Evaluating Partial Reconfiguration for Embedded FPGA Applications

Ross Hymel, Alan D. George, and Herman Lam
{hymel, george, lam}@chrec.org
NSF Center for High-Performance Reconfigurable Computing (CHREC), University of Florida

Abstract—**Recent advances in Xilinx's FPGA hardware and commercial software design tools, spurred in large part by the DOD's Joint Tactical Radio System initiative, offer the possibility of incorporating dynamic partial reconfiguration (PR) into high-performance, embedded systems outside of academic research laboratories. PR can provide the flexibility and run-time reconfigurability that no pure hardware or software solution can offer. By multiplexing the hardware resources of a single programmable device with time-independent tasks, a common architecture in DOD systems, a single FPGA can handle the same processing workload as a multi-device equivalent. This paper analyzes the performance impact of using PR to perform remote updating, an important capability often used in embedded applications.**

I. Introduction

Generically, an SRAM-based FPGA is a multiprocessing device in that multiple, user-defined hardware modules can operate in parallel and independently within the same chip. One of the great advantages of such a device is the ability to modify its configuration memory easily and at any time. PR enhances this paradigm by reconfiguring only a portion of the chip's configuration memory, allowing the user to load and unload these functional hardware modules without interrupting or resetting the rest of the device. Despite this advantage, commercial interest in PR has never materialized due mainly to a lack of supporting software tools and merciless design flows. Nevertheless, different academic approaches have been developed to incorporate PR into embedded systems using the Virtex-II FPGA [1-2]. Recently, however, the release of the Virtex-4 and Virtex-5 series of FPGAs, with their tile-based frame architectures, coupled with the lucrative software-defined radio market, has pushed Xilinx to engineer a workable PR design flow [3]. While still unreleased to the general public, the new design flow eliminates many of the burdensome requirements put in place by the previous flow [4] and now supports the Virtex-4 (thought not yet the Virtex-5).

Unfortunately, due to the relatively recent unveiling of this new design flow, as well as the still restricted nature of its release, there exists a vacuum in research and results exploring high-performance PR systems targeting these new devices. In response, we present a study of the performance impact (timing, resource utilization, and other metrics) of the new design flow when targeting Virtex-4 FPGAs, with remote updating, an important usage of PR, as a platform for analysis.

II. Target Application

Although commercial FPGAs have enjoyed great success as development and testing platforms, their use in embedded applications has been limited due to their reduced flexibility after field-deployment and relative high cost. If an embedded FPGA's reconfigurable resources become static, the device turns into an expensive, power-hungry, low-performance ASIC. Thus, for FPGAs to become more practical as end-use devices, there needs to be a way to maintain true field-reprogrammability once deployed, i.e., the use of remote updating. Remote updating for FPGAs is the equivalent of in-application programming for microprocessors and is used to dynamically tailor the hardware to the application's needs in real-time.

Traditionally, an external configuration controller, usually a separate FPGA or microprocessor, performs remote updating. In the most generic sense, partial bitstreams that define hardware modules are sent to this device from a local or remote storage, over some type of communication link (e.g. MIL-STD-1553). The external controller then proceeds to fully reconfigure the user FPGA. This baseline approach has distinct advantages, namely that it provides an extremely flexible development environment since 100% of the user FPGA logic/routing resources are available for processing with no reconfiguration overhead or performance degradation.

Unfortunately, the need for an external controller presents undesirable drawbacks. Because the entire user FPGA is reconfigured, a full device bitstream must be transmitted over the communication link even if the designer only wishes to change a small portion of the design. This requirement results in a needlessly high data transfer, which is especially detrimental in bandwidth-limited applications, such as satellite payloads, where the update bitstreams may not be stored on-board. Furthermore, fully reconfiguring the user FPGA produces the longest possible reconfiguration period, translating into lost processing time.

A second drawback is the increased component count and PCB requirements necessary to accommodate an external device. Besides increasing the cost of the design, the increased complexity allows more failure points to exist in all phases of the system's lifetime (fabrication, assembly, testing, deployment, etc.). Most DOD designs are particularly affected since they must be qualified to strict environmental standards with regard to shock, vibration, ESD, etc.

In this paper, we describe an approach for configuration control in which we embed the controller within the user FPGA. By using the Internal Configuration Access Port (ICAP) to perform partial reconfiguration, the remote update is performed in-situ, eliminating the need for an external device. In addition to mitigating many of the disadvantages previously mentioned, there are many advantages inherent in this approach. Most importantly, unrelated processing can continue uninterrupted during partial device reconfiguration, automatically maintaining state information. The remainder of this paper analyzes the performance impact of incorporating remote updating into three permutations of a generic PR architecture targeting an XC4VLX25 FPGA.

III. Experimental architectures

In order to facilitate PR in real hardware with a commercially-available design flow, key design issues and trade-offs must be addressed, including the number of partially reconfigurable regions (PRRs), the PRR shape, size, and placement, the PRR's access to the global clock network and I/O pads, and the communication interface amongst different PRRs and the static portion of the design. A complete description of each experimental study will appear in the full presentation, while a condensed version appears here.

Each design permutation contains a static communication and configuration controller, as well as a different number of PRRs, ranging from one PRR of maximal size, to two side-by-side PRRs, to four PRRs arranged in a 2x2 fashion. Each of the regions has a generic black-box, top-level interface. The advantage of such an approach is that a designer can use any high- or low-level tool to synthesize the PRR, so long as the top-level interfaces match. Then the designer need only run an existing script that automatically handles the details of the PR design flow to generate the partial bitstreams.

We evaluated each design permutation using different high-performance computing cores, including Radix-4 FFT, AES, ARM7 soft-core processing, and others. We measured the minimum clock period at which each design could run twice, once when the design operated without any PR modifications and once after plugging into the experimental PR architecture. We also measured the size of the programming bitstream twice in the same fashion.

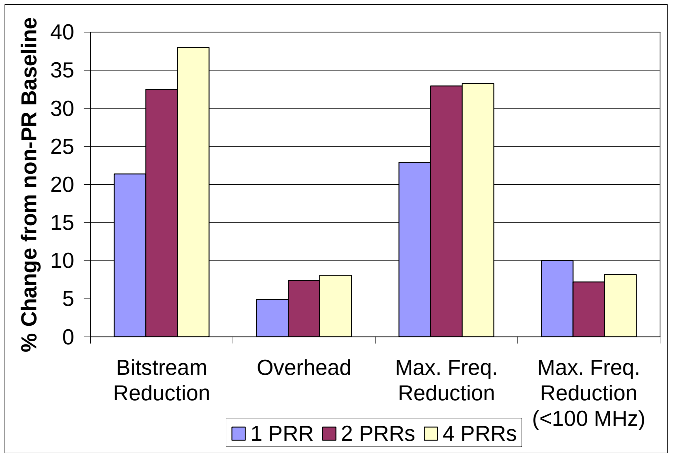

Figure 1: Measured Effects of PR vs. non-PR Baseline

Figure 1 displays a set of average measured PR performance effects, including the bitstream size reduction, the PR overhead of each design, and the decrease in maximum clock frequency due to PR. The PR overhead consists of resources that the FPGA uses to facilitate the design flow (e.g. bus macros) but that do not contribute to processing. The clock frequency numbers are split into two categories, one for all designs and one for designs that originally operated at less than 100 MHz. The discrepancy is due to a single enable net in the static region whose purpose is to put the PRRs into a known state during reconfiguration. This net is most often the critical path for designs over 100 MHz due to its length and fanout. In absolute terms, the results averaged across all design permutations are -162 KB, +727 slices, -57.6 MHz, and -8.09 MHz, respectively. In addition, the relative percentages should remain constant across different device sizes. The full presentation will include a detailed breakdown of these results.

IV. Conclusions

The use of partial reconfiguration in conjunction with commercial FPGAs and software tools can provide a reliable, resource-saving, and flexible means for updating the processing load of a deployed programmable device. By time-multiplexing the device, the designer has, in effect, an FPGA that contains more resources than are actually physically present, providing multiprocessing across both time and space. This method not only reduces the reconfiguration time but also the amount of bitstream data. Furthermore, using a generic architecture simplifies the design flow at the hardware level to allow rapid system development by designers untrained in the nuances of PR. These factors are especially important in DOD systems, as the generic hardware can be qualified to the necessary environmental standards and then reused in other platforms without knowledge of the low-level details.

Future directions for this work include exploring "full" partial reconfiguration. As Virtex-4 devices contain two separate ICAP primitives, we have the ability to reconfigure the reconfiguration engine itself by switching configuration control between different regions. Doing so would allow us to update the previously static controller, e.g., to change the encryption standard or the communication protocol it uses.

V. Acknowledgements

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge tools and equipment provided by Sandia National Laboratories and Xilinx that helped make this work possible.

VI. References

[1] M. Ullmann, B. Grimm, M. Hübner, and J. Becker, "An FPGA Run-Time System for Dynamical On-Demand Reconfiguration," *Proc. IEEE Parallel and Distributed Processing Symposium*, Santa Fe, NM, Apr. 26-30, 2004.

[2] M. Hübner, J. Becker, "Exploiting Dynamic and Partial Reconfiguration for FPGAs – Toolflow, Architecture, and System Integration," *Proc. 19th SBCCI Symp. on Integrated Circuits and Systems Design*, Ouro Preot, Brazil, 2006.

[3] Early Access Partial Reconfiguration User Guide, UG208 (v1.1), Xilinx Inc., Mar. 6, 2006.

[4] Two Flows for Partial Reconfiguration: Module Based or Difference Based, XAPP290 (v1.2), Xilinx Inc., Sept. 9, 2004.