PRML: A Modeling Language for Rapid Design Exploration of Partially Reconfigurable FPGAs

Rohit Kumar, Ann Gordon-Ross
{*kumar, ann*}@*chrec.org*
NSF Center for High-Performance Reconfigurable Computing
University of Florida, Gainesville, FL USA 32611

***Abstract*—Leveraging partial reconfiguration (PR) can improve system flexibility, cost, and performance/power/area tradeoffs over non-PR functionally-equivalent systems, however, realizing these benefits is challenging, time-consuming, and PR must be considered early during application design to reduce design exploration time and improve system quality. To facilitate realizing these benefits, we present an application design framework and an abstract modeling language for PR (PRML). By applying extensive PRML modeling guidelines to a complex arithmetic core, we show PRML's potential for efficient PR capability analysis, enabling designers to determine Pareto optimal systems during application formulation based on designer-specified area and performance metrics.**

***Keywords*-FPGA; partial reconfiguration; design space exploration**

I. Introduction and Motivation

In a partially reconfigurable (PR) field-programmable gate array (FPGA), reconfiguration is isolated to pre-defined reconfigurable regions, which enables runtime reconfiguration of hardware functionality without halting system execution. As compared to non-PR FPGAs, which must halt system execution for reconfiguration, PR provides many additional benefits, such as increased area and power reduction potential and facilitates system maintainability, adaptability, reliability, and fault-tolerance.

However, effectively leveraging PR is a challenging, time-consuming design task due to specialized PR design tools, flows, and manual design steps and, thus PR is not widely used. Designers must create the application's PR-architecture, which partitions the application into one or more reconfigurable modules, which run in the reconfigurable regions (reconfigurable modules can time-multiplex reconfigurable regions) and a static module, which runs in the static region (the static region is never reconfigured). Given the reconfigurable modules' resource requirements, the designer must also manually floorplan the FPGA to specify the reconfigurable regions' and static region's locations and area constraints. Additionally, given an intractable number of alternative PR-architectures and floorplans (we refer to each combination as a *PR-design point*) that provide different area/performance/power tradeoffs, designers require design exploration assistance. Currently no conventional methodology exists to automatically generate and analyze the PR-design point tradeoffs with respect to designer-specified application goals/requirements.

In this paper, we present a formulation-level framework for rapid PR design space exploration. Our framework uses a novel PR modeling language (PRML) to model application algorithms using specialized modeling blocks (PRML blocks) that represent generic algorithmic-level components (ALCs), such as loops, conditions, computations, memory reads/writes, etc. To evaluate an ALC's amenability to PR, our framework leverages FoRSE [1], an in-house formulation-level tradeoff analysis tool. To improve the tradeoff analysis accuracy, ALCs are augmented with per-ALC estimated attributes, such as device-specific area, performance, and/or power overheads. Since FoRSE does not physically implement each PR design point, FoRSE's execution time is minimal. Using a complex arithmetic core, we illustrate PRML's ability to accurately model an application and provide fast and accurate tradeoff analysis.

II. Related Work and Contributions

Several previous works [2][3] present application-formulation-level modeling, but few consider tradeoff analysis. Antola et al. [4] performed tradeoff analysis using an Impulse-C/CoDeveloper-based framework that analyzed feasible PR-architectures based on execution time, inter-module communication interface bit-width, and resource usage. However, Impulse-C is proprietary, which inhibits portability/usability, and the partitioning methodology and tradeoff analysis details were not provided. Shallenberg et al. [5] avoided using a proprietary language in OSSS+R by leveraging SystemC. OSSS+R provided custom SystemC constructs to model PR. Since OSSS+R did not generate PR-architecture(s) from a non-PR application, OSSS+R could not perform tradeoff analysis of an application. Ponpandi et al. [6] performed tradeoff analysis on an application's netlist. However, since a netlist description contains fine-grain application and target device specifications, the designer had no control over partitioning or tradeoff analysis. In addition to academic tools [4][5][2][3][6], few industrial tools, such as Labview

and Simulink provide algorithmic-level modeling via modeling blocks but these tools do not explore application's PR design space.

In our work, we leverage algorithm-level modeling languages to mitigate portability/usability limitations and eliminate physical implementation using PRML. PRML provides an accurate and flexible method for modeling application behavior using both fine- and coarse-grained modeling to represent primitive operations (e.g., Boolean) or a large set of operations with a single PRML block, respectively. PRML uses interdependence modeling to model and consider inter-module communication interface requirements and overheads.

III. PRML Algorithm Modeling

To provide simple modeling guidelines and accurate ALC representation, PRML uses PR-compliant and PR-resistant blocks, which represent ALCs that are amenable to PR (e.g., arithmetic and logical operations, loops, macros, conditional forks) and not amenable to PR (e.g., off-chip memory, input/output (I/O) devices, joins, syncs, waits), respectively. PR-compliant blocks include computation blocks for modeling computations, iteration blocks for modeling repetitive computations, memory blocks for modeling memory operations, hierarchy blocks for modeling a hierarchy or group of operations, and choice blocks for modeling conditions that are triggered by one or more PR-compliant block(s). PR-resistant blocks model inter-ALC dependence relationships and create block interconnections using data and control arcs. A data arc block denotes a data dependence and control arc block denotes a control dependence. Additionally, PR-resistant blocks model dependence operations, such as dependence or-merge, and-merge, and split/clone.

PRML extends the PR-compliant and PR-resistant blocks' modeling capabilities with per-block/arc attributes, which enable designers to easily incorporate an application's architectural and implementation characteristics (e.g., resource requirements and worst case delay) in the PRML model. Since an ALC can be implemented in several alternative ways on an FPGA, a PR-compliant block's resource and latency attributes can assume multiple values. PRML leverages the attribute's ability to uniquely identify a block's architectural and implementation characteristics to perform partitioning and tradeoff analysis.

IV. Partitioning and Tradeoff Analysis

Our framework performs an application's PR design space exploration using two phases: the partitioning phase, which generates feasible PR-architectures, and the tradeoff analysis phase, which evaluates these PR-architectures.

Partitioning Phase: The partitioning phase uses partitioning rules to systematically analyze the PRML model to generate the PR-architecture(s). The partitioning rules leverage graph-theory concepts and the per-block/arc attributes to logically segregate common and mutually exclusive ALCs and generate sets of ALC combinations (i.e., the PR-architectures).

A PRML model is a directed graph of PRML blocks. An execution path/cycle is a directed path/cycle between any pair of memory blocks. Trivial paths do not contain any computation or iteration blocks and do not show resource or latency overheads. A path X is a parent/child of a path Y if all blocks and arcs of path X are a proper superset/subset of path Y. Two paths are siblings/clones if the paths have the same parent/terminal blocks. A path's uncles are the path's parent's siblings or parent's clones. A path's cousins are a path's uncle's children. A weak data/control component is a maximal weakly connected component where all arcs are data/control arcs. An iteration/computation superblock represents a weak data component that contains at least two blocks and at least one/all block(s) are iteration/computation blocks. To exemplify the partitioning phase, we partition a complex arithmetic core that implements a set of arithmetic operations (add, subtract, multiply, divide, square root) for a complex number. Since describing all partitioning rules and the rules' effects on the PRML model is infeasible due to space constraints (there are 390 different PR-architectures for the core), Table I depicts the fundamental partitioning rules, in the order that the rules are executed, and a brief description of the rules' execution results.

Tradeoff Analysis Phase: Tradeoff analysis takes as input a PR-architecture, as generated by the partitioning phase, and evaluates these PR-architectures based on the area, performance, and inter-reconfigurable region communication overheads according to the block's attributes.

To determine the complex arithmetic core's per-ALC resource and delay attribute values, we developed and implemented the core using Xilinx ISE 13.2 and used FoRSE to perform tradeoff analysis. Although implementation of all reconfigurable modules generated in the partitioning phase (rule 6 in Table I) was necessary to obtain area and delay values, FoRSE does not require a PR-architecture's floorplans' physical implementation. FoRSE determines the Pareto optimal set of floorplans and target devices for a PR-architecture using high-level formulations and mathematical models of the target devices' resource organizations and, thus, affords fast design space exploration time.

To generate the Pareto optimal set of PR-architectures and target devices, FoRSE leverages several metrics, such as expected resource savings, actual resource savings, and PR overhead. Expected and actual resource savings are the percentage of saved resources that the designer expects to attain and actually achieves, respectively, by using a PR-architecture as compared to a non-PR functionally-equivalent architecture. PR overhead is the difference between the expected and actual resource savings.

Table I: Fundamental partitioning rules and brief description of the rules' execution results after the rule is applied to a PRML model.

Fundamental Partitioning rules	**Execution results**
1. Eliminate hierarchy blocks and memory blocks inside the hierarchy blocks.	Eliminates redundant memory blocks by flattening the PRML model.
2. Identify computation and iteration superblock(s).	Reduces the number of blocks by merging interdependent blocks.
3. Identify all execution paths/cycles (L1 paths) except data paths/cycles and trivial paths.	L1 paths identify input-to-output paths that show non-zero resource/delay overheads and include all control choices.
4. Identify distinct smaller paths (L2 paths) from L1 paths (sequentially break L1 paths at choice and or-merge blocks but exclude data paths).	L2 paths identify smaller data paths from the non-trivial input-to-output paths based on control choices.
5. Identify distinct smaller paths (L3 paths) from L2 paths (break L2 paths at iteration blocks and iteration superblocks).	L3 paths identify small computation kernels.
6. Identify all sets of static module and reconfigurable modules based on L2 paths, L3 paths, and the block's divergent attribute value.	Identifies all possible path combinations by merging the paths generated by rules 3-5, and identifies clone, sibling, and cousin paths from these merged paths. Clone, sibling, and cousin paths represent reconfigurable modules.
7. Assign reconfigurable modules to reconfigurable regions: (*a*) clone reconfigurable modules are assigned to the same reconfigurable region; (*b*) sibling reconfigurable modules are assigned to different reconfigurable regions; (*c*) cousin reconfigurable modules can be assigned to the same or different reconfigurable regions.	Calculates the number of reconfigurable regions required for each combination generated by rule 6 and creates all possible reconfigurable module-to-reconfigurable region assignments.
8. Create PR-architectures.	Different PR-architectures are created for each reconfigurable module variant and each reconfigurable module-to-reconfigurable region assignment.

Additionally, we augmented FoRSE to include longest path delay analysis for each floorplan.

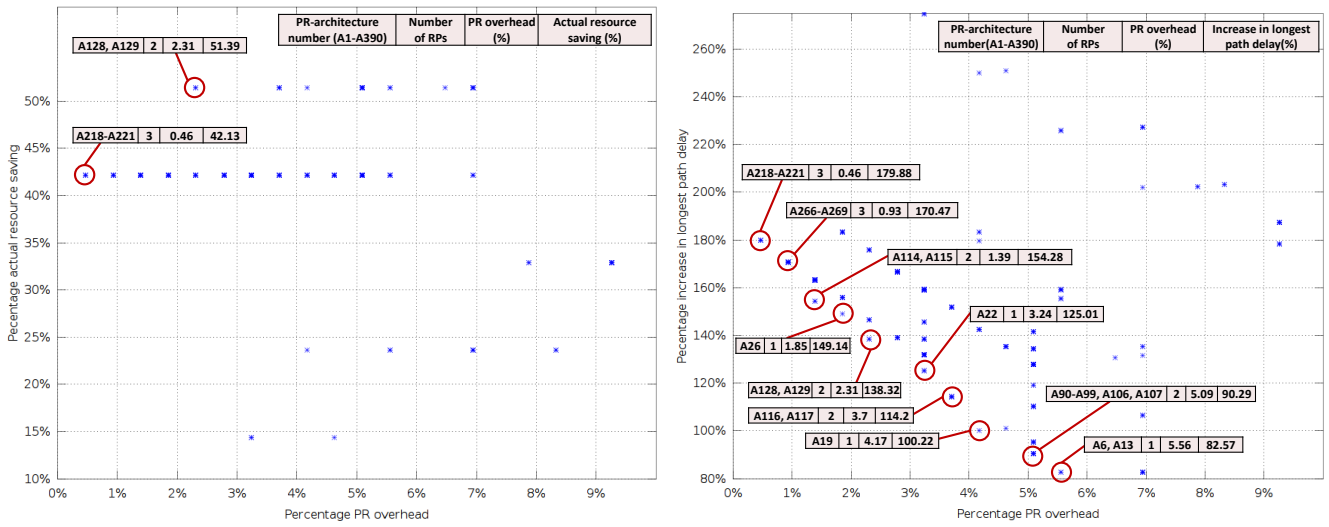
V. Experimental Results

We demonstrate the efficacy of PRML's partitioning and tradeoff analysis using a complex arithmetic core. We created the core's PRML model with the yED tool, which is an extended markup language (XML)-based diagram editor. We developed Python scripts and used the NetworkX library to parse and partition the core's PRML model to generate the PR-architectures. We leveraged our enhanced version of FoRSE to perform tradeoff analysis of these PR-architectures. To show FoRSE's efficacy, we considered the actual resource savings, longest path delay, and the PR overhead as the comparison metrics. We determined the PR overheads in terms of configuration frames using the Xilinx V5LX100T FPGA.

A Python script applied the partitioning rules (Table I) o generate all feasible PR-architectures. The partitioning rules generated only PR-architectures with one, two, or three reconfigurable regions. Rule 6 revealed that there exists only one set of clone paths and thus, rule 7(a) produced PR-architectures with one reconfigurable region. Additionally, rule 6 revealed that there exists two or three children for a path and thus, rules 7(a) and 7(b) produced PR-architectures with two or three reconfigurable regions, respectively. The partitioning generated 390 PR-architectures. To concisely reference individual PR-architectures during tradeoff analysis, we number the PR-architectures A1 through A390.

Fig. 1a and Fig. 1b depict the tradeoff between the PR-architectures' actual resource savings and longest path delays, respectively, and the PR overheads, with the Pareto optimal PR-architectures circled and the numbers of reconfigurable regions, PR overheads, and actual resource savings or increase in longest path delay in the adjacent rectangles. Since these metric values depend on the floorplan and the floorplans reconfigurable module-to-reconfigurable region assignments (rule 6 and 7 in Table I) and different PR-architectures may have similar floorplans, single points may represent multiple PR-architectures. In Fig. 1a, PR-architectures A128 and A129 show the highest actual resource savings due to mapping a large number of reconfigurable modules into two small reconfigurable regions as compared to one larger reconfigurable region or three smaller reconfigurable regions. Since the difference between the sum of the reconfigurable modules' resource requirements and the floorplans resource requirements is lowest for A218-A221, these PR-architectures show the lowest PR overhead, but since these PR-architectures' floorplans use three reconfigurable regions, the actual resource savings is low. In Fig. 1b, PR-architectures A6 and A13 have the smallest increase in longest path delay because these PR-architectures have only one reconfigurable region and the longest path delay increases as the number of reconfigurable regions increases, but these PR-architectures have high PR overheads. Similar arguments justify the remainder of the Pareto optimal PR-architectures' metric values.

Since the PR overheads are typically small as compared to the actual resource savings and the increase in the longest path delay, designers can select either a Pareto optimal PR-architecture with high actual resource savings to fit a small target device or a Pareto optimal PR-architecture with lower longest path delay for higher performance. Thus, this tradeoff analysis enables designers to carefully



(a) (b)

Figure 1: a) Percentage of actual resource savings with respect to PR overhead for each PR-architecture; b) Percentage increase in longest path delay with respect to PR overhead for each PR-architecture.

select an application's most suitable PR-architectures based on designer-specified goals during application formulation, thereby pruning the design space and reducing design exploration time while improving the system's ability to adhere to designer-specified goals.

VI. Conclusions and Future Work

In this paper, we present a framework for modeling an applications PR amenability and performing PR design space exploration. To enable rapid modeling and accurate exploration, our framework uses a partial reconfiguration modeling language (PRML) and formulation-level tradeoff analysis [1]. To show our framework's capability for application-formulation-level modeling and rapid design space exploration, we apply our framework to a complex arithmetic core and showed that the tradeoff analysis can aid designers in selecting a PR-architecture for implementation based on a designer's goals, such as a small target device or high performance. Future work includes partitioning enhancements using an iterative process that incorporates feedback from tradeoff analysis and the PR application's runtime performance/throughput to repartition the application.

VII. Acknowledgments

This work was supported 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. We gratefully acknowledge tools provided by Xilinx.

References

[1] R. Kumar and A. Gordon-Ross, "Formulation-level design space exploration for partially reconfigurable fpgas," in *Field-Programmable Technology (FPT), 2011 International Conference on*, Dec., pp. 1–6.

[2] N. Abel, "Design and Implementation of an Object-Oriented Framework for Dynamic Partial Reconfiguration," *International Conference on Field Programmable Logic and Applications*, pp. 240–243, Aug. 2010.

[3] R. Ahmed and P. Hallschmid, "Modeling and Evaluation of Dynamic Partial Reconfigurable Datapaths for FPGA-Based Systems Using Stochastic Networks," *21st International Conference on Field Programmable Logic and Applications*, pp. 70–75, Sep. 2011.

[4] A. Antola, M. Santambrogio, M. Fracassi, P. Gotti, and C. Sandionigi, "A novel hardware/software codesign methodology based on dynamic reconfiguration with ImpulseC and CoDeveloper," in *3rd Southern Conference on Programmable Logic*, 2007, pp. 221–224.

[5] A. Schallenberg, W. Nebel, A. Herrholz, P. A. Hartmann, and F. Oppenheimer, "OSSS+ R: A framework for application level modelling and synthesis of reconfigurable systems," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, 2009, pp. 970–975.

[6] S. D. Ponpandi and A. Tyagi, "Partial reconfiguration logic synthesis by temporal slicing," in *International Conference on Field-Programmable Technology (FPT)*, 2011, pp. 1–6.