCL scripts.

The rest of the paper is organized as follows: Next Section presents a general description of the RTRLib. Section III compares the RTRLib with previous works in the literature, Section IV presents the *high-level modeling framework* and, before concluding, Section V presents an example and results.

CAPES. CNPq. CISB. Saab AB.

Authorized licensed use limited to: University of Edinburgh. Downloaded on June 14,2020 at 14:37:53 UTC from IEEE Xplore. Restrictions apply.

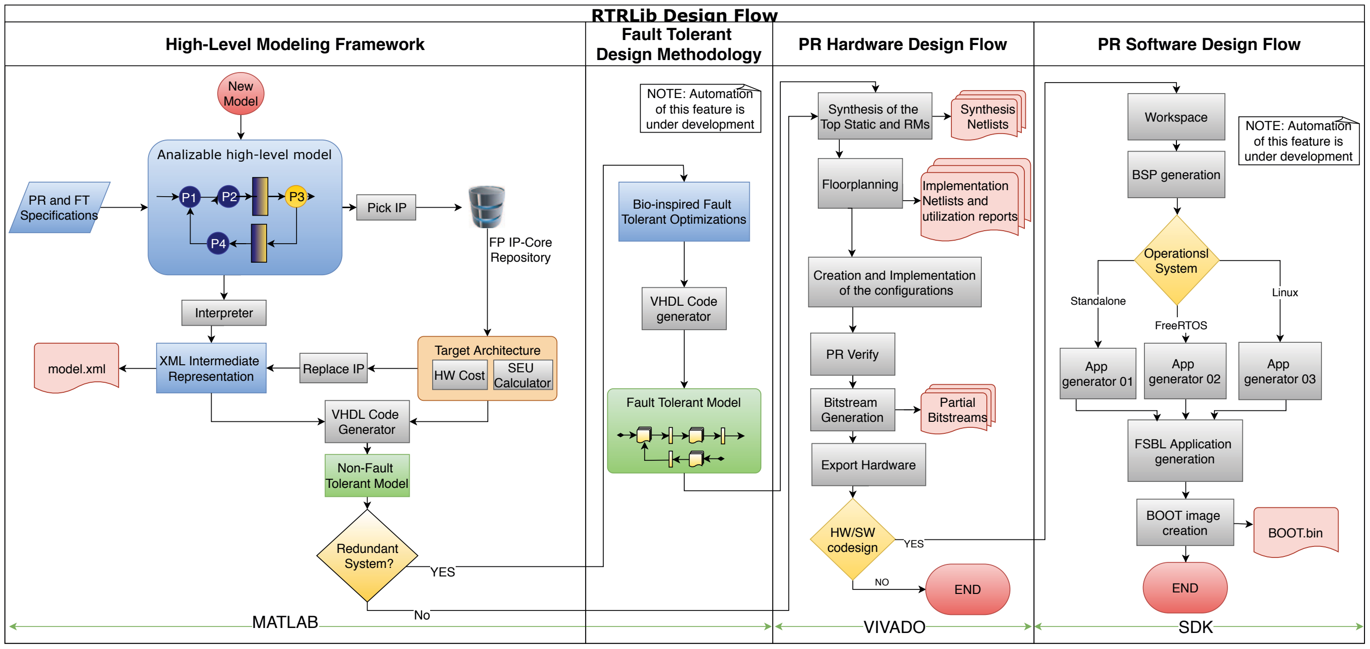

Fig. 1. Overview of the RTRlib proposed design flow. The blue rectangles represent high-level models. The green squares represent the version of the fault-tolerant models. The tasks related to the design transformation and characterization, such as the automatic VHDL code generation, are identified by the gray boxes. The generated files are represented in magenta, while the decision points are shown in yellow.

II. RTRLIB - GENERAL DESCRIPTION

RTRLib is a high-level modeling tool that automatically provides the Vivado and SDK TCL scripts for developing DPR systems on Xilinx Zynq-7000 devices, allowing the estimation of the resource utilization, latency, and reconfiguration time of each RM.

These features are possible because the RTRLib is a platform-based design tool that uses the refinement-by-replacement methodology [4], allowing non-experts users to make use of functional blocks, interconnecting them as a process network to express a specific behavior of the system.

The proposed RTRLib design flow can be divided into four different approaches, mapped in the following lanes: *high-level modeling framework, hardware redundancy design methodology, PR hardware design flow*, and *PR software design flow*. Figure 1 illustrates the overall RTRLib design flow.

The first step of the design flow using the RTRLib is the *high-level modeling framework* description, Section IV describes in detail this part of the system.

In the context of the hardware redundancy design, the RTRLib aims to implement a fault-tolerant design methodology based on the structural design of an NMR solution [5]. The designer provides the mission details, such as the FPGA family and mission time of each module. From this data, the resources estimation, the size in MB of each RP, and the system reliability are estimated and provided. Two bio-inspired methodologies were applied (mono-objective and multi-objective) where the primary goal is to find the optimum redundancy level for each stage, that is, to discover the optimal number of replicas of each module in each step.

Taking the model constructed from the description, for both cases redundant and non-redundant, the RTRLib executes the PR hardware design flow of which the main result is a script with the TCL commands to generate the system desired.

From the point of view of the *PR software design flow*, the Zynq platform enables the utilization of three different embedded OS, the C/C++ standalone, the FreeRTOS823 Xilinx, and C/C++ Linux. At the configuration of the RTRLib ARM block, the user can select from any of them. In the specific case of the C/C++ standalone solution, the RTRLib creates a template that allows the reconfiguration through software triggers sent through the UART. In the case of the FreeRTOS solutions, the RTRLib provides a visual and simple interface to construct personalized applications.

III. STATE OF THE ART

A few academic tools have been developed for automating the entire design flow and overcome the difficulties in designing DPR systems. When designing DPR systems, several decisions must be adopted to guarantee the correctness of the implementations. The interface between the static logic and the reconfigurable regions, the routing isolation and decoupling mechanisms [6], [7], the reconfiguration strategy, and the communication between RPs are some of those decisions.

Table I summarizes the features provided by the works in the state-of-the-art compared with RTRLib. Other relevant approaches are presented in [8], [9] [10], [11], and [17].

Koch et al. [12] introduce its I/O bar communication technique and the RecoBus-builder tool to automatically generate

Authorized licensed use limited to: University of Edinburgh. Downloaded on June 14,2020 at 14:37:53 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISION OF THE MAIN PR DESIGN TOOLS

Feature	**Recobus Builder [12]**	**GoAhead [6]**	**RePaBit [13]**	**IMPRESS [7]**	**FASTER [14]**	**CAOS [15]**	**RTRLib**
Interface	Various macros	Direct wire binding	Bus macro	Virtual Interface	Not reported	Not reported	Partition Pins
Decoupling	Yes	Yes	No	Yes	No	No	Yes
PR-PR Communication	No	Yes	No	Yes	No	No	No
GUI	Yes	Yes	No	No	Not reported	Yes	Yes
HDL generation for RMs	No	No	No	No	No	using HLS	Only structural design
High-level analysis	Yes	No	No	No	Yes	Yes	Yes
Fault-Tolerance	No	No	No	No	No	No	Only hardware redundancy
Application generation	No	No	No	No	Yes	Yes	Standalone,Linux,FreeRTOS
IDE	ISE (XDL)	ISE (XDL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)	VIVADO(TCL)
Supported devices	Virtex II/Pro and Spartan 3	Virtex and Spartan 6	Zynq Soc	Zynq Soc	Virtex 5 and Zynq	Xilinx VU9P	Zynq Soc

the communication infrastructure in PR designs. The GoAhead tool [6] implements reconfigurable interfaces without introducing resource overheads and also provides static/partial decoupling, hierarchical reconfiguration, and reconfigurable crossing region.

The RepaBit tool [13] makes use of bus macro interfaces and adopts the isolation design flow (IDF) for producing relocatable partial bitstreams for Xilinx Zynq devices. The IMPRESS tool [7] develop a custom *virtual interface* based on fixed nodes shared between the static logic and the RPs, supports relocatable bitstreams by avoiding *feed-through* nets, and permits communication between RPs through the use of the AXI interface.

The FASTER tool [14] is an integrated semi-automatic framework that allows the development, from a C-based description, of reconfigurable heterogeneous MPSoC systems. The CAOS platform [15] is designed to encourage the community to adopt, develop, and improve HPRC (High-Performance Reconfigurable Computers) systems. It comprehends the full process of application optimization, from the identification and optimization of the kernel functions to the generation of the runtime management for the target system. For both tools, the authors do not mention in detail all the target devices, however it is mentioned study cases using Virtex 5 and Zynq devices, in [16], and Xilinx VU9P devices, in [14] respectively.

This paper introduces RTRLib focusing on the description of the RTRLib *high-level modeling framework*, as described in the next Section.

IV. HIGH-LEVEL MODELING FRAMEWORK

The RTRLib has been developed in MATLAB and Simulink. Some essential aspects of the RTRLib modeling process are discussed below.

Initially, the user must instantiate and configure the *Top Module Block*, selecting the target board (Zedboard or Zybo) and the reconfiguration strategy between the three options:

1) *PRC (Partial Reconfigurable Controller) + ICAP:* This strategy uses the PRC IP. When hardware trigger events occur, the PRC pulls partial bitstreams from memory and delivers them to the ICAP. The PRC is configured according to the number and names of the RPs, number and names of the RMs, and also the address and size of each RM.
2) *PCAP + ARM:* This strategy allows only software triggers events. In this case, the ARM controls the reconfiguration process through the PCAP.
3) *Manual:* By this strategy, using the Vivado Hardware Manager Tool through the JTAG connection, the user can manually configure each RP by downloading each partial bitstream.

Then, the *Reconfigurable Partitions Blocks* must be inserted inside the *Top Module Block*. Each RP is modeled as a variant system, allowing the high-level model to use global variables for simulating a reconfiguration process by selecting one of the available subsystems. At this step, the designer must define if each RP will be connected through the AXI protocol or not. In the case of the AXI protocol, the tool automatically generates the HDL code of the AXI-Lite communication interface connected to the RP. On the other hand, the RPs that are not connected to the ARM are configured to reset after the reconfiguration process.

In a deeper level of abstraction, the designer inserts the *Reconfigurable Module Blocks* into each RP block, as many RM there are in each RP. At this level, the blocks do not have to be connected. During the simulation step, the connectivity is automatically determined based on the name of the active variant system.

The RTRLib repository is composed of arithmetic operators, FIR filters, and neuron networks, among others. As the system is modeled based on these pre-characterized IP-Cores, it is possible to have an early estimation about some essential features in the DPR design, such as the utilization resources in terms of LUTs, FFs, DSPs, and BRAMs for the target FPGA, the latency in clock cycles, and to estimate the size of each RP and its reconfiguration time. In the case of hardware redundancy designs, failure rates, and reliability of each RM estimations according to measurements, according to the target device, are also provided. This description allows the users to do preliminary tests and simulations with their architecture even in the high-level phase of the design, reducing the time of extensive behavioral simulation, taking in mind that a lot of errors can be detected in this design step.

In the case of SoC designs, the *ARM_Core block* and the *AXI-Interconnect Blocks* must be included. The parameters configurations for the *ARM_Core block* are based on the usage of the SD-Card, the UART, the XADC, and the DDR3 Controller as well as on the definition of the application

Authorized licensed use limited to: University of Edinburgh. Downloaded on June 14,2020 at 14:37:53 UTC from IEEE Xplore. Restrictions apply.

type that will be embedded in the ARM Core (standalone, FreeRTOS or Linux Application).

The interpreter creates an intermediate representation in the XML format that contains the description of the tree diagram of the system, which includes information on the quantity of RPs, number of RMs in each RP, configuration parameters of each block, and connections.

The floorplanning is done through the GUI. The pblocks, where each RP will be placed, must be manually defined, taking into account the resource requirements of each RM. Several DRC rules must be accomplished to ensure the effectiveness of the placement and routing. RTRLib is equipped with some error and warning messages that early prevent the user from making the most common mistakes.

After high-level modeling, RTRLib makes use of the Vivado PR design flow. It executes from the creation of the folder structure and the repository with the available IP-Cores to the hardware exportation. At the end, the script verifies if the system is an HW/SW codesign. In the case that the project is a hardware-only design, it finishes the flow. However, if the project is an HW/SW codesign, the script launches the SDK and initiates the *PR software design flow*.

V. PROOF OF CONCEPT USING RTRLIB

As a proof of concept of the proposed tool, a dynamically reconfigurable architecture with 2 RPs is provided as an example of the utilization of the RTRLib. For the first RP, there are 3 RMs that implement the arithmetic calculation of addition, division, and multiplication by constant. For the second RP, there are 2 RMs that implement the sine and the computation of the exponential function of a given value, where all the five operations are done using the floating-point IP-Cores. For this example, it was selected a non-fault-tolerant system, the strategy of reconfiguration using the PRC, and both RPs connected to the ARM through the AXI-Lite Interconnect.

The ARM processor sends the RPs input data through the AXI-Lite. In the opposite direction, the ARM reads the result of the operations through the AXI. The reconfiguration is done through the ICAP by hardware triggers mapped in the switches of the board and controlled by the PRC. Figure 2 shows the high-level modeling of the system, with the connections between the RPs and the ARM processor. Table II shows the utilization report of all the reconfigurable modules.

TABLE II
RMs UTILIZATION REPORT FOR THE ZEDBOARD (ZYNQ 7020). THE RP_ODD CONTAINS 188 SLICES AND 4 DSPs, WHILE THE RP_EVEN CONTAINS 438 SLICES, 3 BRAMs AND 6 DSPs.

Name RM	Function	LUTs		FFs		DSPs	
		Available:	53200	Available:	106400	Available:	220
RM_t1	ADD	334	0,63%	275	0,26%	0	0%
RM_t2	EXP	1785	3,36%	596	0,56%	1	0.45%
RM_t3	DIV	278	0,52%	320	0,30%	1	0.45%
RM_t4	SINE	1419	2,67%	557	0,52%	1	0.45%
RM_t5	MUL	135	0,25%	255	0,24%	1	0.45%

Figure 3 (A) shows the block design created through the script generated using RTRLib, which has the IPs related to

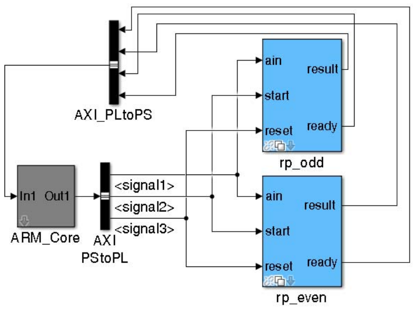

Fig. 2. Detail of the system-level model using the RTRLib, showing the RPs, AXI, and the connections. The data input of the RPs is sent to the ARM processor through the UART port (not depicted in the figure).

the RPs, the ARM-Core processor IP, PRC, and the AXI Interconnect.

The size in kB of each partial bitstream was obtained and using the model reported by [18], which also uses Zedboard and states that the relationship between the reconfiguration time and the size of the partial bitstream is essentially linear, the reconfiguration time was estimated. Those results are presented in Table III. Figure 3(B) shows the circuit layout containing the floating-point adder in the first RP and the floating-point sine estimator in the second RP.

TABLE III
COMPARISION BETWEEN THE SIZE OF THE PARTIAL BITSTREAMS AND THE RECONFIGURATION TIME

Partial bitstream	Size (KB)	Estimated Reconfiguration Time (ms)
Blanking	3951	30,422
RP odd	223	1,717
RP even	484	3,727

VI. CONCLUSIONS AND FUTURE WORK

This paper introduced the RTRLib, a new platform-based high-level tool that was projected to automate the partial reconfiguration design flow steps. The main contributions of the RTRLib are: (a) DPR system design is more accessible to non-expert designers; (b) the RTRLib allows for an early estimation of the hardware resources and size of each reconfigurable module, thus enabling an analysis of the effect of the latency and reconfiguration time of each RP over the system; (c) The proposed tool also provides a design methodology for fault-tolerant systems based on the hardware redundancy approach. All these features allow the prototyping time reducing and enable fast creation of high-performance systems.

A simple proof of concept was modeled in RTRLib, in which two reconfigurable partitions implement floating-point arithmetic and trigonometric operations. The obtained scripts were effectively executed in Vivado, and the circuits were characterized in terms of resource consumption and size (in KB) of the RPs and reconfiguration time.

As future works, it is crucial to investigate strategies to enhance the RTRLib tool floorplanning algorithm to enable

Authorized licensed use limited to: University of Edinburgh. Downloaded on June 14,2020 at 14:37:53 UTC from IEEE Xplore. Restrictions apply.

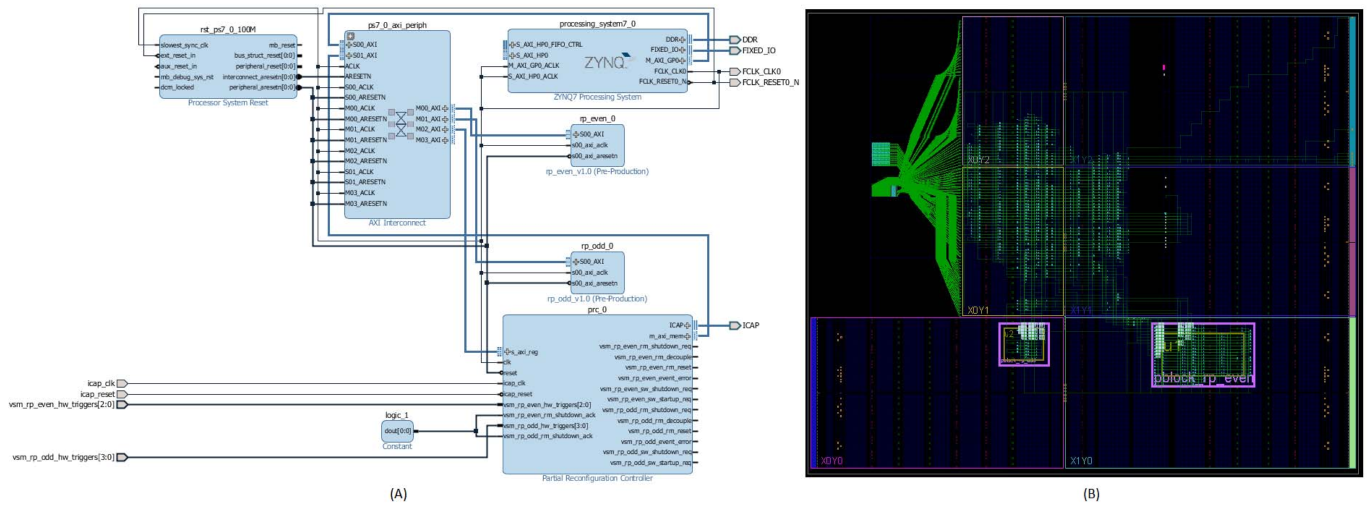

Fig. 3. (A) Block Based Architecture with two RPs, PRC, AXI switch and Zynq7 Processing System (B) Final implementation of the reconfigurable architecture, including the static system and the reconfigurable partitions, with the reconfigurable modules that compute the adder and the calculus of the sine function, respectively. Implemented on a Zynq-7020 device.

slot and grid styles, analyze data dependency between RPs, integrating models for reconfiguration time and power consumption estimation, and implementing optimization algorithms to minimize the reconfigurable resources into the RPs.

Acknowledgment

Authors thank to Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 and the financial support of CNPq, National Council of Scientific and Technological Development of Brazil, CISB, Swedish-Brazilian Research and Innovation Centre, and of Saab AB.

References

[1] M. Wolf, *High-performance embedded computing: applications in cyber-physical systems and mobile computing*. Newnes, 2014.

[2] X. Zhang and K. W. Ng, "A review of high-level synthesis for dynamically reconfigurable FPGAs," *Microprocessors and Microsystems*, vol. 24, no. 4, pp. 199–211, 2000.

[3] J. Zhu, I. Sander, and A. Jantsch, "Performance analysis of reconfigurations in adaptive real-time streaming applications," *ACM Transactions on Embedded Computing Systems*, vol. 11S, no. 1, pp. 12:1–12:20, Jun. 2012. [Online]. Available: http://doi.acm.org/10.1145/2180887.2180888

[4] S. H. A. Niaki and I. Sander, "Co-simulation of embedded systems in a heterogeneous MoC-based modeling framework," in *IEEE Int. Symposium on Industrial and Embedded Systems*, June 2011, pp. 238–247.

[5] E. Dubrova, *Fault-tolerant design*. Springer, 2013.

[6] C. Beckhoff, D. Koch, and J. Torresen, "Go ahead: A partial reconfiguration framework," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 37–44.

[7] R. Zamacola, A. G. Martínez, J. Mora, A. Otero, and E. de La Torre, "Impress: Automated tool for the implementation of highly flexible partial reconfigurable systems with Xilinx Vivado," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2018, pp. 1–8.

[8] A. A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood, "Openpr: An open-source partial-reconfiguration toolkit for xilinx fpgas," in *2011 IEEE International Parallel & Distributed Processing Symposium Workshops and PhD Forum*. IEEE, 2011, pp. 228–235.

[9] R. Oomen, T. Nguyen, A. Kumar, and H. Corporaal, "An automated technique to generate relocatable partial bitstreams for Xilinx FPGAs," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2015, pp. 1–4.

[10] K. Bruneel and D. Stroobandt, "Automatic generation of run-time parameterizable configurations," in *2008 International Conference on Field Programmable Logic and Applications*. IEEE, 2008, pp. 361–366.

[11] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapidsmith: Do-it-yourself cad tools for xilinx FPGAs," in *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 349–355.

[12] D. Koch, C. Beckhoff, and J. Teich, "Recobus-builder—a novel tool and technique to build statically and dynamically reconfigurable systems for FPGAs," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*. IEEE, 2008, pp. 119–124.

[13] J. Rettkowski, K. Friesen, and D. Göhringer, "Repabit: Automated generation of relocatable partial bitstreams for Xilinx Zynq fpgas," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2016, pp. 1–8.

[14] D. Pnevmatikatos, K. Papadimitriou, T. Becker, P. Böhm, A. Brokalakis, K. Bruneel, C. Ciobanu, T. Davidson, G. Gaydadjiev, K. Heyse *et al.*, "Faster: facilitating analysis and synthesis technologies for effective reconfiguration," *Microprocessors and Microsystems*, vol. 39, no. 4-5, pp. 321–338, 2015.

[15] M. Rabozzi, R. Brondolin, G. Natale, E. Del Sozzo, M. Huebner, A. Brokalakis, C. Ciobanu, D. Stroobandt, and M. D. Santambrogio, "A cad open platform for high performance reconfigurable systems in the extra project," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 368–373.

[16] M. Rabozzi, "Caos: Cad as an adaptive open-platform service for high performance reconfigurable systems," in *Special Topics in Information Technology*. Springer, 2020, pp. 103–115.

[17] F. Cancare, M. D. Santambrogio, and D. Sciuto, "A design flow tailored for self dynamic reconfigurable architecture," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.

[18] E. Fazzoletto, "Characterization of partial and run-time reconfigurable FPGAs." KTH Royal Institute of Technology, 2016, Technical report.

Authorized licensed use limited to: University of Edinburgh. Downloaded on June 14,2020 at 14:37:53 UTC from IEEE Xplore. Restrictions apply.