

An Integrated Development Toolset and Implementation Methodology for Partially Reconfigurable System-on-Chips

Abelardo Jara-Berrocal and Ann Gordon-Ross

NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611
{berrocal, ann}@chrec.org

Abstract—Partial reconfiguration (PR) enhances traditional FPGA-based system-on-chips (SoCs) by providing additional benefits such as reduced area and increased functionality as compared to non-PR SoCs. However, since leveraging these additional benefits requires specific designer expertise and increased development time, PR has not yet gained widespread usage. In this paper, we present an integrated development toolset that automates the implementation of PR SoCs on FPGA devices and leverage this tool in a rapid design space exploration case study.

Keywords—reconfigurable computing; partial reconfiguration; system-on-chip.

I. INTRODUCTION

To meet designer demands, field programmable gate array (FPGA)-based system-on-chips (SoCs) incorporate hardcoded components, such as microprocessors, memory, multipliers, Ethernet cores, and high-speed telecommunication transceivers, in the FPGA fabric, which execute alongside designer-architected hardware modules. Partial reconfiguration (PR) enhances the fabric's reconfigurability, enabling uninterrupted time-multiplexing of the fabric's resources during runtime, which provides reduced power, area, and cost and increased runtime flexibility as compared to non-PR SoCs. The combination of hardcoded components and fabric reconfiguration enables complex SoC designs, however area, power, and performance design constraints make SoC development challenging.

Development tools provide designers with automated SoC development assistance, affording improved productivity and adherence to design constraints, however, few tools/methods leverage PR, leaving designers with a largely manual and time-consuming, tedious development process. Designers must create the PR architecture (PR floorplan), which partitions the FPGA fabric into the static region and one or more PR regions (PRRs) and defines the PRRs' physical locations and dimensions. This PR architecture must also support the application's required inter-module, module-to-component, and module-to-static region communication. Additionally, in order to fully exploit PR's benefits, designers must consider the PR architecture during application development and partition the application's functionality between the static region and one or more PR modules (PRMs), which execute in the PRRs. Since PR applications require hardware modules to be developed in hardware description languages (HDLs), such

as VHDL or Verilog, PR application development requires increased development time and is more error prone as compared to non-PR application development, which typically leverages high-level languages, such as C or C-like languages.

In this paper, we introduce the VAPRES SoC builder (VSB), a toolset that assists PR FPGA SoC architecture and application development for the VAPRES (Virtual Architecture for Partially Reconfigurable Embedded System) [8] base template architecture. The VSB provides an integrated development environment (IDE) for application development, which allows designers to use C/C++ code for the application software and either HDL or Impulse C code [14] for the hardware modules. The VSB automatically compiles the application software, performs PR floorplanning, and generates the application's hardware modules' partial bitstreams (for execution inside the PRRs) and the system's static bitstream for the static region. We quantify the effects of different VAPRES architectural layouts, which can assist in rapid design space exploration.

II. RELATED WORK

Much previous work focused on designing scalable and flexible SoC architectures that provide SoC designers with base template systems for building PR FPGA SoCs and introduced novel architectural features, such as streaming inter-module communication [8][9][11], dynamic clock adaptation [8], and compatibility with partial bitstream relocation [4]. However, to the best of our knowledge, no previous work on PR FPGA architectures reported implementation of a toolset for automatically building the PR FPGA architecture from SoC designer specifications. One previous work introduced the RecoBus builder tool [9] for

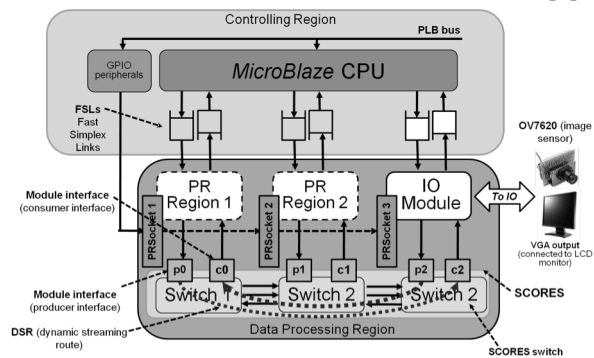


Figure 1: Sample VAPRES architectural layout showing two partially reconfigurable regions (PRRs) and one I/O module (IOM).

automated implementation of RecoBus PR systems. However, the RecoBus builder tool did not automate the implementation of a complete PR FPGA SoC and only provided a Virtex-4-specific hard-macro inter-module communication architecture.

To assist application designers in implementing applications for PR FPGAs, previous work proposed using high level synthesis (HLS) for application modeling. Lee et al. [10] proposed a high-level synthesis framework for PR application development using a modified form of C (RT-C). Craven et al. [3] introduced an HLS framework for PR application development using Impulse C code. Abel et al. [1] proposed an object oriented programming (OOP)-based HLS framework for PR application development. Mitra et al. [12] proposed leveraging the Riverside Optimizing Compiler for Configurable Computing (ROCCC) to develop hardware modules for PR FPGA SoCs. However, each of these previous works did not provide a flexible inter-module communication, which hindered the application’s functionality and/or performance.

III. VAPRES ARCHITECTURE AND APPLICATIONS

The VSB leverages the VAPRES architecture, a multipurpose PR FPGA SoC (we refer the reader to [7][8] for further details). Figure 1 depicts a sample VAPRES architectural layout with two PRRs and one input/output module (IOM). VAPRES’s controlling region contains a soft-core MicroBlaze microprocessor in the static region and a set of static peripherals, and is responsible for controlling the data processing (via *PRSockets*), performing system level functions (such as reconfiguring the PRRs via the internal configuration access port (ICAP)), and executing software modules. VAPRES’s data processing region contains the PRRs, IOMs, and SCORES – a scalable communication architecture for reconfigurable embedded systems [7]. PRRs and IOMs communicate using SCORES and IOMs directly interface to external I/O pins or peripherals. IOMs and PRRs interface with the MicroBlaze through asynchronous FSL (fast simplex link) interfaces and with SCORES using dynamically established data streaming routes (DSRs) over producer and consumer module interfaces. For each switch-PRR or switch-IOM pair in SCORES, one *PRSocket* allows the MicroBlaze microprocessor to control the hardware module, IOM, and module interface operation.

VAPRES applications typically match the structure of a reconfigurable stream processing system (RSPS) [6]. RSPSs are composed of a set of hardware and software modules connected together to transform a data input stream into a processed data output stream. Since the required data stream transformations may be dependent on stream characteristics, application requirements, or available resources, VAPRES provides a method to change the processing modules without interrupting data stream processing. To provide synchronization mechanisms, VAPRES enables RSPS runtime assembly, a technique that places RSPS hardware modules inside PRRs at runtime and uses SCORES’s channels to dynamically establish the required inter-module communication.

Impulse C provides an ideal framework to develop RSPSs for VAPRES. The Impulse C programming model derives

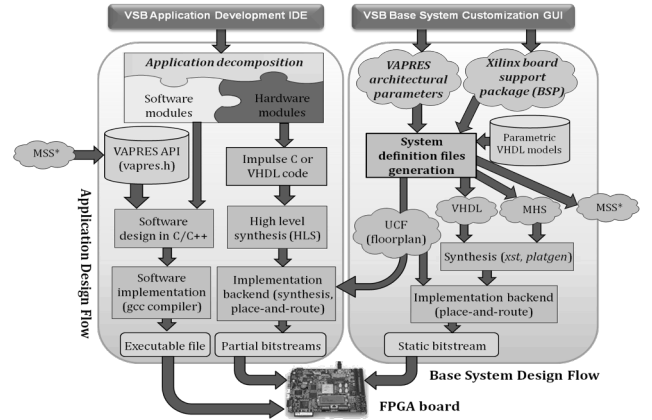


Figure 2: Base system and application design flows

from the concurrent sequential process (CSP) paradigm introduced by Hoare [5]. An Impulse C application consists of a set of intercommunicating hardware and software processes, which map to the software modules running on the MicroBlaze and the hardware modules executing inside the PRRs, respectively. SCORES transparently integrates Impulse C-generated hardware modules into VAPRES because the SCORES module interfaces (both producer and consumer) are fully compatible with the Impulse C streaming interfaces.

IV. BASE SYSTEM AND APPLICATION IMPLEMENTATION METHODOLOGIES

The VSB automates PR FPGA SoC and application implementation using a base system customization graphical user interface (GUI), an application development IDE, and an implementation backend to seamlessly interface with the Xilinx tools (Figure 2). Creating FPGA PR SoC architectures and applications using the VSB requires two design flows: (1) the *base system design flow* assists SoC designers in creating a VAPRES base system (Figure 2 right), and (2) the *application design flow* assists application designers in creating applications to run on the VAPRES base system (Figure 2 left). In this section, we describe the architecture of the VSB implementation backend and both design flows.

A. Implementation Backend

The VSB implementation backend performs synthesis, place-and-route, and bitstream generation for the FPGA PR SoC static region and each of the applications’ hardware modules. We leverage Python’s PyXilTCL to map Xilinx ISE, EDK, and PlanAhead TCL shell commands to Python functions and objects. This method allows the implementation backend to transparently open, modify, and execute ISE, EDK, and PlanAhead projects using both synchronous and asynchronous interfaces. Synchronous interfaces enable the implementation backend to perform serialized operations (e.g., synthesis, place-and-route, and bitstream generation must be performed in this order), while asynchronous interfaces enable the implementation backend to perform parallel operations (e.g., the implementation backend can simultaneously launch place-and-route on all hardware modules).

B. Base System Design Flow

Figure 2 (right) depicts the VSB base system design flow. SoC designers specify the board support package (BSP) [15], which is provided by Xilinx or third-party vendors and contain

all of the platform-specific information required to produce an FPGA SoC design on a specific FPGA development board. Next, the SoC designer customizes the VAPRES architectural parameter values, such as the SCORES parameters, the number of modules (PRRs and IOMs), and PRRs’ width and height, which are specified as an integer number of adjacent CLBs and clock regions, respectively.

After an SoC designer completes the base system customization, the VSB generates the *system definition files* from the VAPRES architectural parameters, the BSP, and the parametric VHDL models for SCORES. System definition files include the VHDL code modeling the static region, a *microprocessor hardware specification* (MHS) file defining the system structure for the Xilinx EDK tool platgen, a *microprocessor software specification* (MSS) file defining the base system build process for the Xilinx EDK tool libgen, and a *user constraints file* (UCF) representing the system floorplan. In order to match the VAPRES PRRs with the Xilinx FPGA clock regions, the VSB generates VAPRES floorplans by stacking equally sized homogeneous PRRs vertically (Figure 3).

While the VSB generates the MHS, MSS, and UCF files using the Xilinx EDK bindings using the PyXilTCL API (the BSP provides the Xilinx EDK with the required information to configure MicroBlaze peripherals in addition to defining the PR FPGA SoC I/O pins), the VSB generates the VHDL file modeling the static region from a YAML (Yet Another Markup Language)-based [17] VHDL template. After generating the system definition files, the VSB automatically synthesizes and implements the PR FPGA SoC by providing these system definition files to the implementation backend. The implementation backend interfaces with PyXilTCL and implements the static region.

C. Application Design Flow

The VSB includes an IDE for developing VAPRES applications, which enables application designers to write both software and hardware module code. Figure 2 (left) depicts the application design flow. Since the VSB is not an automated design space exploration tool for hardware/software partitioning, the application designer must manually decompose the application into the software and hardware modules. After application partitioning, the software modules follow the *software module design flow* and the hardware modules follow the *hardware module design flow*.

In the software module design flow, the application designer develops the application software for the MicroBlaze. In order to assist the application designer in writing VAPRES software modules, VAPRES API functions provide low-level system functionality, such as functions for reconfiguring PRRs with partial bitstreams stored as an array in external SDRAM memory and functions to establish a DSR between PRRs.

In the hardware module design flow, the application designer develops the hardware modules in VHDL or Impulse C. A hardware module’s input and output port *type* can be an FSL slave (reads data from an FSL), an FSL master (writes data to an FSL), a consumer port (reads data from a consumer interface), or a producer port (writes data to a producer interface).

Using Impulse C offers the advantage of isolating application designers from low-level hardware details. After the base system generation, the VSB IDE provides application designers with automatically generated hardware process templates in Impulse C. There is one Impulse C process template for each hardware module version, as PRRs can contain multiple versions of hardware modules at different execution times. The application designer modifies these Impulse C templates to model the desired functionality of the application’s hardware modules. After the application designer modifies the Impulse C templates, the VSB invokes the Impulse C compiler to generate RTL VHDL from the Impulse C templates.

The implementation backend performs synthesis (from either manually written or Impulse C-generated VHDL code), place-and-route, and partial bitstream generation for each hardware module version. Since the static region remains the same, the required time for implementing each version of the hardware module is significantly smaller than the implementation time for the base system, thus reducing application design time.

V. RESULTS

In order to verify the VSB’s flexibility and functionality and to evaluate the VSB’s generated systems, we use the VSB for design space exploration considering several design metrics for different VAPRES-based PR FPGA SoCs by varying VAPRES’s architectural parameters.

A. VSB Verification and Evaluation

We performed functional verification for several VSB-generated VAPRES base systems on a Xilinx ML401 board (XC4VLX25 FPGA) test device for five design metrics: area utilization, maximum clock frequency, partial bitstream size, and ICAP reconfiguration time (the time to perform PR using the ICAP). We measured ICAP reconfiguration time with software running on the MicroBlaze using the Xilinx MicroBlaze *xps_timer* peripheral.

To gather design metric results, we reconfigured each VAPRES PRR with one 512-bit counter (actual system function has little effect on the design metric results). The 512-bit counter consumed nearly all of the hardware resources inside a minimally sized PRR (the smallest PRR size available is 640 slices and the 512-bit counter required 592 slices). A minimally sized PRR is one clock region high and requires one fourth of the FPGA’s width (the minimum PRR width is less than half of the FPGA width as the non-reconfigurable clocking resources vertically span the middle of the fabric).



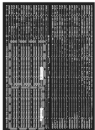

	Design 1	Design 2	Design 3	Design 4
Number of PRRs	1	1	2	3
PRR height	1 row (16 CLBs)	2 rows (32 CLBs)	2 rows (32 CLBs)	1 row (16 CLBs)
PRR width	10 CLBs	10 CLBs	10 CLBs	10 CLBs
SCORES parameters	N/A	N/A	N=1, kr=1,kl=1	N=2, kr=1,kl=1
Post-place and route implementation for base static system				

Figure 3: Sample VAPRES floorplans for different architectural parameters. Dotted boxes indicate placement and sizing for PRRs.

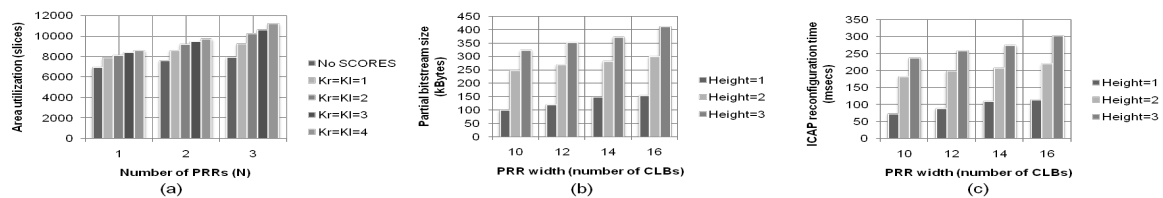


Figure 4: VAPRES profiles for different design metrics: (a) area utilization, (c) partial bitstream size, and (c) ICAP reconfiguration time.

Figure 4 (a) depicts area utilization for the 512-bit counter versus varying number of PRRs (N) for different VAPRES configurations. We evaluate VAPRES configurations both with and without SCORES enabled since not all PR SoCs require inter-module communication capabilities. Without SCORES enabled, the average area utilization, which consisted of the *PRSocket* and FSL digital circuitry connected to each PRR, increased by 278.2 slices as N increased. With SCORES enabled and $Kr = Kl = 1$ (minimum sized SCORES configuration), the average area utilization increased by 646.7 slices as N increased. This larger area utilization increase as compared to a VAPRES system without SCORES enabled was due to the addition of the SCORES switch with each PRR. With SCORES enabled and increasing Kr and Kl from 1 to 4 in increments of 1 and maintaining $Kr = Kl$, the average area utilization increased by 302.4 slices as N increased. Both increased area requirements when increasing the number of PRRs (278.2 slices) and the number of SCORES switches (646.7 slices) represents only a nominal portion of the total FPGA area (2.58% and 6.01% of a XC4VLX25 FPGA, respectively).

Figure 4 (b) and (c) depict the effects of varying PRR width in number of CLBs and height in number of clock regions on the partial bitstream size and the ICAP reconfiguration time, respectively, for the 512-bit counter. The ICAP reconfiguration time showed a linear relationship with respect to the partial bitstream size because the partial bitstream size and the ICAP reconfiguration time remained proportionally constant over all of the experiments (the measured ICAP bandwidth is roughly the partial bitstream size divided by the ICAP reconfiguration time). Our results also showed that increasing the PRR height by one extra clock region (16 CLBs) increased the partial bitstream size (and consequently the ICAP reconfiguration time) by 13.5% on average. By default, the Xilinx PR design tools generate partial bitstreams using a compressed format (*bitgen -g*) such that the configuration frames targeting CLBs in the same clock region can share the same FAR (frame address register). An increase in PRR height causes the place-and-route tools to disperse CLBs into a larger number of clock regions, therefore the generated partial bitstreams (after place-and-route) utilize a higher number of FARs and thus partial bitstream size increases.

VI. CONCLUSIONS

In this paper, we introduced the VAPRES system builder (VSB), an integrated toolset that assists SoC and application designers in implementing PR FPGA SoCs and the

applications that execute on these SoCs, respectively. Our VSB provides SoC designers with full customization of the VAPRES base system architecture and with automated SoC hardware implementation. In order to simplify the development of applications' hardware modules, our VSB permits application designers to model hardware modules in a high level language (Impulse C) in addition to a traditional hardware description language (VHDL). Future work includes exploring automated mapping of hardware processes and streaming channels in generic Impulse C applications to VAPRES PRRs and SCORES DSRs.

VII. ACKNOWLEDGMENTS

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. We gratefully acknowledge tools provided by Xilinx.

REFERENCES

- [1] N. Abel, F. Grull, N. Meier, A. Beyer, and U. Kerschull. Parallel hardware objects for dynamically partial reconfiguration. FPL, 2008.
- [2] C. Bobda, Introduction to Reconfigurable Computing: Springer-Verlag New York, LLC, 2007.
- [3] S. Craven and P. Athanas. High-level specification of runtime reconfigurable designs. ERSA, 2007.
- [4] A. Flynn, A. Gordon-Ross, and A.D. George. Bitstream relocation with local clock domains for partially reconfigurable FPGAs. DATE, 2009.
- [5] C. A. R. Hoare. Communicating sequential processes. Communications of the ACM, vol. 26, pp. 100-106, 1983.
- [6] A. Jara-Berrocal and A. Gordon-Ross. Runtime Temporal Partitioning Assembly to Reduce FPGA Reconfiguration Time. ReConFig, 2009.
- [7] A. Jara-Berrocal and A. Gordon-Ross. SCORES: A scalable and parametric streams-based communication architecture for modular reconfigurable systems. DATE, 2009.
- [8] A. Jara-Berrocal and A. Gordon-Ross. VAPRES: A virtual architecture for partially reconfigurable embedded systems. DATE, 2010.
- [9] D. Koch, C. Beckhoff, and J. Teich. ReCoBus-Builder - A novel tool and technique to build statically and dynamically reconfigurable systems for FPGAS. FPL, 2008.
- [10] T. K. Lee, A. Derbyshire, W. Luk, and P. Y. K. Cheung. High-level language extensions for run-time reconfigurable systems. FPT, 2003.
- [11] M. Majer, J. Teich, A. Ahmadi, C. Bobda. The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer. Journal of VLSI Signal Processing Systems, vol. 47, pp. 15-31, 2007.
- [12] A. Mitra, Z. Guo, A. Banerjee, and W. Najjar. Dynamic co-processor architecture for software acceleration on csocs. ICCD, 2006.
- [13] K. Paulsson, M. Hubner, and J. Becker. On-line optimization of FPGA power-dissipation by exploiting run-time adaption of communication primitives. SBCCI, 2006.
- [14] D. Pellerin and S. Thibault, Practical FPGA Programming in C. Upper Saddle River, N.J.: Prentice Hall, 2005.
- [15] Xilinx, Inc. EDK Concepts, Tools, and Techniques. UG683
- [16] Xilinx, Inc. Partial Reconfiguration User Guide. UG702 v12.3
- [17] YAML Yet Another Markup Language <http://www.yaml.org>