Configuration Prefetching and Reuse for Preemptive Hardware Multitasking on Partially Reconfigurable FPGAs

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

***Abstract*—Partially reconfigurable (PR) FPGAs enable preemptive hardware (HW) multitasking using PR regions (PRRs). To enable this multitasking, the HW task's partial bitstream is downloaded to only the task's PRR, and only that PRR is reconfigured. Since only a small portion of the FPGA fabric is reconfigured, reconfiguration time is significantly reduced as compared to reconfiguring the entire fabric, however this time is not negligible. Reconfiguration time can be reduced/hidden using two techniques: configuration prefetching and configuration reuse. Even though these techniques can effectively reduce/hide reconfiguration overhead, prior works in preemptive HW multitasking did not use these techniques. To the best of our knowledge, no prior work evaluated physical implementations of these techniques on PR FPGAs, which precludes consideration of physical-implementation-specific details, such as delays in accessing bitstreams, speed limitations during reconfiguration, etc. In this work, we present a novel implementation of configuration prefetching and reuse for preemptive HW multitasking on a Virtex-5 FPGA, however, our established fundamentals are device-family independent.**

I. Introduction and Motivation

A partially reconfigurable (PR) field-programmable gate array's (FPGA's) device fabric is logically partitioned into one static region and one or more PR regions (PRRs). PRRs are reconfigured by downloading a hardware (HW) task's partial bitstream using a device configuration port (such as the internal configuration access port (ICAP)), which affords faster reconfiguration time as compared to reconfiguring the entire device using a full bitstream. Since the static region is configured only once at device power up and the PRRs can be reconfigured without disrupting the execution of the static region or the other PRRs, partial reconfiguration enables isolated HW multitasking (i.e., HW task time-multiplexing).

There are two categories of HW multitasking: non-preemptive and preemptive. In non-preemptive HW multitasking, HW tasks execute from beginning to end, without interruption. Preemptive HW multitasking affords more task scheduling flexibility (e.g., closer adherence to task deadline and/or priority), where HW tasks may be interrupted, replaced by other HW tasks, and can resume execution at a later time.

Replacing a HW task in a PRR with another task requires reconfiguring the PRR with the new task's partial bitstream. The time to reconfigure a PRR—the configuration overhead—delays HW task execution, is PRR size dependent, and is an important design challenge, that if not considered, could significantly impinge system performance. System designers can use configuration prefetching and configuration reuse to reduce the impact of the configuration overhead on system performance. Configuration prefetching configures a PRR with a HW task's partial bitstream before the task's execution is required, which overlaps the PRR configuration while the same PRR is running another HW task. Configuration reuse reduces the total number of reconfigurations by reusing previously downloaded partial bitstreams.

To provide a complete evaluation of configuration prefetching and reuse for PR FPGAs, we present a novel physical implementation of HW multitasking for preemptable HW tasks on PR FPGAs. Our technique leverages the ICAP and modifications to the partial bitstreams generated by the Xilinx tools (Section III). Our work aids system designers in incorporating configuration prefetching and reuse in physical implementations of preemptive HW multitasking in any PR system, which, to the best of our knowledge, has not been addressed holistically in any prior work.

We evaluated our preemptive HW multitasking on a Xilinx Virtex-5 LX110T device using a MicroBlaze softcore processor running embedded Linux OS. Even though we evaluated this particular system setup, the fundamental process of preemptive HW multitasking using our configuration prefetching and reuse is equally applicable to other Xilinx devices, and is easily extended to non-preemptive HW multitasking.

II. Background and Related Work

Prior work extensively evaluated reducing PR FPGA reconfiguration overhead for non-preemptive HW multitasking using different scheduling techniques [1][3][9], but these prior works only presented simulated results, and did not evaluate physical implementations. Only Charitopoulos et al. [2] physically implemented different scheduling techniques for non-preemptive HW multitasking using a run-time system manager (RTSM) on a Zedboard, which uses the Xilinx Zynq 7020 device. However, the work did not overlap PRR reconfiguration with task execution in the same PRR, and thus, did not perform configuration prefetching.

Some prior works physically implemented preemptive HW multitasking on PR FPGAs, however, these works were not holistic solutions since no associated scheduling technique was presented. The REPLICA filter [6] introduced a HW implementation of preemptive HW multitasking that incorporated HW *task relocation*. Task relocation enables an executing HW task to be preempted, and the task's execution

state to be saved and later restored in another suitable PRR (i.e., same size and resource distribution) via *bitstream manipulations*, creating a new partial bitstream for the suitable PRR. Experiments were performed on a Xilinx Virtex-E FPGA.

Jozwik et al. [5] introduced two techniques to save and restore HW task execution state on Virtex-4 FPGAs. The first technique—configuration port access (CPA)—used the ICAP, and the second technique—task specific access structures (TSAS)—used extra logic for each HW task's flip-flops (FFs) in order to read/save the FFs' values, and subsequently write/restore the FFs' values.

To enhance the flexibility of configuration prefetching and reuse, and facilitate physical implementation of preemptive and non-preemptive HW multitasking in any PR system, we leverage fundamentals established in [4][7][8] for HW task preemption, and HW state saving and restoring on Virtex-6 [4] and Virtex-5 [7][8] devices. However, according to [5], access to FFs in DSPs (digital signal processing blocks) is restricted, thus DSPs cannot be used by HW tasks in preemptive HW multitasking, but still can be used in HW tasks in non-preemptive HW multitasking.

III. VIRTEX-5 FPGA DEVICE CONFIGURATION FOR HW MULTITASKING

Since physically implementing HW multitasking on PR FPGAs is a complex process that requires detailed device knowledge, we review the Xilinx Virtex-5 FPGA device configuration [10]. This information is necessary for assisting designers in incorporating configuration prefetching and reuse in any arbitrary PR system.

The Xilinx Virtex-5 FPGA [11] and newer device families, such as the Virtex-6 and -7 series, can be configured using full or partial bitstreams, which are used to configure the entire device or only a single PRR, respectively. The bitstream's configuration information is organized in configuration frames and is stored in the FPGA's internal configuration memory (CM). A configuration frame establishes the configuration of the device resources, such as DSPs, CLBs (configurable logic blocks), and BRAMs (embedded random access memory blocks), and the routing information to access these resources.

Full device configuration requires sequential execution of three phases: the setup phase, the bitstream loading phase, and the startup sequence phase [10]. While the configuration frames are downloading, the device continuously calculates the cyclic redundancy check (CRC) value. After downloading all of the configuration frames, the device verifies the CRC by comparing the calculated CRC with the bitstream's expected CRC. If the CRCs match, the startup sequence phase begins, which initializes the device's FFs and BRAMs, and the device enters the *user* mode.

Once the device is in user mode, partial reconfiguration of a PRR can be performed using the ICAP. PRR reconfiguration does not execute the startup sequence phase since the device is already in user mode, and the reconfigured PRR's FFs, BRAMs, and routing information are reconfigured with the new values in the CM defined in the partial bitstream, provided that CRC verification was successful.

Downloading a Xilinx-tool-generated partial bitstream to a PRR, while a task is executing in that PRR, causes the task to terminate and the new task starts execution immediately. This termination can be eliminated by downloading a modified partial bitstream, replacing the partial bitstream's CRC with the RCRC (reset CRC) sequence [10] in order to skip the CRC verification, thus the PRR's FFs and BRAMs will not change during the execution of the task (only the CM changes).

Future reinitialization of FFs and BRAMs can be forced by toggling the internal global set reset (GSR) signal using the Xilinx user primitive STARTUP_VIRTEX5 to execute the startup sequence phase if the CRC verification is skipped [4]. However, since toggling GSR reinitializes the entire device with the FFs' and BRAMs' initial values in the CM as defined in the full bitstream, a protection/unprotection mechanism for the static region and PRRs must be provided [8]. Protection/unprotection avoids/allows future reinitialization of FFs and BRAMs when the GSR is toggled. The GSR was used to facilitate state restoration of preempted HW tasks in [4][7][8], and can be used in configuration prefetching and reuse for HW multitasking on PR FPGAs. For HW multitasking, the static region must be protected at all times, and the PRRs are dynamically unprotected/protected (with a scheduler) for implementing configuration prefetching and reuse. We note that a protected PRR does not allow the task to save the task's FFs and BRAMs values.

For newer devices (e.g., Virtex-6/-7 series, Zynq-7000) and tools (e.g., starting from the Xilinx PlanAhead 14.3 tool [12]) the RESET_AFTER_RECONFIG=TRUE (RaR) constraint may be applied to PRRs in order to avoid the manual unprotection/protection of PRRs and manual protection of the static region after full configuration. The partial bitstream generated with this constraint contains the ICAP command sequence to protect the entire FPGA, unprotect/protect the PRR, and the GRESTORE and START commands [10] to force the startup sequence. However, preemptive HW multitasking requires generation of the CRC with custom hardware, which incurs hardware overhead (1,218 FFs and 5 BRAMs for the Virtex-4) [5]. Since partial bitstreams using the RaR constraint contain the ICAP commands to protect the entire FPGA before configuring the PRR [4], these partial bitstreams are extremely large in size as compared to partial bitstreams without using the RaR constraint, which increases the PRR reconfiguration time. Thus, all of the fundamentals explained in our work for the Virtex-5 are still valid for the newer devices.

IV. CONFIGURATION PREFETCHING AND REUSE FOR PREEMPTIVE HW MULTITASKING ON PR FPGAS

Since preemptable HW tasks may be interrupted, replaced by a different task, and resumed for execution, the preempted task's execution state must be saved before replacing that task with a new task, and the preempted task's execution state must be restored when the task resumes execution. To restore a task's execution state, the task's saved execution state is used to generate a new modified partial bitstream that includes the FFs' and BRAMs' saved values, using bitstream manipulations, before downloading the new modified partial bitstream.

Fig. 1 depicts HW multitasking of preemptable tasks in PR FPGAs using configuration prefetching and GSR a) for a sin-

gle-PRR system, and b) and c) for dual-PRR systems. In Fig. 1a), T_{LX}, T_{EX}, T_{SX}, and T_{RX} denote the times to reconfigure (L), execute (E), save the execution state (S), and restore the execution state (R) of task X (1, 2, and 3), respectively. We use the task X's execution state that was saved during T_{SX} to produce a new modified partial bitstream and reconfigure the PRR during T_{RX} to restore the previous task X's execution state and resume execution. In Fig. 1a), the PRR is unprotected at all times, and the execution of tasks *1*, *2*, *3*, and *1* at times t_1, t_2, t_3, and t_4, respectively, can be started by toggling GSR at these times. We cannot overlap T_{LX} for task X with T_{SY} for task Y because the ICAP can only be used for a single operation at a time.

Configuration reuse can be leveraged in this single-PRR system. In Fig. 1a), if we assume that *R1* is not executed, and task *3* needs to restart from task *3*'s last execution at t_4, re-execution of task *3* at t_4 can be done by toggling GSR at t_4, without downloading task *3*'s partial bitstream again.

Fig. 1b) and Fig. 1c) depict HW multitasking of four preemptable tasks in two PRRs using configuration prefetching without and with task execution overlap, respectively. T_{LXY}, T_{EXY}, T_{SXY}, and T_{RXY} denote the times to reconfigure, execute, save the execution state, and restore the execution state of task Y (1, 3 or 2, 4) in PRR X (1 or 2), respectively, and t_{XY} denotes the execution start time of task Y in PRR X.

In Fig. 1b), HW tasks start execution after another HW task in another PRR is preempted and the execution state of the preempted task is saved, where toggling GSR at times t_{22}, t_{13}, and t_{24} allows the execution of tasks *2*, *3*, and *4* in PRRs *2*, *1*, and *2*, respectively, where the PRRs are unprotected at all times. In Fig. 1c), the PRRs cannot be unprotected at all times because toggling GSR at times t_{22}, t_{13}, and t_{24} will affect the execution of tasks *1*, *2*, and *3*, during T_{E11}, T_{E22}, and T_{E13}, respectively. In Fig. 1c), the PRRs must be unprotected during times T_{LXY}, T_{SXY}, and T_{RXY}, with the clock that drives each PRR (with dedicated clock gating, using buffer BUFGCE) enabled only during times T_{E11}, T_{E22}, T_{E13}, and T_{E24}.

Configuration reuse can be leveraged in a multi-PRR system if task Y was already executed and preempted in PRR X, task Y needs to begin a subsequent execution from the same state that task Y started in during task Y's last execution, and if no other partial bitstreams were downloaded during task Y's last execution. In this case, task Y may restart execution in PRR X without executing a reconfiguration (T_{LXY}) by toggling GSR, provided that PRR X is unprotected.

V. Performance Evaluation of Preemptive HW Multitasking on PR FPGAs

Since non-preemptive HW multitasking addresses a subset of preemptive HW multitasking's challenges and considerations, this section evaluates the execution times of various steps required in preemptive HW multitasking systems on PR FPGAs using configuration prefetching and reuse.

A. Experimental Setup

We evaluated our preemptive HW multitasking using the Xilinx XUPV5 board with a Xilinx Virtex-5 LX110T device. We implemented our PR system using the Xilinx ISE 12.4, XPS 12.4 and PlanAhead 12.4 tools. Our system contained a

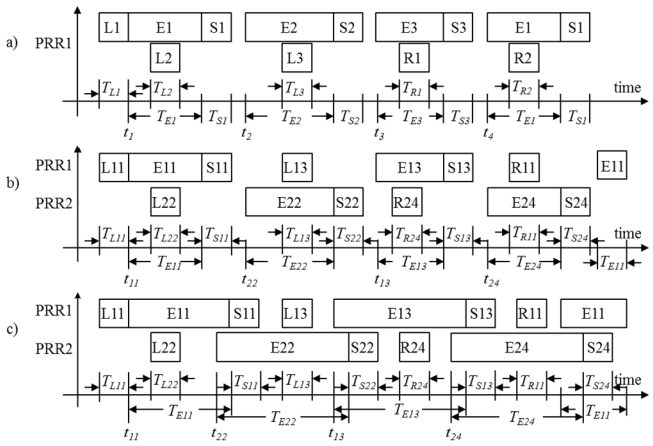

Fig. 1. HW multitasking of preemptable tasks a) with prefetching and preemption, and no configuration overhead in a single-PRR system, b) with prefetching and preemption, and no configuration overhead or task overlap in a dual-PRR system, and c) with prefetching, preemption, and task overlap, and no configuration overhead in a dual-PRR system.

MicroBlaze softcore processor, executing a Linux OS, that executed a software (SW) application that orchestrated the steps necessary for executing HW tasks, and for saving and restoring/resuming the execution state of preempted tasks.

The major steps of the SW application were: PRR reconfiguration, which initialized the PRR resources with the HW tasks' FFs' and BRAMs' initial values; HW task state saving, which saved the current values of the HW task's FFs and BRAMs to a file stored off chip in external SDRAM; and HW task state restoration, which restored the previously saved FF and BRAM values from the file. The evaluated HW tasks represented varying PRR sizes and organizations with one row, one to eleven CLB columns (160 to 1,760 FFs, respectively), and one BRAM column per PRR, which gave partial bitstreams sizes ranging from 32,704 to 91,744 bytes, which are representative of real world applications [5][6]. We measured the execution times of the SW application steps using the Xilinx programmable timer XPS_TIMER.

B. Execution Times

We evaluated and analyzed the execution time trends of the SW application steps with respect to ICAP, CPU, overhead (due to inefficiencies in the PR system), and total execution times, using the *syscall()* library function. The CPU times were in terms of the user time, and the total execution times were in terms of the wall-clock time, or elapsed time. The ICAP times are based on the ICAP's clock frequency, which is 100 MHz, the ICAP data bus width, which is 32 bits, and the amount of data transferred through the ICAP. The total execution times are the summation of the ICAP, CPU, and overhead times.

Table I through Table III depict the execution times for the SW application steps and for the different PRR sizes in terms of HW task FFs. Each value in Table I through Table III represents an average of five experiments. Table I summarizes the execution times for the PRR reconfiguration $T_{prr} = T_{prr\_icap} + T_{prr\_cpu} + T_{prr\_ov}$, where T_{prr}, T_{prr_icap}, T_{prr_cpu}, and T_{prr_ov}, are the total, ICAP, CPU, and overhead execution times, respectively.

Table II summarizes the execution times for saving the HW task's state $T_{save} = T_{save\_icap} + T_{save\_cpu} + T_{save\_ov}$, where T_{save},

Table I. EXECUTION TIMES (ms) FOR PRR RECONFIGURATION (T_{prr})

	HW task FFs										
Time	160	320	480	640	800	960	1120	1280	1440	1600	1760
T_{prr_icap}	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.2	0.2
T_{prr_cpu}	3.2	3.7	4.3	5.1	5.7	6.3	7.0	7.5	8.0	8.4	8.9
T_{prr_ov}	0.0	0.1	0.1	0.2	0.2	0.2	0.3	0.4	0.6	0.8	1.1
T_{prr}	3.3	3.9	4.5	5.4	6.0	6.7	7.5	8.1	8.8	9.4	10.2

Table II. EXECUTION TIMES (ms) FOR HW TASK STATE SAVING (T_{save})

	HW task FFs										
Time	160	320	480	640	800	960	1120	1280	1440	1600	1760
T_{save_icap}	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
T_{save_cpu}	19.7	20.5	23.3	24.5	26.9	28.0	29.7	30.7	33.1	34.2	36.1
T_{save_ov}	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	1.0	1.0
T_{save}	20.7	21.5	24.3	25.5	27.9	29.0	30.7	31.7	34.1	35.3	37.2

T_{save_icap}, T_{save_cpu}, and T_{save_ov}, are the total, ICAP, CPU, and overhead execution times, respectively. T_{save} includes the times for unprotecting and protecting the PRR for the general case of HW multitasking as in Fig. 1c).

Table III summarizes the execution times for restoring the HW task's state $T_{rest} = T_{rest\_icap} + T_{rest\_cpu} + T_{rest\_ov}$, using configuration prefetching. T_{rest}, T_{rest_icap}, T_{rest_cpu}, and T_{rest_ov}, are the total, ICAP, CPU, and overhead execution times, respectively. Our measured time for toggling GSR to restore the task's state was 26.5 µs (part of T_{rest_cpu}), which is independent of the PRR's size or resources. T_{rest} includes the times for unprotecting and protecting the PRR for the case of HW multitasking as in Fig. 1c).

The execution time for bitstream manipulations, which can be performed while the tasks are executing in the PRRs, ranges from 23 ms to 37 ms for the smallest to the largest PRR in our experiments, respectively. All PRRs use all 160 FFs in every CLB column and one BRAM (36 Kbits).

VI. CONCLUSIONS

We presented, to the best of our knowledge, the first physical implementation of preemptive hardware (HW) multitasking for partially reconfigurable (PR) field-programmable gate arrays (FPGAs) to enable configuration prefetching and reuse. These techniques effectively hide/reduce configuration overheads, which improves PR system performance as compared to a system without these techniques. Experimental results evaluated our approach on a Xilinx Virtex-5, but the results and fundamental techniques can easily be extended for newer device families and for non-preemptive HW multitasking (Section III). Configuration prefetching and reuse hides reconfiguration overheads down to the order of milliseconds as compared to not overlapping task execution with downloading the next scheduled task's bitstream. Results also showed that configuration prefetching and reuse only required an average of 26.5 µs to begin execution of a previously-downloaded HW task's bitstreams, irrespective of the PR region (PRR) size, compared to several milliseconds. Future work includes incorporating configuration prefetching and reuse with a run-time reconfiguration scheduler for HW multitasking on a PR FPGA running an embedded Linux operating system in order to help system designers fully leverage our techniques.

Table III. EXECUTION TIMES (ms) FOR HW TASK STATE RESTORATION (T_{rest})

	HW task FFs										
Time	160	320	480	640	800	960	1120	1280	1440	1600	1760
T_{rest_icap}	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.2	0.2	0.3
T_{rest_cpu}	5.9	6.8	7.7	8.8	9.6	10.4	11.5	12.3	13.1	13.9	14.8
T_{rest_ov}	0.1	0.1	0.1	0.2	0.2	0.3	0.4	0.5	0.6	0.8	1.1
T_{rest}	6.1	7.0	7.9	9.1	10.0	10.9	12.1	13.0	13.9	14.9	16.2

ACKNOWLEDGMENTS

This work was supported by Fondo para la Innovación, la Ciencia y la Tecnología (FINCyT), Perú, under contract N° 121-2009-FINCyT-BDE, by the I/UCRC Program of the National Science Foundation (NSF) under grants EEC-0642422 and IIP-1161022, and the NSF CHREC membership support of Draper Laboratory. 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, and the tools provided by Xilinx.

REFERENCES

[1] M. Bassiri, and H. Shahhoseini, "Configuration reusing in on-line task scheduling for reconfigurable computing systems," *Journal of Computer Science and Technology*, Vol. 26, No. 3, pp. 463-473, 2011.

[2] G. Charitopoulos, I. Koidis, and K. Papadimitriou, "Hardware task scheduling for partially reconfigurable FPGAs," in *Proc. of the 11th Int'l Symp. on Applied Reconfigurable Computing (ARC'15)*, LNCS, Vol. 9040, pp. 487-498, 2015.

[3] J. A. Clemente, J. Resano, C. Gonzáles, and D. Mozos, "A hardware implementation of a run-time scheduler for reconfigurable systems," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol. 19, No. 7, pp. 1263-1276, 2011.

[4] M. Happe, A. Traber, and A. Keller, "Preemptive hardware multitasking in ReconOS," in *Proc. of the 11th Int'l Symp. on Applied Reconfigurable Computing (ARC'15)*, LNCS, Vol. 9040, pp. 79-90, 2015.

[5] K. Jozwik, H. Tomiyama, M. Edahiro, S. Honda, and H. Takada, "Comparison of preemption schemes for partially reconfigurable FPGAs," *IEEE Embedded Systems Letters*, Vol. 4, No. 2, pp. 45-48, 2012.

[6] H. Kalte and M Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," in *Proc. of the Int'l Conf. on Field Programmable Logic and Applications (FPL'05)*, pp. 223-228, 2005.

[7] A. Morales-Villanueva and A. Gordon-Ross, "On-chip context save and restore of hardware tasks on partially reconfigurable FPGAs," in *IEEE Int'l Symp. on Field Programmable Custom Computing Machines (FCCM'13)*, pp. 61-64, 2013.

[8] A. Morales-Villanueva and A. Gordon-Ross, "HTR: on-chip hardware task relocation for partially reconfigurable FPGAs," in *Proc. of the 9th Int'l Symp. on Applied Reconfigurable Computing (ARC'13)*, LNCS, Vol. 7806, pp. 185-196, 2013.

[9] J. Resano, D. Mozos, F. Catthoor, "A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware," in *Design, Automation and Test in Europe Conference and Exhibition (DATE'05)*, pp. 106-111, 2005.

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

[11] Xilinx, Virtex-5 FPGA User Guide v5.4 (UG190), March 16, 2012.

[12] Xilinx, Partial Reconfiguration User Guide v14.3 (UG702), October 16, 2012.