VAPRES: A Virtual Architecture for Partially Reconfigurable Embedded Systems

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 - **Due to the runtime flexibility offered by field programmable gate arrays (FPGAs), FPGAs are popular devices for stream processing systems, since many stream processing applications require runtime adaptability (i.e. throughput, data transformations, etc.). FPGAs can offer this adaptability through runtime assembly of stream processing systems that are decomposed into hardware modules. Runtime hardware module assembly consists of dynamic hardware module replacement and hardware module communication reconfiguration. In this paper, we architect a flexible base embedded system amenable to runtime assembly of stream processing systems using custom communication architecture with dynamic streaming channel establishment between hardware modules. We present a hardware module swapping methodology that replaces hardware modules without stream processing interruption. Finally, we formulate two design flows, system and application construction, to provide system and application designer assistance.**

I. INTRODUCTION

Field programmable gate arrays (FPGAs) have traditionally been used in reconfigurable computing to implement high performance custom data paths [2]. However, in addition to increasing FPGA fabric density (measured in number of slices), modern FPGAs include additional components, such as on-chip memories, hardware multipliers, and hardcoded microprocessors. These additional components provide the necessary building blocks to construct complex FPGA-based systems-on-chip (SoCs) using a single FPGA device, as opposed to traditional SoCs, which may require multiple interconnected application specific integrated circuits (ASICs).

Multipurpose FPGA SoCs alleviate many of the design challenges associated with application-specific FPGA SoCs and can meet design constraints for a wide range of applications. *System designers* architect generalized multipurpose FPGA SoCs with commonly used configurable peripherals, which serve as a base system for *application designers* to design and implement applications. In general, a multipurpose base system reduces application development time and increases reliability due to more thorough testing compared to application-specific FPGA SoCs.

Partial reconfiguration (PR) [12] is a feature of some modern FPGA devices, which isolates reconfiguration to specific partial reconfigurable regions (PRRs) without interrupting execution outside the reconfigured PRR. PR enhances FPGA SoCs by enabling dynamic *hardware module switching*, a technique that dynamically places hardware modules (hardware-based application functional units) in available PRRs on demand during runtime. Hardware module switching is an enabling technology in novel operating system frameworks [4], fault tolerance [5], and artificial intelligence systems [2]. Hardware module switching is particularly advantageous for reconfigurable streaming processing systems (RSPSs), which are composed of a set of hardware and software modules (software modules execute on an embedded microprocessor core) connected together to transform a data input stream into a processed data output stream. However, for hardware module switching to be most advantageous, the switching process must be quick and incur minimum stream processing interruption.

Despite PR advantages, PR significantly increases system design challenges. Current PR FPGA design tools (i.e. Xilinx Early Access PR flow [12]) are exceedingly complex, requiring system and application designers to have advanced knowledge of their target FPGA architecture, in addition to manually performing several time-consuming steps such as manually partitioning an application into the static region and one or more PRRs and creating the system floorplan by explicitly defining PRR physical locations and dimensions (size, height, and width).

In order to assist PR design for RSPSs, we present a Virtual Architecture for Partially Reconfigurable Embedded Systems (VAPRES). VAPRES is a multipurpose PR FPGA SoC composed of a soft-core Microblaze processor connected to a set of PRRs. VAPRES introduces several novel architectural features, such as the ability to operate hardware modules at independent and configurable clock frequencies. In addition, VAPRES includes hardware support for dynamic streaming channel establishment between arbitrary PRRs. To augment the VAPRES architectural features, we introduce a novel hardware module switching methodology that avoids stream processing interruption and a systematic design and implementation flow for building architecturally customized VAPRES base systems and applications. A key VAPRES feature is architectural customization, which enables optimizations to meet specific design constraints. Finally, we prototype and evaluate VAPRES on a Virtex-4 FPGA.

II. RELATED WORK

Conger et al. [3] formulated two methodologies to efficiently design and implement PR systems: a special-purpose and a multipurpose system design flow. The special-purpose system design flow targeted highly optimized PR

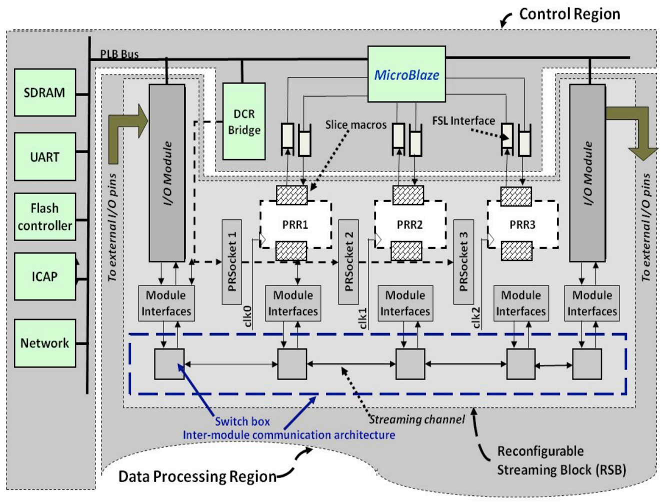

Figure 1: VAPRES architectural layout showing a single reconfigurable streaming block (RSB)

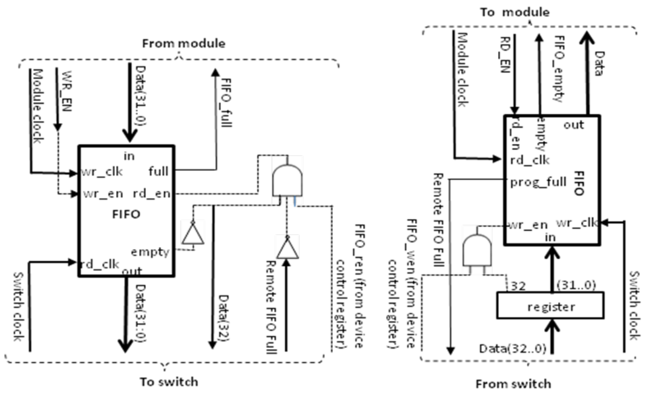

Figure 2: (a) Producer interface (b) Consumer interface

systems and required complete application specification and behavior knowledge during system design time. In contrast, the multipurpose system design flow targeted the design of PR base systems for implementing a wide range of applications.

In the area of multipurpose PR FPGA SoC design, Ullmann et al. [10] proposed a PR architecture for the Virtex-2 Pro. This architecture targeted automotive systems and included a Microblaze processor, an internal configuration access port (ICAP) controller [12], and four user-definable PRRs. The communication architecture required all communication between PRRs to be routed through the Microblaze.

Sedcole et al. [8] developed Sonic-on-a-Chip, an image processing PR architecture for the Virtex-2 Pro and Virtex-4. The architecture allowed dynamic streaming channel establishment directly between PRRs by allocating slots on a time-multiplexed bus. However, due to long routing delays, the reported bus clock frequency was 50 MHz.

Sudarsanam et al. [9] developed PolySAF, a PR architecture for reconfigurable processing based on systolic kernels for the Virtex-4. The architecture allowed communication between adjacent PRRs (PRRs placed next to each other in the floorplan) and between the Microblaze soft-core and PRRs via a multiplexed FIFO-based interface.

III. VAPRES Architecture and Operation

Figure 1 depicts the VAPRES architectural layout comprised of two fundamental regions: the *controlling region* and the *data processing region*.

A. *Controlling Region*

The VAPRES controlling region contains a Microblaze processor and a set of static peripherals. The controlling region is responsible for three main functions: controlling data processing region operation via PRSockets (Section III.B), performing system-level functions such as reading hardware module bitstreams from external memory and performing PR via the ICAP, and executing software modules.

B. *Data Processing Region*

The data processing region contains one or more reconfigurable streaming blocks (RSBs). Each RSB has several PRRs (each RSB can have a different number of PRRs), I/O modules (IOMs), and an inter-module communication architecture. IOMs and the inter-module communication architecture are located inside the static region of the system. Figure 1 depicts a sample VAPRES system with one RSB containing three PRRs and two IOMs. PRRs and IOMs within each RSB communicate using the inter-module communication architecture. IOMs directly interface to external I/O pins or peripherals (i.e. ADCs, DACs, etc.). PRRs interface with the Microblaze processor through asynchronous FSL (fast simplex link) interfaces. The inter-module communication architecture consists of a linear array of switch boxes. Each PRR and IOM connects/pairs to/with one switch box through asynchronous FIFO-based module interfaces, which connect to a hardware module's input port (*consumer interface*) and output port (*producer interface*). Figure 2 depicts the internal architecture of a producer interface and a consumer interface.

For each switch box-PRR or switch box-IOM pair, a PRSocket allows the Microblaze processor to control switch box, hardware module, IOM, and module interface operation. PRSockets contain one *device control register* (DCR) [11] and additional interfacing logic. The DCR connects as a slave peripheral to the Microblaze processor through a PLB-to-DCR (Processor Local Bus) bridge. Figure 3 depicts the structure of a sample PRSocket with one consumer interface and one producer interface for each PRR and Table 1 defines the associated PRSocket DCR bits and functions.

From a top-level view, each switch box has several input and output ports connected to adjacent switch boxes or to module interfaces. Internally, a switch box consists of a set of multiplexers and one register connected to each switch box input port. To establish a streaming channel between a producer and a consumer interface, the Microblaze controls the multiplexer select signals (MUX_sel bits in the PRSocket DCR) for each switch box on the path between the two communicating module interfaces. After streaming channel establishment, data words flow from the producer to the consumer interface in a pipelined fashion using switch box registers. This pipelined communication increases the

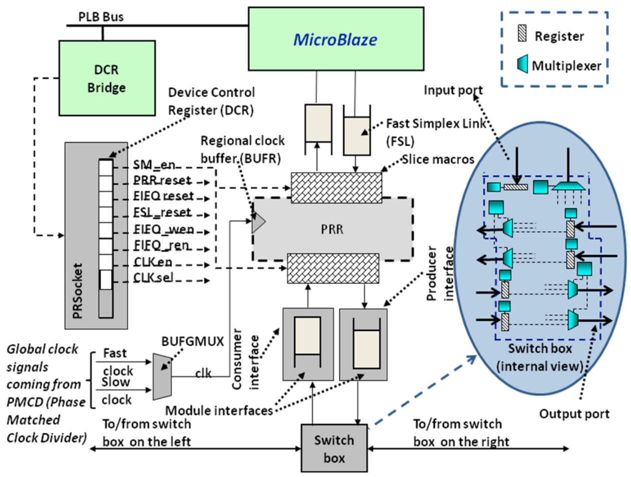

Figure 3: PRSocket signals to PRR, switch box, and module interfaces.

maximum communication clock frequency, and thus throughput, by reducing routing and combinational delays between registers.

When a producer interface FIFO contains data words and the Microblaze asserts FIFO_ren of the corresponding PRSocket DCR (Table 1), the producer interface reads the data words from the interface's internal FIFO. In order to ensure only valid data words are transferred between producer and consumer interface FIFOs, the producer interface bit-extends the data words by adding the negated FIFO empty flag as one extra most significant bit (MSB). A streaming channel transports the extended data words from the producer to the consumer interface. The MSB of the received data words serve as the write enable for the consumer interface FIFO.

When a consumer interface FIFO becomes full, all subsequent data words are discarded. However, a feedback FIFO full signal pipelined backwards on the streaming channel from the consumer to the producer interface avoids this data loss. In order to account for pipeline latency, the consumer interface asserts the feedback FIFO full signal when the consumer FIFO's remaining space is $2*(N-d)$, where N is the maximum FIFO capacity and d is the number of switches between the two communicating PRRs/IOMs.

1) *RSPS runtime assembly*

The process of *RSPS runtime assembly* consists of placing hardware modules in PRRs and establishing on-demand inter-module communication through the inter-module communication architecture. RSPSs assembled using the inter-module communication architecture approximates a Kahn Process Network (KPN), a widely used model for implementing streaming digital signal processing systems [8]. Hardware modules map to KPN nodes and module interface FIFOs and FSLs map to KPN stream buffers. Figure 4 shows a possible mapping of nodes and buffers of an example KPN inside a VAPRES RSB.

Hardware modules read/write data from/to module interfaces and FSLs through FIFO-based ports, which offer advantages over alternative NoC (network-on-chip) architecture interfaces [1][2]. First, hardware modules can read/write to/from FIFOs using a simple, well-known communication protocol instead of the complex addressing

Bit	Name	Function
0	SM_en	Enables/disables slice macros between the PRR and the static region
1	PRR_reset	Reset signal for the hardware module inside the PRR
2	FIFO_reset	Reset signal for the FIFOs inside the module interfaces
3	FSL_reset	Reset signal for the FIFOs inside the FSLs
4	FIFO_wen	Enables/disables the switch box to write data to the consumer interface
5	FIFO_ren	Enables/disables the switch box to read data from the producer interface
6	CLK_en	Enables/disables the clock signal for the PRR
7	CLK_sel	Select signal for the BUFGMUX primitive feeding the PRR clock signal
31..8	MUX_sel	Select signals for multiplexers inside the switch box

Table 1: PRSocket DCR bits and associated functions

and synchronization schemes common in NoCs. FIFOs transparently implement blocking-read and blocking-write synchronization mechanisms when hardware modules detect FIFO empty and FIFO full signals, respectively. Secondly, FIFO-based ports increase the system design abstraction level, enabling application designers to develop hardware modules independently of VAPRES architecture details. However, application designers must encapsulate hardware modules (the *original modules*) inside special *module wrappers* to connect the original module's input and output ports with the external FIFO-based ports.

2) *Local clock domains (LCDs)*

Local clock domains (LCDs) enable an RSPS to regulate data processing throughput. For example, in a system with a series of digital filter hardware modules and a fixed processing throughput requirement, some hardware modules may require more processing cycles, and thus a higher clock frequency than other hardware modules. To provide this configurability, the VAPRES static region and PRRs are independently clocked, and each constitutes a separate LCD. The Microblaze sets LCD clock frequencies using the PRSocket DCR clk_sel bits (Table 1). The asynchronous FIFOs inside the FSLs and module interfaces provide isolation between the PRRs and the static region LCDs.

In order to implement PRRs as LCDs on the Virtex-4, PRRs must be constrained to fit inside a group of adjacent Virtex-4 local clock regions [6]. Virtex-4 local clock regions vertically span sixteen CLBs and horizontally span half of the FPGA device. To ensure successful system implementation, local clock regions used by different PRRs may not intersect. In addition, we used Virtex-4 regional clock buffers (BUFRs) [6] to implement buffered clock signals inside each PRR and Virtex-4 clock multiplexer primitives (BUFGMUX) to generate the clock signals feeding the BUFR's clock inputs. Since a Virtex-4 BUFR can only drive the two regional clock

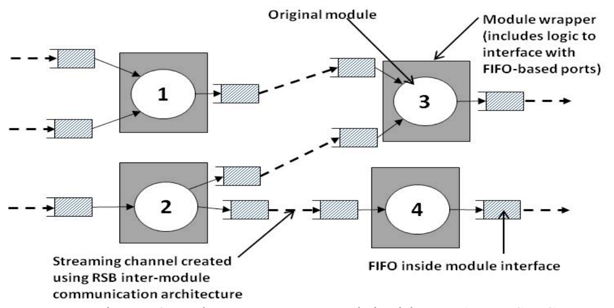
Original module
Module wrapper (includes logic to interface with FIFO-based ports)
1
2
3
4
Streaming channel created using RSB inter-module communication architecture
FIFO inside module interface

Figure 4: Kahn process network inside a VAPRES RSB

nets in the same local clock region where the BUFR is located and the two clock nets in the adjacent local clock regions (up to three local clock regions), the PRR height must be no greater than 3x16=48 CLBs. The PRSocket DCR clk_sel bits connect to the BUFGMUX select signals, therefore enabling the Microblaze to configure the PRR clock frequency during runtime. We implemented the multiple clock signals feeding the BUFGMUX primitives using the Virtex-4 DCM (Digital Clock Manager) and PMCD (Phase Matched Clock Divider) primitives.

3) *Hardware module switching methodology*

Efficiently leveraging PR for hardware module switching presents several challenges. First, PR imposes stream processing interruption because the reconfigured PRR must halt operation as the new hardware module is loaded. However, since the new hardware module is downstream from other hardware modules, the upstream hardware modules must halt operation. Since PRR reconfiguration can take on the order of hundreds of milliseconds [4][7], this stream processing interruption may be unacceptable. In some cases, module interface FIFOs can buffer data to alleviate stream processing interruption. However, for RSPSs with high stream processing throughput requirements, FIFOs may fill quickly, resulting in significant stream processing delays. Second, in many RSPSs, a new hardware module's initial operational state must match the replaced hardware module's current operational state. Additionally, the replaced hardware module may have computed dynamic variables required by the new hardware module. The capability to save and restore state registers inside hardware modules enables the operational state and dynamic variables to be transferred from the replaced hardware module to the new hardware module.

VAPRES addresses these challenges using a custom hardware module switching methodology. Figure 5 exemplifies this methodology using a digital filter example where circled numbers indicate intermediate steps. The system is composed of one RSB with one IOM and two PRRs. P0, p1, and p2 denote the producer module interface FIFOs and c0, c1, and c2 denote the consumer module interface FIFOs. R0, r1, and r2 denote the FSL links flowing towards the Microblaze and t0, t1, and t2 denote the FSL links flowing towards the PRRs/IOMs. This example assumes that prior to RSPS operation, the Microblaze placed filter A inside the first PRR and configured switch boxes SW1 and SW2 to establish streaming channels between p0 and c1 and between p1 and c0.

The RSPS initially operates as follows: filter A receives streamed input data from the IOM and sends the processed streamed output data back to the IOM (step 1). While filter A processes data, filter A periodically sends monitoring information about input data characteristics through r1 to the Microblaze processor (step 2). The Microblaze evaluates this monitoring information to determine if filter B would better meet the design constraints (i.e. reduced power, higher precision, etc.). If filter B is determined to be more appropriate, the Microblaze reconfigures the second PRR to store filter B while filter A continues data processing (step 3).

After the second PRR reconfigures to filter B, the Microblaze configures the switch boxes to release the streaming channel between p0 and c1, in addition to establishing a new streaming channel between p0 and c2 (step 4). Filter A continues processing the remaining data words present in the consumer interface FIFO. After processing the remaining data, filter A sends a special end of stream word (represented by "10101…0" (32 bits)) to the IOM (step 5) and the state register values to the Microblaze through r1 (step 6). The Microblaze initializes filter B using the state register values (step 7). After the IOM detects the special end of stream word arriving from c0, the IOM informs the Microblaze that filter A operation has completed by writing a message through r0 (step 8). The Microblaze configures the switch boxes to connect p2 and c0, completing hardware module switching (step 9).

This hardware module switching methodology overlaps module operation with PRR reconfiguration, which avoids stream processing interruption. The new hardware module is placed outside the current RSPS processing path and begins operation only after partial reconfiguration has finished.

IV. VAPRES System Design and Implementation

Creating an FPGA-based PR SoC using the VAPRES architecture requires two design flows: (1) the base system flow assists system designers in creating a VAPRES base system (Figure 6 right), and (2) the application flow assists application designers in creating applications to run on the VAPRES base system (Figure 6 left).

A. *Base System Flow*

In the base system flow's first step, the system designer determines the *base system specifications* by specializing the VAPRES architectural parameters. In order to leverage

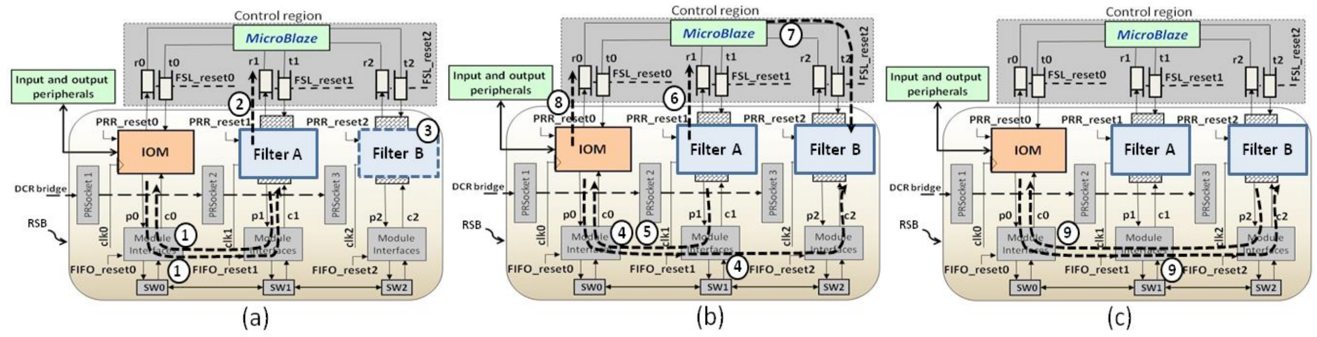
Control region
MicroBlaze
r0 t0 FSL_reset0 r1 t1 FSL_reset1 r2 t2 FSL_reset2
Input and output peripherals
PRR_reset0 PRR_reset1 PRR_reset2
IOM
Filter A
Filter B
DCR bridge
RSB
PRSocket 1 PRSocket 2 PRSocket 3
p0 c0 p1 c1 p2 c2
clk0 clk1 clk2
Module Interfaces
FIFO_reset0 FIFO_reset1 FIFO_reset2
SW0 SW1 SW2
1 2 3 4 5 6 7 8 9
(a)
(b)
(c)

Figure 5: Switching digital filters (hardware modules) inside a VAPRES RSB: (a) Initial RSPS operation and placement of filter B in the second PRR; b) Intermediate RSPS operation and detection of the end of stream condition; (c) Final RSPS operation. Circled numbers indicate intermediate steps.

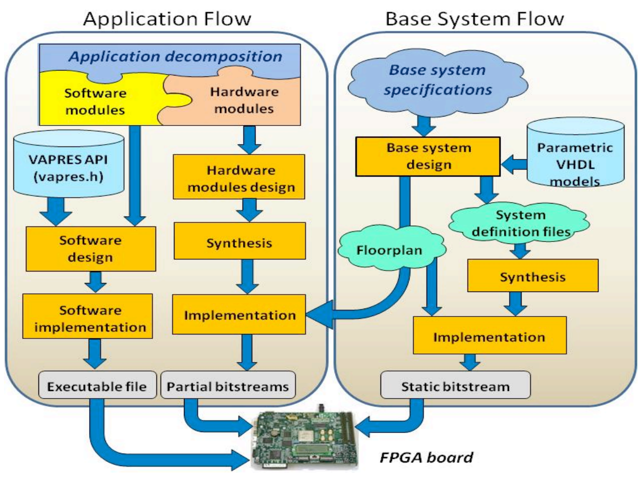

Figure 6: VAPRES design and implementation flows

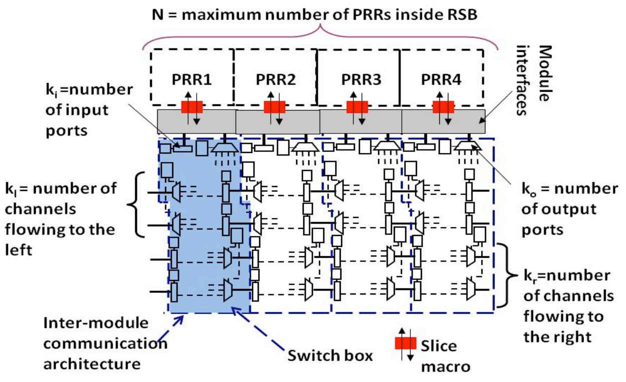
N = maximum number of PRRs inside RSB
PRR1
PRR2
PRR3
PRR4
Module interfaces
k_i=number of input ports
k_l= number of channels flowing to the left
k_o = number of output ports
k_r=number of channels flowing to the right
Inter-module communication architecture
Switch box
Slice macro

Figure 7: Sample RSB with the following architectural parameters: N=4, w=32, kr=2, kl=2, ki=1, ko=1

reusability and architectural specialization, Figure 7 shows the VAPRES data processing region's architectural parameters for a single RSB. Architectural parameters include the maximum number of PRRs (*N*), communication channel width (*w* bits), number of one-way communication channels between switch boxes (*kr* channels flowing to the right and *kl* channels flowing to the left), and the number of input channels (*ki*) and output channels (*ko*) between each PRR and the connected switch box. This architectural specialization supports a wide variety of hardware module and application requirements and enables system designers to balance resource utilization with communication flexibility.

In the *base system design* step, the system designer designs the base system floorplan and creates the *system definition files*. 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.

To ensure that the VAPRES floorplan is suitable for the Virtex-4, system designers must ensure that each PRR fits inside one to three adjacent local clock regions and that local clock regions used by different PRRs do not intersect. In general, three adjacent local clock regions are required for PRRs containing large hardware modules, but large PRRs can increase resource fragmentation (wasted resources when a hardware module requires fewer resources than a PRR provides). An alternative solution constrains PRRs to fit within one local clock region, and hardware modules that require more resources than a PRR provides can span multiple adjacent PRRs. Finally, the *synthesis* and *implementation* steps generate the base system's static bitstream.

B. *Application Flow*

After downloading the base system's bitstream to the FPGA device, an application designer designs applications for the base system. The application designer decomposes an application into software and hardware modules using hardware/software co-design techniques. After decomposition, the hardware and software modules follow two separate flows. During the *software module design* flow, the application designer writes the application software that will run on the Microblaze processor. In order to assist the application designer in writing software modules for the VAPRES systems, Application Program Interface (API) functions provide low-level system functionality (Table 2). For example, *vapres_CF2ICAP* and *vapres_array2ICAP* allow reconfiguration of a PRR when the partial bitstream is stored either as a file in external compact flash memory or as an array in external SDRAM, respectively. Additionally, *vapres_establish_ channel(comm._state* current_state, Xuint8 prrx, Xuint8 prry)* establishes a streaming channel between PRR X and PRR Y, where *current_state* stores the available switch box channels. The function returns one and updates *current_state* if the streaming channel is successfully established, or zero otherwise.

During the *hardware module design* flow, the application designer designs the hardware modules and hardware module wrappers. Application designers are insulated from low-level PR design tasks involving PRR definition, floorplanning, and other base system implementation details. However, the application designer must consider the number of, data-width, and type of input and output ports connected to each hardware

Function	**Purpose**
int vapres_CF2ICAP(XHwIcap *hwicap, Xuint8* filename) ;	Transfers a partial bitstream stored as a file in CF memory to ICAP port
int vapres_array2ICAP(XHwIcap *hwicap, char* bitstream) ;	Transfers partial bitstream stored as a ***bitstream*** array in SDRAM to ICAP port.
int vapres_CF2array(char* bitstream, int* size, Xuint8* filename) ;	Transfers a partial bitstream file from CF memory to a ***bitstream*** array in SDRAM. Array size is returned on argument ***size***.
int vapres_module_clock (int num, bool enable);	Enables the regional clock buffer (BUFR) for HW module identified by ***num***
int vapres_module_reset(int num, bool assert);	Resets the HW module identified with number ***num***
int vapres_module_write(int num, int value);	Writes ***value*** to hardware module input identified with number ***num***
int vapres_module_read(int num, int value);	Reads a ***value*** from the ***num***-th hardware module identified with number ***num***
int vapres_establish_channel(comm._state* current_state, Xuint8 prrx, Xuint8 prry)	Establishes a streaming channel between PRRs identified with number ***X*** and ***Y***

Table 2: Sample VAPRES API functions.

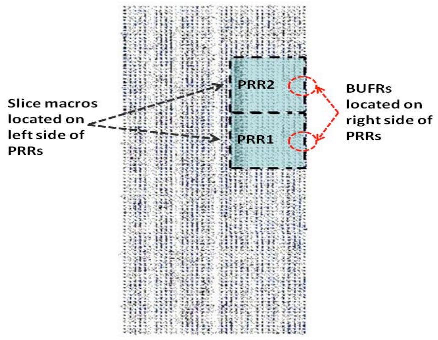
Slice macros located on left side of PRRs
PRR2
PRR1
BUFRs located on right side of PRRs

Figure 8: VAPRES prototype floorplan on the VLX25 indicating location of regional clock buffers (BUFRs) and slice macros.

module. A hardware module's input and output port type can be an FSL slave (reads data from an FSL link), an FSL master (writes data to an FSL link), a consumer port (reads data from a consumer interface), or a producer port (writes data to a producer interface). During the application flow, only logic associated with each hardware module is synthesized and placed and routed, as the base design logic remains unchanged. This isolation between the application flow and the base system flow reduces synthesis and place and route times, which otherwise can be exceedingly high during the iterative development and testing stages of large, complex designs.

V. ANALYSIS AND RESULTS

A. Experimental Setup

We implemented a VAPRES prototype system on a Xilinx ML401 evaluation board to test system functionality and evaluate the reconfiguration time for individual PRRs. Figure 8 depicts the FPGA fabric layout consisting of one RSB with two PRRs and one IOM (sufficient for functionality testing purposes). We customized the inter-module communication architecture with two 32-bit channels flowing both left and right between switch boxes and one 32-bit module input port and one 32-bit module output port connecting PRRs to switch boxes. Module interface FIFOs and FSL links were implemented using Virtex-4 BlockRAM, which buffer 512 32-bit words. The Microblaze processor and switch boxes executed at 100 MHz. In addition, PRRs were constrained to fit inside separate Virtex-4 local clock regions and contained 640 slices, which spanned sixteen vertical CLBs and ten horizontal CLBs. We point out that these PRR sizes are relatively small, and larger PRRs might be required for applications with larger hardware modules, but however are sufficient for testing purposes.

B. Prototype Evaluation

The VAPRES static region (including the Microblaze soft-core processor and the inter-module communication architecture) required 9,421 slices (approximately 86% of the VLX25), of which the inter-module communication architecture required only 1,020 slices (approximately 15% of the VLX60 device). We generated both static and partial bitstreams with the Xilinx Early Access Partial Reconfiguration Flow [12]. Hardware module partial bitstreams were stored as files in external flash memory.

We evaluated PRR reconfiguration time for the *vapres_CF2ICAP* and *vapres_array2ICAP* functions using the Microblaze xps_timer peripheral. Reconfiguration of a single PRR using *vapres_CF2ICAP* accounted for 1043,388,614 clock cycles (1.043s) of which transferring the partial bitstream from flash memory to the ICAP BRAM buffer accounted for 95.3% of the time and writing the partial bitstream to the ICAP accounted for 4.7% of the time. Reconfiguration of a single PRR was reduced to 71,944,572 clock cycles (71.94 ms) when using the *vapres_array2ICAP* function (partial bitstream was copied from flash memory to an array in SDRAM memory at system startup). Since partial bitstream size will directly influence reconfiguration time and thus system performance, a focus of our future work includes analyzing the tradeoffs between resource fragmentation and system performance for large verses small PRRs.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we designed and prototyped VAPRES – a multipurpose PR FPGA SoC for reconfigurable streaming processing systems (RSPSs). VAPRES enables intense architectural specialization to meet design constraints through numerous architectural parameters and local clock domains. A novel hardware module switching methodology enables dynamic system reconfiguration without stream processing interruption. In order to assist system and application designers in developing VAPRES base systems and applications, we formulated two customized design flows. Future work includes additional design support in the form of scripting tools for system floorplan definition and system definition file creation.

ACKNOWLEDGEMENTS

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] E. Beigne, P. Vivet. Design of on-chip and off-chip interfaces for a GALS NoC architecture. 12th IEEE International Symposium on Asynchronous Circuits and Systems, 2006. March 2006
[2] C. Bobda. Introduction to Reconfigurable Computing. Architectures, Algorithms and Applications. Springer, 2007
[3] C. Conger, A. Gordon-Ross. A. George. FPGA Design Framework for Partial Run-Time Reconfiguration. ERSA, 2008.
[4] E. El-Araby, I. Gonzalez, T. El-Ghazawi: Exploiting Partial Runtime Reconfiguration for High-Performance Reconfigurable Computing. ACM Trans. on Reconf. Technology and Systems (TRETS), 2009
[5] J. Emmert, C. Stroud, M. Abramovici. Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration. FCCM, 2000
[6] E. Eto. BUFR in partial reconfigurable modules. Xilinx WP 344, 2008.
[7] R. Hymel, A. D. George, H. Lam. Evaluating Partial Reconfiguration for Embedded FPGA Applications. HPEC, 2007
[8] P. Sedcole, P. Cheung, W. Luk: Run-Time Integration of Reconfigurable Video Processing Systems. IEEE Trans. VLSI Syst. 15(9), 2007
[9] A. Sudarsanam, R. Barnes, J. Carver, R. Kallam, A. Dasu Dynamically Reconfigurable Systolic Array Accelerators: A case Study with EKF and DWT algorithms. In-print IET Comput. and Digit. Tech., 2010
[10] M. Ullmann, B. Grimm, M. Hübner, J. Becker. An FPGA Run-Time System for Dynamical On-Demand Reconfiguration. IEEE Parallel and Distributed Processing Symposium, 2004
[11] Xilinx Inc. Device Control Register Bus (DS402, v. 2.9), May 2005
[12] Xilinx Inc. Early Access PR User Guide (v1.1). March 2006