On-chip Context Save and Restore of Hardware Tasks on Partially Reconfigurable FPGAs

Aurelio Morales-Villanueva and Ann Gordon-Ross
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering, University of Florida
Gainesville, Florida, USA
E-mail: {morales, ann}@chrec.org

***Abstract*—Partial reconfiguration (PR) of field-programmable gate arrays (FPGAs) enables hardware tasks to time multiplex PR regions (PRRs) by isolating reconfiguration to only the reconfigured PRR, which avoids halting the entire FPGA's execution. Time multiplexing PRRs requires support for unloading/loading tasks and for resuming a task's execution state. In order to resume a task's execution state, the execution state (context) must be saved when the task is unloaded so that the execution state can be restored when the task resumes—context save (CS) and context restore (CR), respectively. In this paper, we present a software-based, on-chip context save and restore (CSR) for PR-capable FPGAs. As compared to prior work, our CSR is autonomous (i.e., does not require any external host support), does not require custom on-chip hardware, is portable across any system design, and does not require tool flow modifications or special tools. Experimental results extensively evaluate the CSR execution time based on PRR size, enabling designers to trade off PRR granularity for CSR execution time based on application requirements.**

Keywords—FPGA, partial reconfiguration, context save and restore.

I. INTRODUCTION

Partial reconfiguration (PR) enhances FPGA reconfigurability by isolating reconfiguration to pre-defined PR regions (PRRs). PR partitions the FPGA fabric into a static region and one or more PRR(s). The static region is fixed (i.e., is never reconfigured) and only the reconfigured PRR halts operation while the remainder of the FPGA continues execution. Hardware tasks, also referred to as PR modules (PRMs), are mapped to and run in PRRs. Several PRMs may time multiplex the same PRR, which reduces total FPGA area requirements and power consumption.

In order to time multiplex shared PRRs, dynamic task switching (loading/unloading a task in a PRR) is required. Task switching is challenging if the task's execution state must be restored when the task is reloaded, which is essential for applications such as dynamic load balancing and task migration between pooled FPGA resources. Thus, unloading a task requires saving the task's execution state (i.e., context save (CS)) by saving the flip-flops' values, and loading a task requires restoring the task's execution state (i.e., context restore (CR)) by restoring the saved flip-flops' values. With context save and restore (CSR), the reloaded task avoids lengthy re-execution time to return to the task's execution state when the task was unloaded. For this purpose, CSR requires a new *initial* partial bitstream created by *merging* the task's *initial* partial bitstream with the saved context, creating a *merged* partial bitstream that effectively restores the task's execution state.

In this work we propose, to the best of our knowledge, the first autonomous, software-based, on-chip CSR, which executes in an on-chip softcore processor in the static region and uses the FPGA's internal configuration access port (ICAP) for PRR reconfiguration. Our CSR does not require custom hardware, the CSR software is portable/reusable across any application/system design, and requires no design tool flow changes. We give a detailed description of our CSR and provide implementation results for a Xilinx Virtex-5 LX110T FPGA with a MicroBlaze softcore processor running an embedded Linux operating system. Our CSR methods can be easily ported to any newer PR-capable Xilinx device family, such as the Virtex-6, 7-series, and Zynq-7000. Results show that the execution times for CSR from a small to a large PRR are on the order of milliseconds. The main contributions of our work are the fundamental CSR constructs and methodologies, including detailed information about the CSR process, which are intended to enable system designers to quickly incorporate CSR into their FPGA system designs.

II. RELATED WORK

Prior CSR research is not easily comparable since the works used different designs and devices, which imposed different inherent inefficiencies with respect to the FPGA's internal components (e.g., ICAP, flip-flop distribution) and connections (e.g., SelectMAP, JTAG) with external devices (e.g., host, microcontroller) and these inefficiencies were not always explicitly included in the results.

Landaker et al. [4] presented a non-autonomous CSR method, which used the FPGA as a co-processor installed on a board attached to a host CPU. Results on Xilinx Virtex XCV1000 showed that CS, CR, and bitstream merging for the entire device (PR was not leveraged) required 407 ms, 365 ms, and 465 ms, respectively. Kalte et al. [3] extended non-autonomous CSR to leverage PR-capable FPGAs and introduced context relocation—the ability to CS from one

PRR and CR to a different PRR. Results for a Xilinx Virtex XCV2000E showed that CS and CR required 1.2 ms and 6.8 ms, respectively, for a PRR with 2,287 flip-flops, but these results did not include the communication overhead between the FPGA and the host CPU or SelectMAP's inefficiencies. Jozwik et al. [2] presented a similar CSR method that reduced CS and CR times using a modified ICAP to support direct memory access (DMA). Results showed that CS and CR required 60.11 us and 536.9 us, respectively, for a PRM with 612 flip-flops on a Virtex-4 FPGA. The bitstream merging time was not reported and the FPGA/CPU communication overhead was not considered.

Autonomous hardware-based CSR eliminates the requirement for slow external host communication and reduces CSR times. Jovanovic et al. [1] proposed a modified scan-path chain of flip-flops for CSR. Jovanovic's CSR method introduced a 30% overhead in flip-flops and a 22% decrease in maximum clock frequency on a Xilinx Virtex II, but no results for CS and CR times were included.

III. VIRTEX-5 FPGA ARCHITECTURE

CSR is a complex and intricate process that requires detailed device knowledge, therefore this section reviews the Virtex-5 FPGA device architecture (see [5] for details).

A. Device Layout and Resources

The Virtex-5 device family's architecture is similar to newer Xilinx devices (e.g., Virtex-6 and 7-series). The Virtex-5 supports two-dimensional (2-D) PR, which allows PRRs to occupy any rectangular region on the device. The Virtex-5 can be configured externally using the JTAG or SelectMAP interfaces or internally using the ICAP. The FPGA's resources (CLBs, DSPs, BRAMs and IOBs) are organized into rows and columns. Each row contains multiple columns and each column contains multiple frames, where the number of frames per column depends on the type of resource in the column. This organization allows 2-D PRRs to contain any mix of resources.

B. Device Configuration

Since CSR uses the ICAP, all partial bitstreams used to reconfigure the PRRs must be 32-bit word aligned (by extracting the header in the partial bitstreams created by Xilinx tools), generating *initial partial bitstreams*.

During CSR, the resources' values in the static region must be maintained (i.e., protected) during successive reconfigurations of the PRRs. Protection/unprotection of the resources is performed by either disabling/enabling the startup sequence [5] from changing the resources' values using the Xilinx user primitive STARTUP_VIRTEX5. Protection of the static region is done only once and protection/unprotection of the PRRs is done for every CR on the reconfigured PRR. Protection/unprotection of the device is nonintrusive and does not disrupt execution.

IV. AUTONOMOUS CONTEXT SAVE AND RESTORE (CSR)

Our autonomous CSR operates entirely on-chip using software, thus we assume the system has a softcore processor and the PRRs have already been defined on the device fabric.

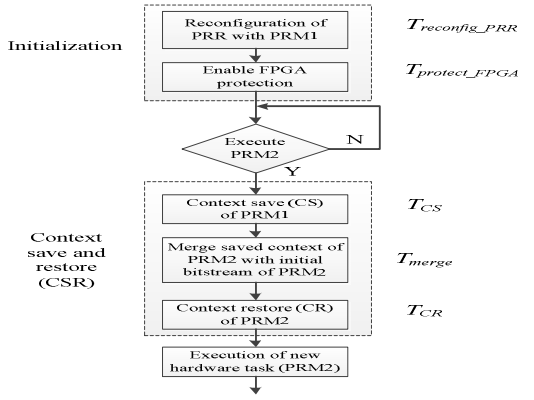

Figure 1. On-chip context save and restore (CSR) flow

Even though PRRs can contain many different resource types, in this section, we detail CSR for CLB resources only.

A. CSR Overview

We illustrate CSR for a system with one PRR, two PRMs (PRM1 and PRM2), and initial partial bitstreams for PRM1 and PRM2 to run in the PRR. Even though this is a small system, this system is sufficient to explain CSR. Figure 1 depicts an overview of the steps required for a sample execution scenario and the time required for each step T_x. We assume that (not shown in the figure) PRM2 executed in the PRR in the past, CS was already performed on PRM2, and currently the PRR is empty.

Initialization reconfigures the PRR with PRM1 by first transferring PRM1's initial partial bitstream to the device and then enabling the FPGA protection to avoid future startup sequences from reinitializing the static region. $T_{reconfig\_PRR}$ is the execution time required for reconfiguring the PRR. $T_{protect\_FPGA}$ is the execution time required to protect the FPGA.

CSR is comprised of three steps: CS, merging the saved context with the initial partial bitstream, and CR. Once CR completes, PRM2 will resume execution as if PRM1 had not interrupted PRM2's execution.

B. Context Save (CS)

To ensure correctness, before PRM1's CLBs' flip-flops' values can be captured, the PRR's clock must be stopped to avoid potential setup/hold violations. The PRR's clock must be stopped before CS and maintained until CR finishes. After capturing the frames containing PRM1's flip-flops' values (i.e., PRM1's context), a software capture-and-save loop converts the saved context to a file. The total execution time required for CS is given by T_{CS}.

C. Merging the Saved Context with the Initial Partial Bitstream

Before CR, the saved context must be merged with the initial partial bitstream to create a merged partial bitstream that will restore PRM2's execution state. The bitstreams are merged at the 32-bit word level by updating multiple flip-flop values in the word. Figure 2 depicts the bitstream merging process. Figure 2a) shows the truth table to update a

ini	cap	msk	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$f = cap \cdot msk + ini \cdot (/msk)$

(a)

msk =	1	0	0	1	0	1	0	1
cap =	1	1	1	0	0	1	0	0
ini =	1	0	1	1	0	0	1	0
cap.msk =	1	0	0	0	0	1	0	0
ini.(/msk) =	0	0	1	0	0	0	1	0
f =	1	0	1	0	0	1	1	0

(b)

Figure 2. Bitstream manipulation for merging the saved context with the initial partial bitstream: (a) bitstream manipulation truth table and (b) four bits being updated in a word boundary.

particular bit of the initial partial bitstream with the saved context, where *ini* denotes a bit in the initial partial bitstream, *cap* denotes the captured value, and *msk* denotes if the particular bit is part of the saved context. The updated value is denoted by *f*, where $f = cap * msk + ini * (/msk)$. Figure 2b) shows an example where there are four bits to be updated in a word boundary (reduced to 8 bits for clarity), producing a final *f* word equal to 10100110.

The merging process is entirely executed on the softcore processor in a software merge loop. T_{merge} denotes the execution time to merge the bitstream and save the merged partial bitstream.

D. Context Restore (CR)

Before CR, the PRR must be unprotected, which allows the PRR's flip-flops to be initialized to the values in PRM2's merged partial bitstream. We note that only the reconfigured PRR needs to be unprotected, while the remainder of the FPGA must remain protected. Next, PRM2's merged partial bitstream is sent to the device in order to reconfigure the PRR with PRM2's saved context. To reconfigure the PRR with PRM2's saved context, we force the execution of the startup sequence [5] by toggling GSR (global set reset input on STARTUP_VIRTEX5). Finally, the reconfigured PRR is protected to prevent the execution of the startup sequence on other PRRs from affecting the operations on the reconfigured PRR. The total execution time required for CR is given by T_{CR}.

V. Experimental Results

A. Experimental Setup

For our experiments, we used the Xilinx XUPV5-LX110T board (ML509) and the Xilinx ISE 12.4, XPS 12.4, and PlanAhead 12.4 tools. The FPGA's static region contained a MicroBlaze softcore processor running at 100 MHz. The reconfigurable region contained a single PRR with CLB resources. The MicroBlaze executed a Linux-like OS 2.6.37 that is based on BusyBox and we generated the binaries for the MicroBlaze using GNU tools. The SDRAM memory stored the OS, the initial and merged partial bitstreams, the CSR binaries, and provided storage for the CSR-generated files.

We modeled the execution scenario explained in Section IV.A with a small PRR (one CLB row by two CLB columns) and followed all of the steps and phases described in Sections IV.B through IV.D, verifying that after CR of PRM2, the first register value the MicroBlaze received was the last value received prior to executing CS on PRM2.

Table I. Execution Times (ms) for $T_{reconfig\_PRR}$

rows	1											
columns	1	2	3	4	5	6	7	8	9	10	11	12
PRR frames	36	72	108	144	180	216	252	288	324	360	396	432
PRM flip-flops	160	320	480	640	800	960	1120	1280	1440	1600	1760	1920
$T_{reconfig\_PRR}$	1.00	1.54	2.09	2.63	3.18	3.73	4.34	4.99	5.60	6.21	6.81	7.49

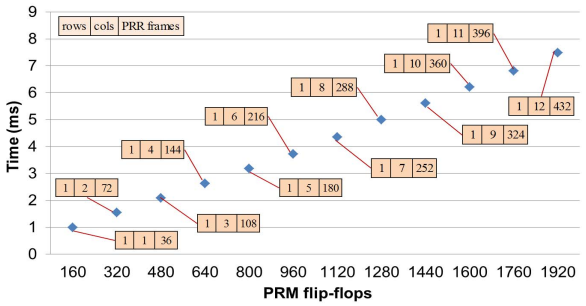

Figure 3. Execution times for $T_{reconfig\_PRR}$ (ms) with respect to the number of PRM flip-flops. The adjacent rectangles indicate the number of rows, columns, and frames for the PRRs.

To generate thorough results for increasing PRR sizes of fine-grained increments in the number of flip-flops, we created *pseudo-PRRs* for all possible initial partial bitstreams for an empty area (area where no CLB columns were already in use) using one row and increasing the number of CLB columns from one to twelve, which is the largest number of contiguous CLB columns on the target device. Performing CR over a pseudo-PRR did not affect the static region's operation and the implemented PRR after protection.

B. Execution Times

Table I through Table IV show the execution times (in milliseconds) for the CSR steps for different pseudo-PRR sizes (a single row of increasing numbers of CLB columns and flip-flops). Figure 3 through Figure 6 plot the results from Table I through Table IV, respectively, for varying numbers of flip-flops in the PRR versus execution time.

Table I and Figure 3 summarize the execution times for $T_{reconfig\_PRR}$, which depends on the number of configuration frames (36 per CLB column) and the number of columns in the PRR. Increasing the number of flip-flops reveals almost a linear increase in $T_{reconfig\_PRR}$. The execution time for $T_{protect\_FPGA}$ for the Virtex-5 LX110T is 67.91 ms. Table II and Figure 4 summarize the total execution time for CS (T_{CS}). Overall, T_{CS} shows an almost linear increase. Table III and Figure 5 summarize the execution time for T_{merge}, which depends on the number of PRM flip-flops in the PRR. Figure 5 shows a linear increase in T_{merge}. Table IV and Figure 6 summarize the total execution time for CR (T_{CR}), which shows almost a linear behavior.

The ICAP and SDRAM overheads and high cache miss rates for large bitstreams will increase the execution times' growth rates, thus making CSR less efficient as the PRRs increase in size.

Table II. EXECUTION TIMES (ms) FOR CS

rows	1											
columns	1	2	3	4	5	6	7	8	9	10	11	12
PRM frames	2	4	6	8	10	12	14	16	18	20	22	24
PRM flip-flops	160	320	480	640	800	960	1120	1280	1440	1600	1760	1920
T_{CS}	4.95	5.64	6.33	7.03	7.74	8.47	9.31	10.08	10.81	11.56	12.30	13.06

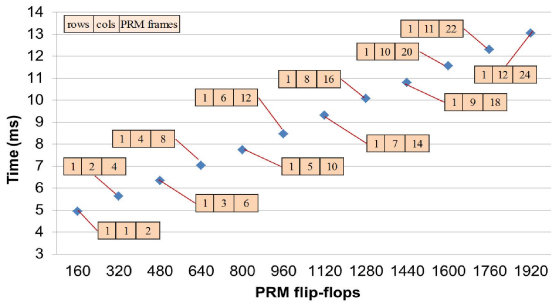

Figure 4. Execution times for T_{CS} (ms) with respect to the number of PRM flip-flops. The adjacent rectangles indicate the number of rows, columns, and frames in the PRM's context.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the first, to the best of our knowledge, software-based, on-chip context save and restore (CSR). Our CSR does not require an external host, and is thus autonomous. Our CSR significantly enhances the functionality of FPGA devices by enabling task execution resumption after a task is unloaded from a PRR, which is suitable for applications such as dynamic load balancing. As compared to prior works, our CSR is application independent, and thus portable across any system design.

Our results detail the CSR's execution times' growth rates based on increasing PRR size, making CSR most attractive for finer-granularity PRRs, allowing a designer to trade off PRR size and CSR overheads based on application-specific requirements. Our future work will extend CSR's functionality to include LUTRAM for CLBs, DSP, BRAM, and IOB blocks and we will incorporate context relocation.

Table III. EXECUTION TIMES (ms) FOR T_{merge}

rows	1											
columns	1	2	3	4	5	6	7	8	9	10	11	12
PRM frames	2	4	6	8	10	12	14	16	18	20	22	24
PRM flip-flops	160	320	480	640	800	960	1120	1280	1440	1600	1760	1920
T_{merge}	4.50	5.85	7.22	8.57	9.91	11.31	12.65	14.01	15.39	16.73	18.04	19.39

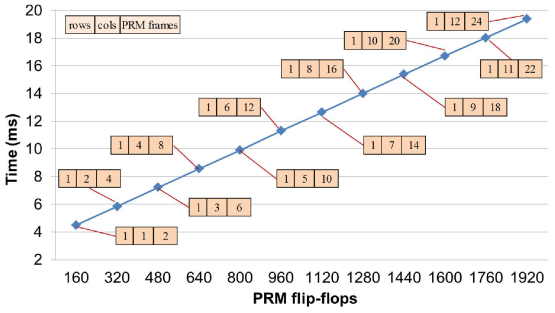

Figure 5. Execution times for T_{merge} (ms) with respect to the number of PRM flip-flops. The adjacent rectangles indicate the number of rows, columns, and frames in the PRM's context.

Table IV. EXECUTION TIMES (ms) FOR CR

rows	1											
columns	1	2	3	4	5	6	7	8	9	10	11	12
PRR frames	36	72	108	144	180	216	252	288	324	360	396	432
PRM flip-flops	160	320	480	640	800	960	1120	1280	1440	1600	1760	1920
T_{CR}	4.00	4.80	5.61	6.41	7.22	8.04	8.88	9.70	10.52	11.38	12.34	13.23

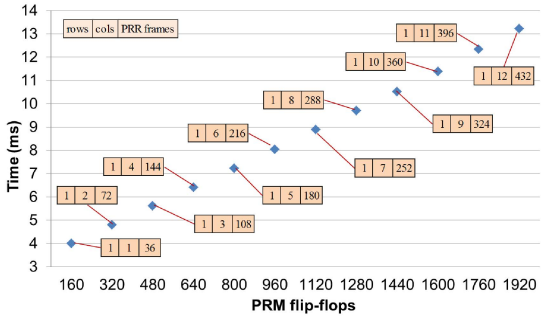

Figure 6. Execution times for T_{CR} (ms) with respect to the number of PRM flip-flops. The adjacent rectangles indicate the number of rows, columns, and frames in the PRR.

ACKNOWLEDGMENTS

This work was supported by FINCyT under contract N° 121-2009-FINCyT-BDE and in part by the I/UCRC Program of the National Science Foundation under Grant Nos. EEC-0642422 and IIP-1161022. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors gratefully acknowledge the support of Universidad Nacional de Ingeniería – Lima, Perú, the Presidencia del Consejo de Ministros, through the Programa de Ciencia y Tecnología – FINCyT, and the tools provided by Xilinx.

REFERENCES

[1] S. Jovanovic, C. Tanougast, and S. Weber, "A Hardware Preemptive Multitasking Mechanism Based on Scan-path Register Structure for FPGA-based Reconfigurable Systems," 2nd NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), pp. 358-364, 2007.

[2] K. Jozwik, H. Tomiyama, S. Honda, and H. Takada, "A Novel Mechanism for Effective Hardware Task Preemption in Dynamically Reconfigurable Systems," International Conference on Field Programmable Logic and Applications (FPL'10), pp. 352-255, 2010.

[3] H. Kalte and M. Porrmann, "Context Saving and Restoring for Multitasking in Reconfigurable Systems," International Conference on Field Programmable Logic and Applications, pp. 223-228, 2005.

[4] W. J. Landaker, M. J. Wirthlin, and B. L. Hutchings, "Multitasking Hardware on the SLAAC1-V Reconfigurable Computing System," Proc. of 12th International Conference on Field-Programmable Logic and Applications (FPL'02), pp. 806-815, 2002.

[5] Xilinx Inc., Virtex-5 FPGA Configuration User Guide v3.10 (UG191), Nov 18, 2011.