DESIGN AUTOMATION FOR PARTIALLY RECONFIGURABLE FPGAS: DESIGN FLOWS, TOOLS AND ARCHITECTURES

By

SHAON YOUSUF

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2015

To my family and friends

ACKNOWLEDGMENTS

Firstly, I'm grateful to Allah, the most beneficial and merciful, in whose worship I have always found respite from many stressful and difficult times I have endured during my time in graduate school.

I would also like to express my deepest gratitude to Dr. Ann Gordon-Ross, my Ph.D. advisor, for her continuous support and guidance in my research. She has always encouraged me and believed in me, and pushed me hard to achieve my potential. I will always be indebted to her.

Special thanks to my Ph.D. committee members Dr. Alan George, Dr. Herman Lam, and Dr. Khandaker Muttalib for their feedback and guidance in my research.

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. I also, gratefully acknowledge the tools provided by Xilinx.

TABLE OF CONTENTS

page

LIST OF TABLES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

DESIGN AUTOMATION FOR PARTIALLY RECONFIGURABLE FPGAS: DESIGN FLOWS, TOOLS AND ARCHITECTURES

By

Shaon Yousuf

December 2015

Chair: Ann Gordon-Ross
Major: Electrical and Computer Engineering

Dynamic partial reconfiguration (PR) enhances traditional FPGA-based high-performance embedded reconfigurable computing by providing additional benefits such as reduced area and memory requirements, increased performance, and increased functionality. However, leveraging PR benefits requires specific designer expertise, which increases design time, and thus PR has not yet gained widespread usage. Even though Xilinx's PR design flow significantly eases PR design [Xilinx UG 702], to fully leverage PR benefits designers require extensive PR design flow knowledge, as well as low-level architectural details of the target FPGA device. Fully leveraging PR benefits, even with extensive PR design flow knowledge, is challenging and requires minute attention to details since even small changes in the PR design flow process can result in PR designs with degraded performance as compared to a non-PR design. The two most performance-critical steps in the PR design flow are PR design partitioning and PR design floorplanning. These steps involve key decisions that affect large tradeoffs between performance parameters, such as reconfiguration time, resource requirements, power consumption, speed, memory requirements, etc., resulting in a large Pareto-optimal design space. Thus, choosing a PR design from this design space that balances

these performance parameters according to system designer needs can be a very challenging and cumbersome process.

This dissertation presents the design automation for partial reconfiguration (DAPR) design flow, which aids system designers during PR design partitioning and PR design floorplanning to quickly and easily generate and analyze PR designs with different performance parameters, and choose a PR design that most closely satisfies system designer goals. Our PR design partitioning methodology is tailored towards hardware/software (HW/SW) co-designed adaptive applications and leverages an exhaustive search algorithm to generate a Pareto-optimal set of HW/SW PR design partitions that tradeoff FPGA resource requirements and reconfiguration time. For performance evaluation, the corresponding total execution time for each HW/SW PR design partition is also reported. Our PR design floorplanning methodology heuristically explores the PR floorplanning design space for a system-designer chosen HW/SW PR design partition using a modified-simulated annealing based floorplanner and works towards improving the PR design's speed clock frequency. Our floorplanner iteratively improves the clock frequency of the PR design and quickly converges on the design with the greatest improvements during first several iterations (12-20 iterations on average), thus requiring very little design exploration time. The floorplanner also generates a Pareto-optimal set of PR design floorplans that tradeoff clock frequency and average partial bitstream size, which affects the reconfiguration time of that design.

The main goal of the DAPR design flow is to isolate PR designers from low-level design complexities involved during the PR design process, which reduces the manual

design time effort spent by a PR designers and thus, making PR more accessible and amenable to a larger range of designers.

CHAPTER 1
INTRODUCTION

Dynamic partial reconfiguration (PR) in modern dynamically reconfigurable field-programmable gate (FPGAs) enable run-time reconfiguration of selected FPGA device fabric partitions while the rest of the device keeps executing. This run-time reconfigurability can be leveraged to design embedded systems that time-multiplex mutually exclusive application functionality to provide benefits such as reduced hardware, power, and memory requirements as compared to a traditional application specific integrated chip (ASICs)-based embedded systems, which provide all application functionality simultaneously. Incorporating PR into FPGA-based systems adds additional PR benefits that can be leveraged to build low-cost, high performance embedded systems, which is typically the goal for embedded system designers. Thus, PR-capable FPGAs have become an attractive alternative platform for embedded system design as compared to fixed-functionality ASICs. PR enables applications that do not simultaneously require all functionality to time-multiplex the FPGA hardware. Some applications that can benefit from PR include cryptographic algorithms [Gonzalez et al. 2005], software defined radios [McDonald 2008], direct cosine transform (DCT) signal processing [Huang et al. 2008], JPEG image processing [Yousuf et al. 2009], MPEG video processing [Hentati et al. 2009], adaptive fault tolerant systems [Yousuf et al. 2011], wireless video receivers [Vipin et al. 2013] etc.

Cutting-edge PR-capable FPGAs can also include processor(s) that potentially expand PR benefits by enabling specific application functionalities that are amenable to software to be mapped to software routines running on the processor. This added ability can reduce hardware requirements further. However, leveraging PR benefits effectively

while designing these hybrid systems is complex and a well-known hardware/software (HW/SW) co-design problem.

One of the most challenging and performance critical steps during HW/SW co-design of PR systems is HW/SW PR partitioning. PR system design for an application is modularized in nature [Xilinx UG 702]. An application's functionality can be represented as a set of modules, where each module represents an application's task, where all tasks collectively perform the complete application's functionality. HW/SW PR partitioning divides these tasks to run on the FPGA hardware and the FPGA software processor, and further maps the hardware tasks to run in one or multiple partially reconfigurable regions (PRRs). PRRs can be loaded/loaded with different, mutually exclusive hardware tasks during runtime to time-multiplex hardware resources.

Figure 1 shows two examples of HW/SW PR partitions for an application with five tasks T1 to T5. In partition A, tasks T1 and T2 are partitioned to run on the FPGA hardware and tasks T3, T4, and T5 are partitioned to run on the FPGA software processor. Tasks T1 and T2 are further mapped to run in two individual PRRs, PRR 1 and PRR 2, respectively. In partition B, tasks T3, T4, and T5 are partitioned to run on the FPGA hardware and tasks T1 and T2 are partitioned to run on the FPGA software processor. Tasks T3, T4, and T5 are further mapped to run on a single PRR, PRR 1. Comparing partition A and partition B and assuming all tasks require the same FPGA hardware resources (which is sufficient for this illustration), partition A requires more hardware resources since more PRRs are assigned to partition A, and thus increases the overall system cost. Alternatively, partition B requires less hardware resources since there is only one PRR assigned, however, partition B will require more reconfiguration

time because the FPGA needs to be reconfigured each time a different task (i.e., T3, T4, or T5) is required to execute and is not already loaded into the PRR. This imposes a potentially large number of FPGA reconfigurations, which can incur significant performance overheads on a PR system since reconfiguring the FPGA device fabric requires extra reconfiguration/execution time (in addition to the execution time of the actual task) and increases the overall execution time of the application. Even though these examples only show two potential PR partitions, for a given application, there exists a large number of functionally-equivalent HW/SW PR partitions with different performance and reconfiguration times, and choosing the PR partitioning that most closely meets a PR system designer's goals requires manually exploring all of these HW/SW partitions. Given this very large design space, this process is difficult, cumbersome, and requires a large design time effort.

Once a HW/SW PR partition that meets system designer goals is determined—the *chosen* HW/SW PR partition—the next performance critical step involved in HW/SW co-design is PR design floorplanning. PR design floorplanning assigns physical FPGA resources to the chosen HW/SW PR partition's PRRs, and also determines the specialized hardwired communication interfaces—partition pins [Xilinx UG 702])—surrounding the PRRs boundaries. PRR resources are assigned by defining area constraints that specify the PRR's dimensions (size and shape) in terms of an XY coordinate system [Xilinx UG702 2013] on the FPGA. Partition pin resources are assigned by defining area constraint set as XY coordinate locations around the PRR boundaries. Given a device's resource layout and the chosen HW/SW PR partition's

PRRs' resource requirements, there are many different functionally-equivalent PR design floorplans, resulting in a very large PRR and partition pin design space.

As an example, assuming partition A was the chosen HW/SW PR partition, Figure 1-2 shows examples of two PR design floorplans (floorplan A and floorplan B) that could be generated from the PR design floorplanning process for partition A. PR design floorplanning requires carefully choosing a PR design floorplan from a very large PRR and partition pin design space, since typically PR design floorplans with smaller total wire length usually result in higher clock frequency, thus higher performance designs. A visual comparison of floorplan A and floorplan B in Figure 1-2 shows that floorplan A requires smaller total wire length as compared to floorplan B, and thus will typically result in a higher clock frequency design. However, there are several other key factors that can also affect the clock frequency of a PR design floorplan, such as the location of the FPGA clocking resources, distance from boundary input/output (I/O) interfaces, and the heterogeneity of FPGA resources. Currently there are no concrete guidelines for choosing a PR design floorplanning given these considerations, thus this processes is challenging and cumbersome for PR system designers. PRR and partition pin floorplans must be manually investigated to determine high clock frequency PR designs. PR system designers manually investigate PRR and partition pin floorplans by exploring the PRR and partition pin design spaces for different PRR and partition pin floorplans, respectively. This manual design space exploration involved during the HW/SW PR partitioning and PR design floorplanning process hinders PR design productivity and can potentially discourage embedded system designers from leveraging PR benefits.

Currently, PR designers have little automated PR design exploration support when performing HW/SW PR design partitioning and PR design floorplanning. Most existing work [Srinivasan et al. 1998][Mei et al. 2000][Byungil et al. 2000][Arato et al. 2013] that considers HW/SW co-design presents various heuristics to optimally determine a HW/SW partition but do not consider PR systems. Most existing work [Banerjee et al. 2007][Banerjee et al. 2009][Craven 2008][Vipin et al. 2012][Rabozzi 2014][Montone et al. 2010][Singhal et al. 2006][Cheng et al. 2004][Feng et al. 2006] that considers PR design floorplanning presents various heuristics to effectively explore the PRR floorplanning design space, but do not consider exploring the partition pin floorplan design space. Furthermore, these works presented HW/SW PR design partitioning and PR design floorplanning techniques independently, providing only partial solutions, and were not integrated into a holistic PR design flow. To the best of our knowledge, there exists no previous effort to completely automate PR design exploration holistically considering both HW/SW PR design partitioning and PR design floorplanning.

Thus, to alleviate PR designer efforts and make PR system benefits more easily attainable, in this thesis, we present the design automation for partial reconfiguration (DAPR) design flow. The DAPR design flow *automatically* performs HW/SW PR design partitioning and PR design floorplanning for a HW/SW co-designed system. The DAPR design flow surpasses traditional PR design flow using automation that eliminates the need for a PR system designer to be familiar with low-level FPGA device details and complexities, and thus reduces the manual design time effort spent by a PR designer, making PR more amenable to a larger range of designers. The DAPR design flow's HW/SW PR partitioning methodology quickly outputs a Pareto-optimal set of HW/SW

PR design partitions that tradeoff hardware resource requirements and reconfiguration time. The corresponding total system execution time is also reported. Our results show that for an example design with 14 tasks, the DAPR design flow's HW/SW partitioning methodology can generate a Pareto-optimal set of HW/SW PR designs in under 10 minutes running on an Intel® Core ™ 7 2.5 GHz CPU with 8 GB of RAM. From this Pareto-optimal set, a system designer can select a *chosen* HW/SW PR design partition that most closely adheres to the desired goals for further floorplanning design space exploration. DAPR design flow's PR design floorplanning methodology performs automated PR design space exploration to improve the clock frequency of a PR design by iteratively exploring the PR design space using a modified simulated annealing (SA)-based algorithm. We improved and tailored the traditional simulated annealing algorithm perturbation function specifically for PR design floorplanning by adding a specialized perturbation function that changes the PR design floorplan starting locations once every few iterations to more accurately explore the PR design space. DAPR's floorplanning results showed that the greatest clock frequency improvements are achieved during first several iterations (12-20 iterations on average).

The reminder of the document is organized as follows: Chapter 2 provides a background on the current vendor support PR design flow and prior work related to PR design flow automation. Chapter 3 presents our proposed DAPR design flow and describes a flexible and scalable target architecture suitable to implement the DAPR design flow's output bitstreams. Chapter 4 describes our HW/SW PR partitioning methodology and evaluates our methodology on a real world application. Chapter 5 describes our automated floorplanning algorithm, along with the design space

considerations and evaluates the floorplanning algorithm on real world applications. Finally, Chapter 6 presents the conclusions of our work and outlines directions for possible future research.

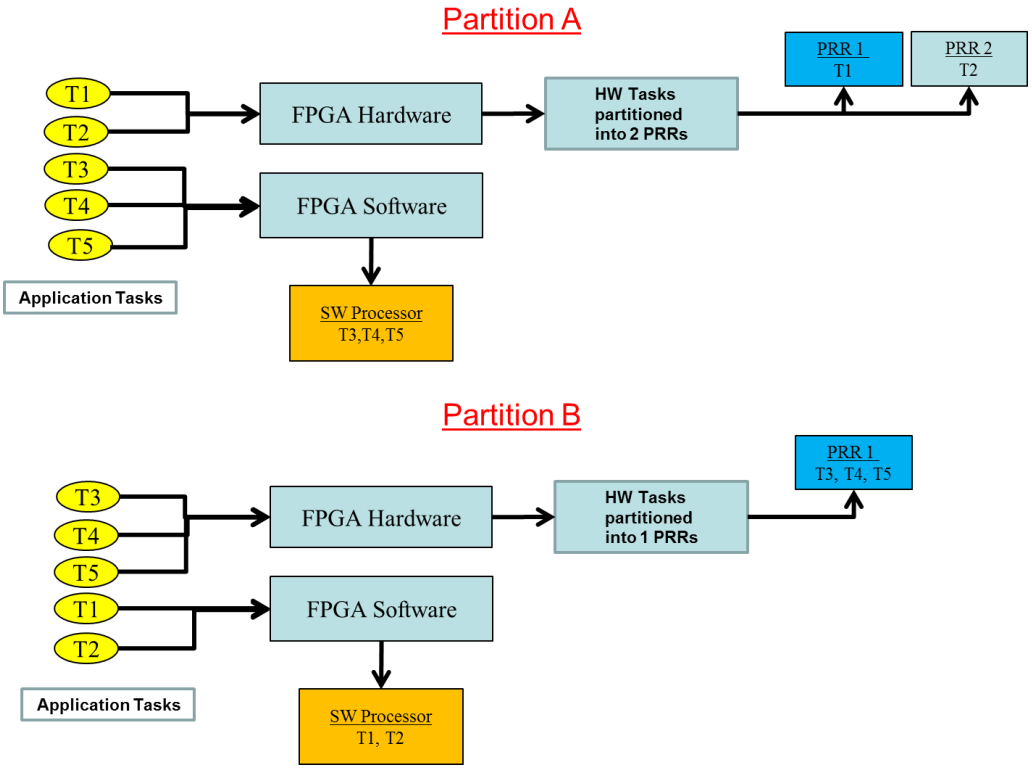

Figure 1-1. Two example HW/SW PR partition for an application with five tasks. In partition A, tasks T1 and T2 are mapped to FPGA hardware, and tasks T3, T4, and T5 are mapped to FPGA software. In partition B, tasks T3, T4, and T5 are mapped to FPGA hardware, and tasks T1 and T2 are mapped to FPGA software

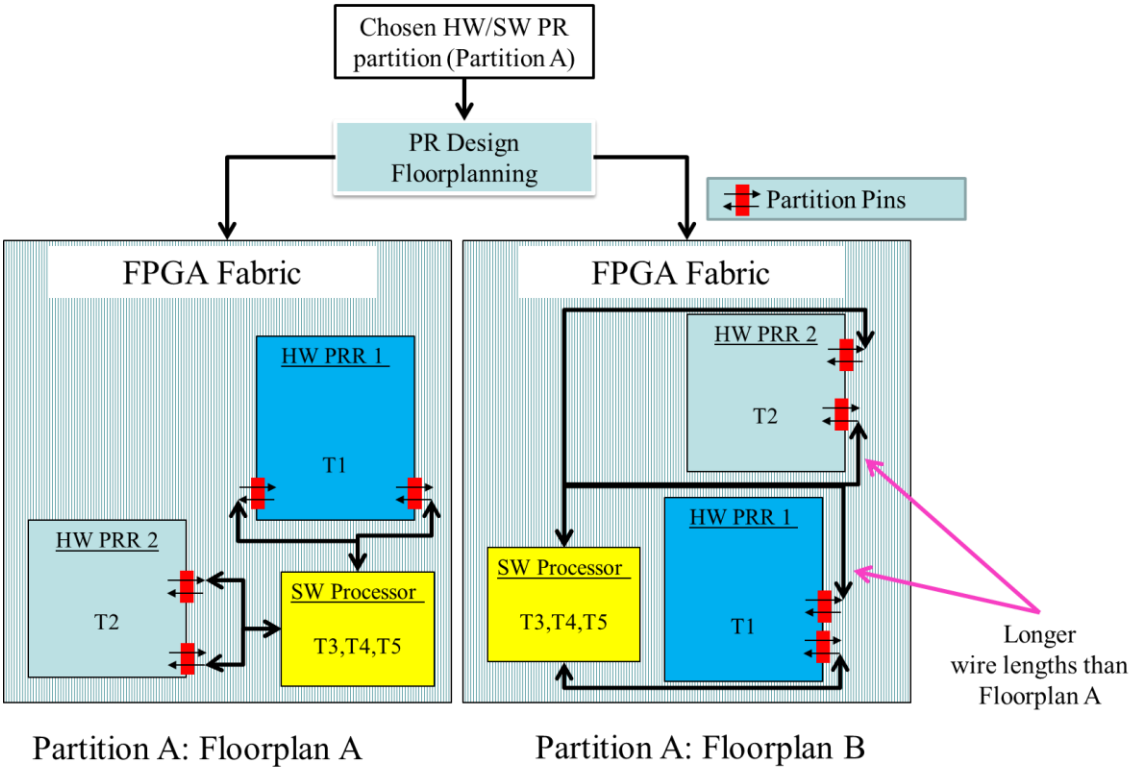

Figure 1-2. Two example PR design floorplans for the chosen HW/SW PR partition (Partition A). Floorplan A's total wire length is less than floorplan B

CHAPTER 2
BACKGROUND AND RELATED WORK

The background and related research in this chapter is divided into two sections. Section 2.1 gives an overview of current vendor supported PR design flow and Section 2.2 discusses related work in the area of PR design flow automation.

2.1 Current Vendor Supported PR Design Flow

Currently, two FPGA device vendors support PR, Xilinx and Altera. PR is supported on Xilinx's Virtex-II and newer FPGAs, and Altera's Stratix-V devices. Xilinx's and Altera's PR design flows follow the same principle steps but in our case we consider the more mature and prevalent Xilinx PR design flow [Xilinx UG702 2013]. Figure 2-1 depicts Xilinx's PR design flow.

Xilinx's PR design flow requires a hierarchical logical partitioning of the HDL design files into a top module, static module(s), and one/or multiple non-overlapping PRMs. Next, the designer must execute the following steps: (1) synthesize each module's files separately to generate the module's respective netlists; (2) set design constraints, which includes creating the PR design's floorplan; (3) implement non-PR designs for every PRM to PRR combination and perform timing analysis on each design to verify timing requirements; (4) generate place and route information for the static region to create a static design with "holes" (un-placed and un-routed regions) for the PRMs and then generate place and route information for each respective hole's (i.e., PRR's) PRMs; and (5) merge the static design's place and route information with each PRM's place and route information to generate the PR design's full bitstream and all of the partial bitstreams.

Xilinx's ISE tool [Xilinx UG702 2013] performs step 1 using the Xilinx's XST [Xilinx UG628 2013] utility, and Xilinx's PlanAhead tool [Xilinx UG702 2013] performs steps 2-5 by aiding PR design floorplanning (step 2) with a graphical user interface, evaluating the corresponding PR design's timing results (step 3) using Xilinx's TRACE [Xilinx UG628 2013] utility, placing and routing the PR design (step 4) using Xilinx's NGDBUILD, PAR, and MAP utilities [Xilinx UG628 2013], and finally, generating the PR design's full and partial bitstreams using Xilinx's BITGEN [Xilinx UG628 2013] utility (step 5).

PR design floorplanning defines the area constraints (set in Xilinx's user constraints file (.ucf)) that specify each PRR's dimension (size and shape) and location on the FPGA, and optionally, the PRRs corresponding partition pin floorplan can also be specified. The PRR dimension area constraints are set as a range of XY coordinates that allocate the device resources within the specified range, and partition pin area constraints are set as XY coordinate locations around the PRR boundaries. To define the area constraints, designers must possess detailed knowledge of the target FPGA's resource locations (XY coordinates). Figure 2-2 depicts a Virtex-5 LX110T FPGA with example XY coordinates for the different resource types that PR system designers can allocate during PR design floorplanning. Resource types include configurable logic blocks (CLBs), block RAMs (BRAMs), first-in first-out buffers (FIFOs), and digital signal processors (DSPs). CLBs consist of slices and are the main logic resource used for sequential and combinatorial circuits.

It should be noted that both Xilinx's PR design flow and Altera's PR design flow do not support HW/SW co-design.

2.2 Related Work

In this section, we discuss prior work on PR design flows that consider Xilinx Virtex-4 or newer FPGA devices. These newer devices allow PRR floorplanning to specify rectangular regions inside the FPGA device, as compared to older devices that only allowed PRR floorplanning to specify columnar regions, which span entire device columns, and may waste resources. Newer devices provide more flexibility during the PR partitioning and floorplanning stage, which makes the PRR design space larger, increases the PRR floorplanning complexity, and increases potential PR benefits.

Previous work [Conger et al. 2008] proposed a framework for an automated PR design flow. The framework proposed partitioning PR designs from an application's task flow graph and performs automated floorplanning using heuristics, but no methodology for partitioning or floorplanning was presented. The authors however, manually investigated and evaluated clock frequency and partial bitstream size with respect to the PRR aspect ratio (PRR height in slices divided by PRR width) for PRMs with different resource requirements, which included slice-intensive, memory-intensive (more BRAMs), DSP-intensive, or a combination of these. The authors did not investigate PRR floorplanning with respect to PRR placement locations on the device.

The GOAHEAD PR design flow framework [Beckhoff et al. 2012] was presented to aid PR system designers in PR design partitioning and PR design floorplanning. No PR design partitioning methodology was presented. However, the design flow included an automated PRR floorplanning methodology that placed PRRs starting from the center of the device and a partition pin floorplanner that placed partition pins on the left and/or right borders of the PRR starting from the bottom. The authors presented the

design flow details, but no results evaluated the effectiveness of their floorplanning methods.

In [Montone et al. 2010] a PR design partitioning and floorplanning methodology was presented. The partitioning methodology leveraged a scheduled application task flow graph and the floorplanning methodology considered minimizing area requirements. Since the partitioning methodology considers a scheduled task flow graph, the presented design flow cannot be applied to adaptive systems.

An automated PR design flow [Vipin et al. 2013] was presented that performed automated partitioning of a PR design's HDL design descriptions, performed automated PRR floorplanning, and outputted the PR designs' bitstreams using Xilinx utilities, similarly to Xilinx's PR design flow. The partitioning methodology considered adaptive systems but does not consider the effects of HW/SW co-design during partitioning. Moreover, the authors did not include results on the quality of the final generated bitstreams with respect to any performance metrics (e.g., clock frequency, partial bitstream sizes). Also, no investigations on how partition pin floorplans would affect the final PR design were presented.

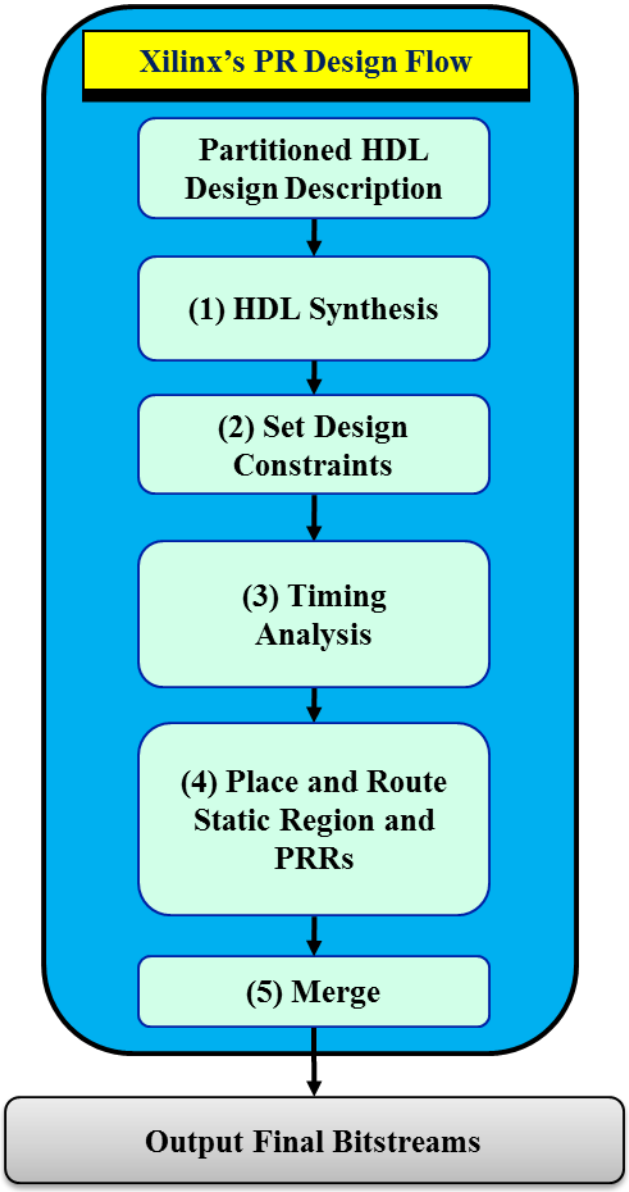

Figure 2-1. Xilinx's PR design flow

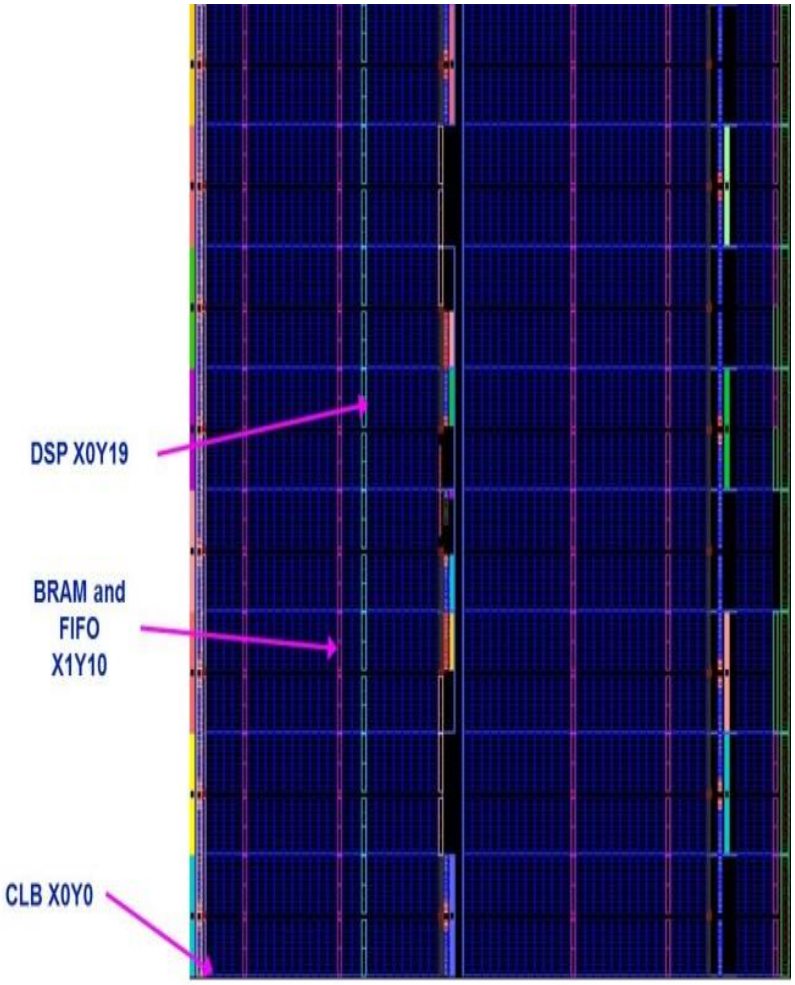

Figure 2-2. Virtex-5 LX110T FPGA with example XY coordinates of the different resource types PR system designers can allocate during PR design floorplanning

CHAPTER 3
DAPR DESIGN FLOW OVERVIEW

In this chapter we provide an overview of our design automation for partial reconfiguration (DAPR) PR design flow. The chapter is divided into two sections. Section 3.1 describes our proposed PR design flow and Section 3.2 discusses the target architecture.

3.1 Proposed PR Design Flow

HW/SW PR design partitioning in one of the most performance critical step in HW/SW co-designed applications. HW/SW PR design partitioning divides and maps application tasks to either software or hardware, where software tasks run on a processor and hardware tasks run on a target FPGA. Hardware tasks are further mapped onto the PRRs. Since, HW/SW PR design partitioning determines the number of hardware tasks and also the number of PRRs, HW/SW PR design partitioning directly determines the hardware resource requirements for a HW/SW co-designed application and the overall reconfiguration time for the design. For example, the more tasks assigned to hardware increases hardware resource requirements, and for an application that switches functionality frequently, a smaller number of PRRs will result in more reconfiguration time. Thus, HW/SW PR design partitioning is the best place to optimize the hardware resource requirements and reconfiguration time of a HW/SW co-designed application.

Another performance critical step in HW/SW co-designed applications is PR design floorplanning. PR design floorplanning determines the physical placements of the hardware tasks on the FPGA assigned during HW/SW PR design partitioning. The quality of the placement determines the wire routing of the design and directly affects

the clock frequency and thus, over all execution time. Thus, PR design floorplanning is the best place to optimize the clock frequency of a HW/SW co-designed application.

Figure 3-1 shows an overview of our proposed design automation for partial reconfiguration (DAPR) PR design flow. The design flow takes an application as input, performs initial analysis and planning on the application to make the application ready for automated partitioning, followed by automated floorplanning to generate the final design bitstreams. Automated partition generates a Pareto-optimal partition list of HW/SW partitioned PR designs that tradeoff resource requirements and reconfiguration time. Automated PR design floorplanning is performed on a system-designer chosen HW/SW PR partitioned design to improve the designs clock frequency.

3.2 Target Architecture

Since the DAPR design flow's HW/SW PR partitioning can generate partitions with any number of PRRs we require a target architecture that can easily vary the number of PRRs parametrically. We leverage VAPRES [Jara-Berrocal et al. 2010] to meet our highly flexible architectural needs. VAPRES is an architecturally customizable multipurpose PR system consisting of a MicroBlaze soft-core processor connected to a designer-specified number of PRRs. VAPRES's highly customizable architectural parameters provides a scalable system to enable system designers to meet varying design specifications for a wide range of application domains. A customizable inter-module communication architecture (SCORES) [Jara-Berrocal et al. 2010] provides inter-PRR communication if necessary. In this section, we provide a brief overview of the VAPRES architecture and the VAPRES system and application design flows.

The VAPRES architectural layout consists of two fundamental regions: the controlling region and the data processing region. The controlling region consists of a

MicroBlaze processor and a set of static peripherals (e.g., ICAP, UART, memory controller, etc.). The controlling region runs software tasks on the MicroBlaze, reads bitstreams from internal or external memory to perform PR via the ICAP, and controls data processing in the PRRs via the PRSockets. The data processing region consists of reconfigurable streaming blocks (RSBs), where each RSB has a designer-specified number of PRRs, I/O modules (IOMs), and a linear array of communication switches (the SCORES inter-module communication architecture). The PRRs in a RSB runs hardware tasks.

Figure 3-2 shows a sample VAPRES architectural layout with one RSB containing three PRRs and two IOMs, each connected to one SCORES switch software (four total switches). Each PRR communicates with the MicroBlaze using asynchronous FIFO-based fast simplex links (FSL) and each IOM allows external I/O pins or peripherals to communicate with the PRRs using SCORES. For each switch-PRR or switch-IOM pair, a PRSocket allows the MicroBlaze processor to control switch, PRR, IOM, and module interface operation using the device control register (DCR) [Xilinx DS402 2005] and additional interfacing logic. The DCR contains control bits that are set by the MicroBlaze using the general purpose input/output (GPIO) peripheral. Figure 3-3 shows a sample PRSocket. Using the PRSocket, the MicroBlaze configures the PRRs' independent clock frequencies using local clock domains (LCDs).

VAPRES system design combines two design flows to create a PR architecture [Jara-Berrocal et al. 2010]. The base system design flow aids system designers in generating a base PR architecture for application development. The application design flow aids application designers in creating applications for the base platform.

The base system design flow leverages designer-specified base system specifications, which includes parameters such as the maximum number of PRRs, communication channel width, number of one-way communication channels between switches, and the number of input and output channels between each PRR and the PRR's switch. After setting the base system specifications, the system designer defines the base system floorplan in a user constraints file (UCF) and creates a VHDL file modeling the static region, the microprocessor hardware specification (MHS) file defining the system structure, and the microprocessor software specification (MSS) file defining the base system build process. Finally, the synthesis and implementation steps manipulate the UCF, VHDL, MHS, and MSS files to generate the base system's static bitstream.

The application flow requires the application designer to decompose an application into the hardware and software modules, in our case the output from our HW/SW PR design partitioning, where Hardware modules follow the hardware module design flow, which requires the application designer to write, synthesize, and implement the VHDL hardware modules' (PRMs') code and respective VHDL wrappers to generate the PRR partial bitstreams. Software modules follow the software design flow, which requires the application designer to write and compile the software code to generate an executable for the MicroBlaze.

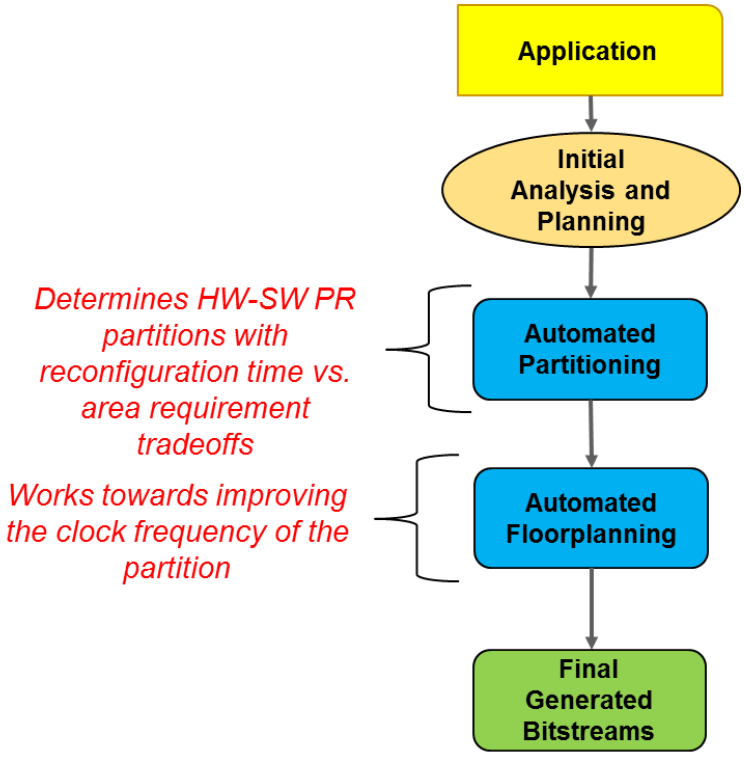

Figure 3-1. Design automation for partial reconfiguration (DAPR) design flow.

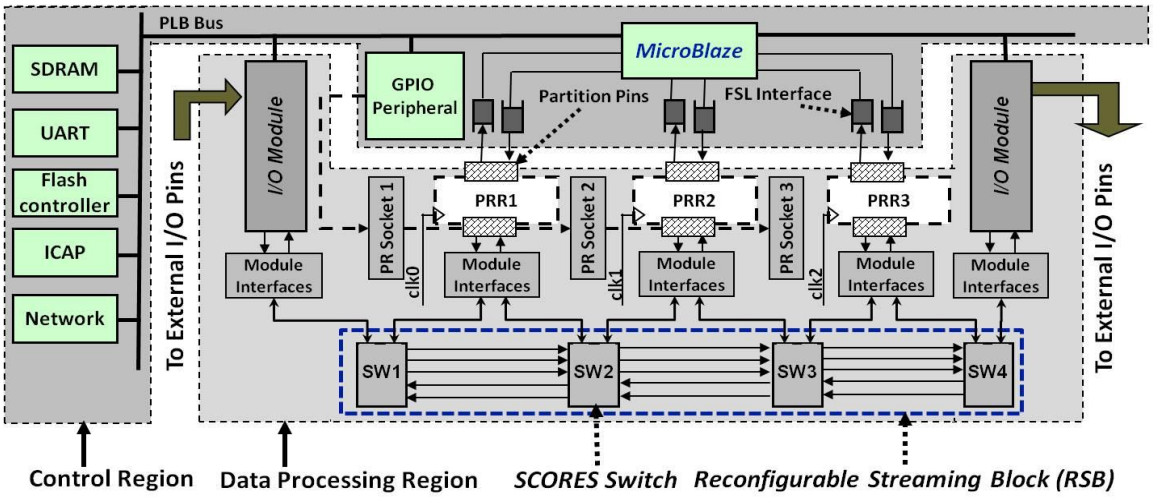

Figure 3-2. Sample VAPRES architectural layout showing a single reconfigurable streaming block (RSB)

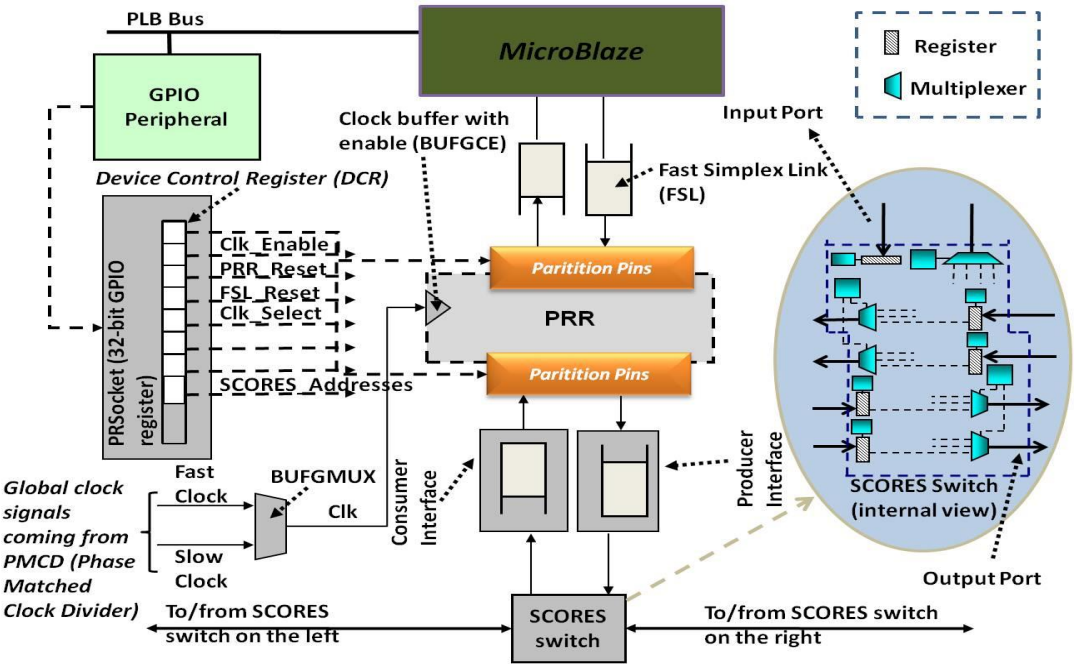

Figure 3-3. PRSocket signals to the PRRs, switch boxes, and module interfaces

CHAPTER 4
DAPR DESIGN FLOW'S AUTOMATED PARTITIONING

The DAPR design flow's automated partitioning performs HW/SW PR partitioning and outputs a Pareto-optimal set of HW/SW co-designed PR design partitions that trade off reconfiguration time and resource requirements. Corresponding total system execution time for each HW/SW PR partition is also output. DAPR design flow's HW/SW partitioning consists of an initial planning and analysis step and an automated step. During the initial step, an application is analyzed to estimate the resource requirements, reconfiguration time, and execution time (hardware and software) for each task. During the automated step the analyzed data is leveraged to output the Pareto-optimal set of HW/SW co-designed PR design partitions.

Section 4.1 describes the formulation involved for the planning and analysis step, Section 4.2 describes the methodology used in the automated step, and Section 4.3 evaluates the DAPR design flow's HW/SW PR partitioning methodology.

4.1 Problem Formulation

PR systems can dynamically change the application's currently executing functionality (i.e., the set of tasks current running in the PRRs) during runtime and thus can have multiple application task flow graphs. We designate each unique task flow graph as a *configuration*. Figure 4-1 shows an example of two possible configurations for the same application with five tasks T1-T5.

Typically for HW/SW PR partitioning, a simple method for hardware and software task assignment is to choose a configuration and generate a HW/SW PR partition optimized for that chosen configuration. The problem with this method is that the current generated partition may severely impact the performance of another configuration by

requiring significantly higher reconfiguration time. PR system designers can also employ another simple method for hardware and software task assignment by assigning control-related tasks to software and computationally-intensive tasks to hardware, which is a logical and feasible assignment. However, assigning all computationally-intensive tasks to hardware will require a lot of hardware resources and may be infeasible due to system designer and device fabric constraints. Also, due to the availability of high speed processors in modern FPGAs, some computationally-intensive tasks run in software may have comparable execution speeds to hardware, so there is no need to run that particular task in hardware, which would incur additional reconfiguration time overhead.

Therefore, there is no one simple solution to determine a HW/SW PR partition and to properly determine a HW/SW PR partition best suited to the system designer's goals, every combination of HW/SW task assignment needs to be evaluated to determine an optimal partition that balances resource requirements and reconfiguration time within a large design space.

To perform the evaluation for every combination of HW/SW assignment for an application, we define the arrays T_i, RR_i, TR_i, and CI_i, CO_i where T_i contains the task list of an application parsed from the modularized application source code. Then for each task in T_i, RR_i contains the list of CLBs, BRAMs and DSPs required by task, TR_i contains the corresponding reconfiguration time for the task, and CI_i contains the corresponding hardware and software execution time for the task. CO_i contains the list of possible configurations given by the PR system designer. Table 4-1 lists the arrays and content description.

RR_i is estimated by synthesizing the task's module using vendor specific tools, in our case the Xilinx xst tool, and parsing the output report file. TR_i is estimated by calculating the number of frames in a region and the device reconfiguration speed. The number of frames is the smallest addressable unit of a FPGA device and the number of frames required by a region along with the device reconfiguration speed directly corresponds to the reconfiguration time for the task. For example, for a Virtex-5 device [Xilinx UG 191], one CLB contains 36 frames, one DSP 28 frames, and one BRAM contains 36 frames and the reconfiguration speed is approximately 234MB/s [Vipin et al. 2013]. CI_i is estimated by parsing out and analyzing the modularized applications source code. For each task, the number of operations (add, multiply, divide and subtract) are counted, multiplied by a weighted cost, and then summed to determine the corresponding total hardware or software execution time. The weighted cost for software and hardware is set by the clock cycles required by the software processor and FPGA hardware to perform each operation and is set by the system designer. To determine the number of operation calculations, the repetition of each operation is also considered by analyzing loops and recursive function calls. For simplicity and without loss of generality, we do not consider dynamically allocated loops and recursive functions, which is beyond the scope of our work. However, we can easily incorporate an application profiler to our methodology as in [Stitt el at.] to accurately estimate dynamically allocated operations.

4.2 HW/SW PR Partitioning Methodology

Figure 4-2 shows our HW/SW PR partitioning methodology and Figure 4-3 depicts the associated HW/SW PR partitioning algorithm. The HW/SW PR partitioning algorithm takes as input a modularized application's task list T_i, and the corresponding

RR_i, TR_i, and CI_i determined from the initial application analysis (line 1). The algorithm also takes as input the list of possible configurations, CO_i, provided by the system designer for the application given by the system designer (line 1). Each configuration in CO_i contains the list of tasks in the configuration.

Line 3-6 initializes four dynamically allocated hash table data structures, %Partition_list, %Partition_max_resources, %Partition_reconfig_time, and %Partition_execution_time. %Partition_list contains are possible HW/SW PR partitions for each configuration. %Partition_max_resources contains the maximum possible resource requirements for all of each configuration's possible HW/SW PR partitions. %Partition_max_resources contains the maximum possible reconfiguration time requirements for all of each configuration's possible HW/SW PR partitions. %Partition_execution_time_configuration contains the maximum possible execution time for all of each configuration's possible HW/SW PR partitions.

%Partition_list is populated in line 7 of our algorithm by taking as input the list of tasks in each configuration CO_i and partitioning the tasks into separate pieces that satisfy the Bell recursion, where each piece is non-empty, disjoint and the union of the pieces is the complete list of tasks. For example, the number of all possible partitions of a set of n elements are known as the Bell numbers and satisfy the recursion: B(0) = 1, and B(n+1) = (n over 0)B(0) + (n over 1)B(1) + ... + (n over n)B(n). We ensure that the Bell recursion is satisfied in order to generate all possible HW/SW PR partitions. Figure 4-4 shows an example %Partition_list hash table snippet generated for three tasks. The primary key in the hash represents the configuration number. The secondary key represents the HW/SW PR partition number. The tertiary key represents the hardware

or software allocation. The tertiary key value represents the tasks allocations. Generating all possible partitions enables exhaustive searching of all HW/SW PR design partitions to determine the best partition suitable for system designer needs. The exhaustive search algorithm is feasible due to the initial T_i, and the corresponding RR_i, TR_i, and CI_i inputs (line 1) and does not take a significant amount of time. For example, for a PR design with sixteen tasks, our algorithm determined all possible partitions in approximately five minutes, when running on an Intel® Core ™ 7 2.5 GHz CPU with 8 GB of RAM. Typical real-world PR-based applications as in [Gonzalez et al. 2005], [McDonald 2008], [Huang et al. 2008], [Yousuf et al. 2009], [Hentati et al. 2009], [Yousuf et al. 2011], [Vipin et al. 2013], require the same or fewer numbers tasks, and thus this estimation is \ realistic.

Next, for each configuration's HW/SW PR partition, line 9-11 populate the %Partition_max_resources, %Partition_ reconfig_time, %Partition_execution_time data structures for each HW/SW PR partition by leveraging the Find_max_resources, Find_ reconfig_time, and Find_excecution_time functions. The Find_max_resources, Find_ reconfig_time, and Find_excecution_time functions populate %Partition_max_resources, %Partition_ reconfig_time, and %Partition_execution_time hash tables by utilizing the task list T_i in conjunction with the task resource requirements from RR_i, task reconfiguration time from TRi, and task execution time (HW or SW) from CI_i respectively.

At the algorithm's completion on line 11, the Pareto function leverages the populated %Partition_max_resources, %Partition_ reconfig_time hash tables to output the Pareto-optimal set of HW/SW PR partitions that tradeoff resource requirements and

reconfiguration time. The Pareto function also outputs the total execution time for each Pareto-optimal HW/SW PR partition leveraging the populated %Partition_execution_time hash table.

4.3 HW/SW PR partitioning Evaluation

The HW/SW PR partitioning evaluation is divided into two sections. Section 4.3.1 describes the test environment and the test design, and Section 4.3.2 discusses the associated initial analysis and results obtained.

4.3.1 Experimental Setup

We executed all experiments on a PC with an Intel® Core ™ 7 2.5 GHz CPU with 8 GB of RAM. We leveraged the Eclipse platform version 3.8.1 with EPIC Perl IDE version 0.5 installed to write our HW/SW partitioning algorithm and leveraged Xilinx ISE 14.7 and a Virtex-5 LX110T to perform the initial analysis of our application.

For our experiments, we evaluated the HW/SW PR partitioning algorithm on a modularized JPEG encoding and decoding (CODEC) application. To determine the list of possible configurations and the application's tasks' corresponding T_i, RR_i, TR_i, and CI_i, we first analyzed the behavior of the JPEG CODEC application.

The JPEG CODEC application has two different functionalities: Encoding and decoding. The JPEG encoding process for color images is divided into four main steps: Color Space Transformation - red, green, blue (RGB) image data to luma (Y), chroma blue (Cb) and chroma red (Cr), forward discrete cosine transform (FDCT), quantization, and entropy encoding. Figure 4-5 shows the JPEG encoding process. The arrowhead represents the dataflow direction. During JPEG encoding, 8 by 8 pixel image data blocks are first fed into the color space transformation algorithm RGB to YCbCr, which converts each pixel block to a luma component and two color differences of chroma

blue (Cb) and (Cr). Even though, image systems display colors in RGB, RGB representation is not efficient for data processing and hence the color space transformation is performed. Next, the color space transformed pixel data blocks are fed into a FDCT block, which converts the input data to 8 by 8 (64) DCT coefficients with lower spatial frequencies [Wallace 1992]. Next, the 64 DCT coefficients are uniformly quantized with a user specified quantization table. The quantization table specification selects the precision in which the 64 DCT coefficients are represented, sets the desired image quality (i.e., better precision means better image quality) and thus, the main lossy portion of the JPEG encoding process. The quantized coefficients are then ordered in a zig-zag sequence from low-frequency to high-frequency coefficients to ease the final JPEG encoding step, entropy encoding. Entropy encoding further compresses the 64 coefficients, and typically uses either Huffman coding or arithmetic coding [Wallace 1992]. Huffman coding requires predefined Huffman tables, whereas arithmetic coding does not. Moreover, arithmetic coding compresses images 5-10% better than Huffman coding. However, the arithmetic coding algorithm is a more complex and more computationally intensive relative to Huffman coding. Figure 4-6 depicts the JPEG decoding process and performs the encoding process in reverse as shown by the arrowhead direction. We refer the reader to [Wallace 1992] for more details on the JPEG encoding and decoding process.

4.3.2 Initial Analysis and Results

To determine the task flow graphs and the corresponding configurations for the JPEG encoding and decoding process, we analyzed hardware implementations of the JPEG encoding and decoding process as in [Yousuf et al. 2009]. Figure 4-7 depicts the hardware implementation of the JPEG encoding process, where 8 by 8 image pixel data

is read from the host and the various JPEG encoding steps occur as a pipelined process to encode the image data. Figure 4-8 depicts a hardware implementation of the JPEG decoding process, where 8 by 8 pixel data is read from external memory, and the various JPEG decoding steps occur as a pipelined process to decode the image data. We determined two possible configurations from the JPEG CODEC's hardware implementation where, the first configuration is the encoding process and the second configuration is the decoding process. Figure 4-9 (A) and (B) show the encoding and decoding processes task flow graphs, respectively.

The hardware implementation of the JPEG encoding and decoding process was analyzed to determine the resource requirements RR_i for each task. Table 4-2 lists the estimated CLBs, BRAMs, FIFOs, and DSPs requirements for the JPEG encoder and decoder tasks. After resource requirements were determined, the corresponding reconfiguration time, TR_i, for each task was calculated using the device reconfiguration speed for the Virtex-5 LX110T device taken from [Liu et al. 2009]. Reconfiguration time was reported in terms of the estimated number of frames required to reconfigure the hardware resources for the allocated task. Table 4-3 lists the number of frames required for the JPEG encoder and decoder tasks. We also determined CI_i by estimating the number of operations for each task by parsing the application source code and multiplying the number of operations with the weighted cost (required clock cycles for operation) for add, subtract, multiply, and divide operations with respect to the Virtex-5 LX110T device. Typical JPEG CODEC applications do not require floating point operations and the weighted costs were thus set for integer operations taken from [Xilinx DS100 2009]. For the software add and subtract, the weighed cost was to 1. For

the software divide and multiply, the weighted cost was set to 3 and 34 respectively. For the hardware add, subtract, multiply, and divide, the weighed costs were set to 1. Table 4-4 shows the estimated hardware and software execution times determined for the JPEG CODEC application.

Figure 4-10 and Figure 4-11 show the JPEG CODEC application's encoder and decoder configuration's resource requirements and reconfiguration time for each HW/SW PR partition. The resource requirements are reported in number of slices (Virtex-5 CLB's consist of four slices) and reconfiguration time is reported in number of frames. As expected, Figure 4-12 and Figure 4-13 illustrate our HW/SW PR partitioning algorithm's design space exploration for the vast number of possible HW/SW PR partitions for the JPEG encoder and decoder configurations.

Figure 4-14 and Figure 4-15 illustrate the JPEG CODEC application's encoder and decoder configuration's resource requirements versus reconfiguration time for each HW/SW PR partition. As expected we can see there is a multitude of possible HW/SW PR partition choices available to a PR system designer depending on system designer goals. For example, if the system designer's only goal is to select a HW/SW PR partition with low resource requirements and reconfiguration time, the HW/SW PR partitions closer to Figure 4-12 and Figure 4-13's origin should be chosen. However, these HW/SW PR partitions may have unacceptable total system execution time with respect to system designer goals, thus Figure 4-14 and Figure 4-15 illustrate the JPEG CODEC application's encoder and decoder configuration's total system execution for each HW/SW PR partition. Utilizing the data obtained from the figures, system designers can easily make an informed choice on a HW/SW PR partition that will meet system

designer goals for multiple configurations. For example, if a PR system designer's goal is to choose a HW/SW PR partition with resource requirements below 12,000 slices, reconfiguration time below 9,000 frames, and total system execution below 25,000 cycles for both the JPEG encoder and JPEG decoder configuration, any HW/SW PR partition between partition numbers 1-1000 can be chosen.

Table 4-1. Initialization Arrays for HW/SW PR design partitioning

Array Name	Array Contents
T_i	Application's task list
RR_i	Resource requirements for task i
TR_i	Reconfiguration time required by task i
CI_i	Software and hardware execution times of task i
COi	List of application's possible configurations

Table 4-2. JPEG CODEC estimated task resource requirements

Task List	CLBs	DSPs	BRAMs
RGB2YCbCR and FDCT	406	7	5
YCbCrtoRGB and IDCT	400	7	5
Run Length Encoding	42	0	2
Run Length Decoding	42	0	2
Huffman Encoder	280	1	2
Huffman Decoder	350	1	2
Byte Stuffer and header encoder	14	0	1
Byte Stripper and header decoder	14	0	1
Quantization	14	2	3
Dequantization	14	2	3
Zigzag	14	0	2
Reorder	14	0	2

Table 4-3. JPEG CODEC estimated task reconfiguration times in frames

Task List	Reconfiguration time (Frames)
RGB2YCbCR and FDCT	1650
YCbCrtoRGB and IDCT	1600
Run Length Encoding	480
Run Length Decoding	480
Huffman Encoder	1000
Huffman Decoder	1100
Byte Stuffer and header encoder	100
Byte Stripper and header decoder	100
Quantization	170
Dequantization	170
Zigzag	120
Reorder	120

Table 4-4. JPEG CODEC estimated hardware and software execution times in cycles

Task Name	SW Execution (cycles)	HW Execution (cycles)
RGB2YCbCR and FDCT	40,000	10,000
YCbCrtoRGB and IDCT	42,000	10,000
Run Length Encoding	10,000	2,000
Run Length Decoding	10,000	2,000
Huffman Encoder	25,000	10,000
Huffman Decoder	30,000	10,000
Byte Stuffer and encoder	20	20
Byte Stripper and header decoder	20	20
Quantization	40	40
Dequantization	40	40
Zigzag	40	40
Reorder	40	40

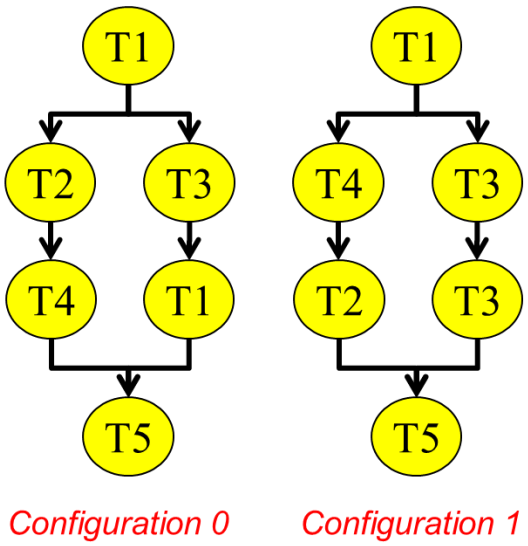

Figure 4-1. Two example configurations of an application with five tasks T1 to T5

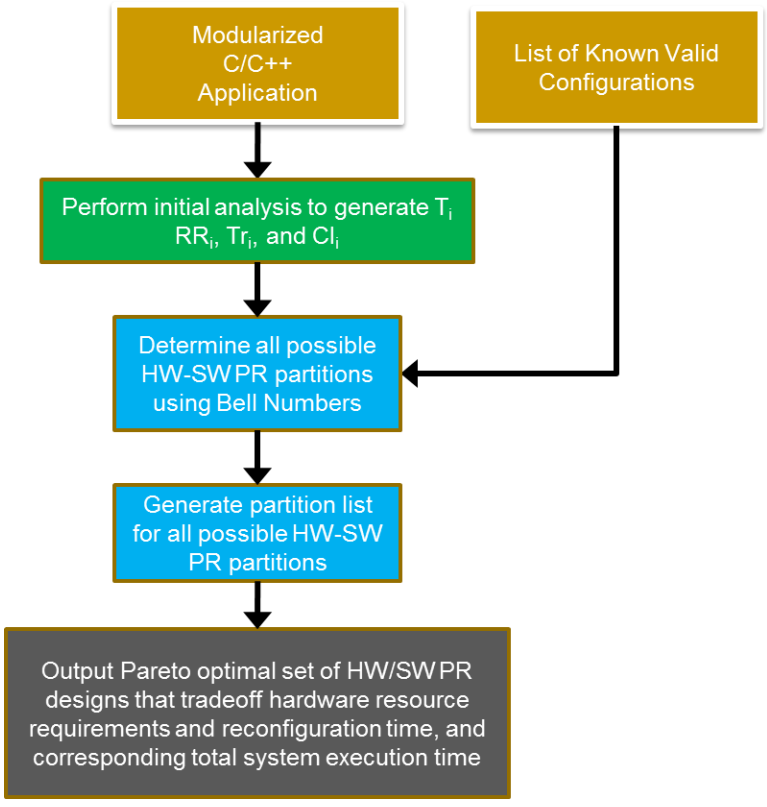

Figure 4-2. HW/SW PR design partitioning methodology

1 **Input:** @Ti, @RRi, @TRi, @CIi, @CO_i
2 **Output:** Pareto-optimal set of PR partitions , corresponding total execution time
3 %Partition_list;
4 %Partition_max_resources ← 0;
5 %Partition_reconfig_time ← 0;
6 %Partition_execution_time ← 0;
7 %Partition_list ← Partition_task($$CO_i$$);
8 %Partition_max_resources ← Find_max_resources (%Partition_list, @Ti, @RR_i);
9 %Partition_ reconfig_time ← Find_ reconfig_time (%Partition_list, @Ti, @TR_i);
10 %Partition_execution_time ←Find_execution_time(%Partition_list, @Ti, @CI_i);
11 **Return** *Pareto*(%Partition_max_resources, %Partition_ reconfig_time,%Partition_execution_time)

Figure 4-3. HW/SW PR design partitioning algorithm

```
$Partition_List = {
        'CO_0' => {
                '0' => {
                    'SW' => 't0,t1,t2'
                  },
                '1' => {
                    'SW' => 't0,t1',
                    'PRR 1' => 't2'
                  },
                '2' => {
                    'SW' => 't0,t2',
                    'PRR 1' => 't1'
                  },
                '3' => {
                    'SW' => 't0',
                    'PRR 1' => 't1,t2'
                  },
                '4' => {
                    'SW' => 't0',
                    'PRR 1' => 't1',
                    'PRR 2' => 't2'
```

Figure 4-4. Example partition list hash data structure snippet generated for three tasks. Primary key represents configuration number. Secondary key represents HW/SW PR partition number. Tertiary key represents hardware or software allocation, and the tertiary key value represents the corresponding allocated tasks

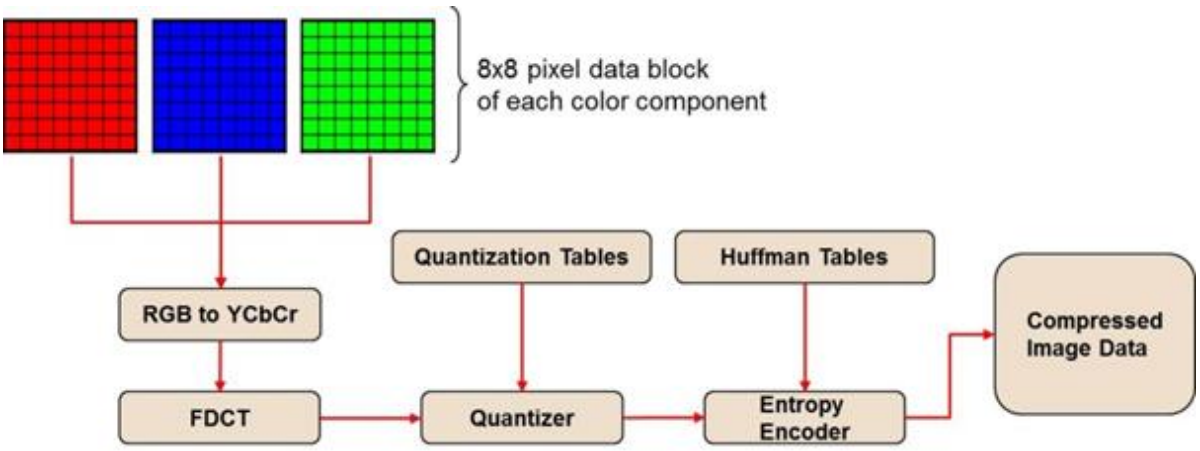

Figure 4-5. JPEG encoding process

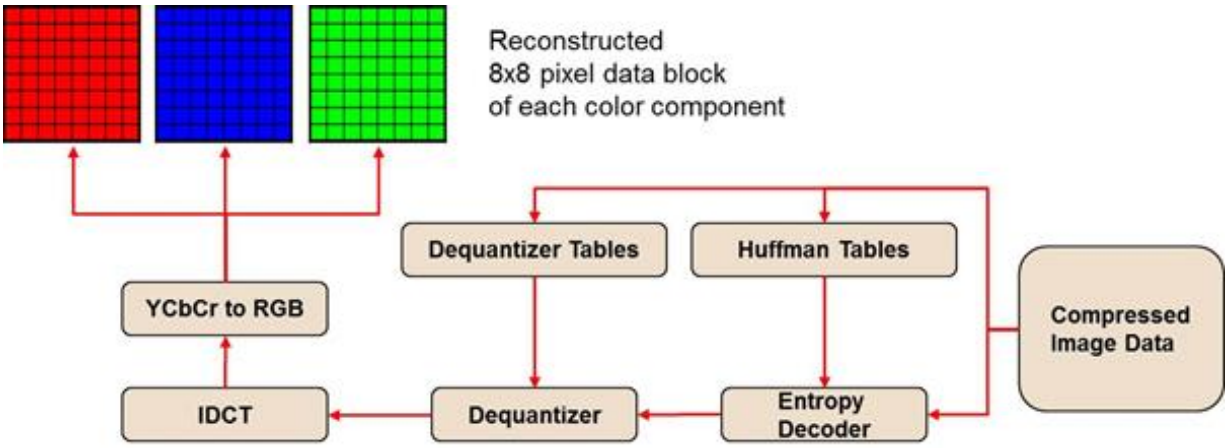

Figure 4-6. JPEG decoding process

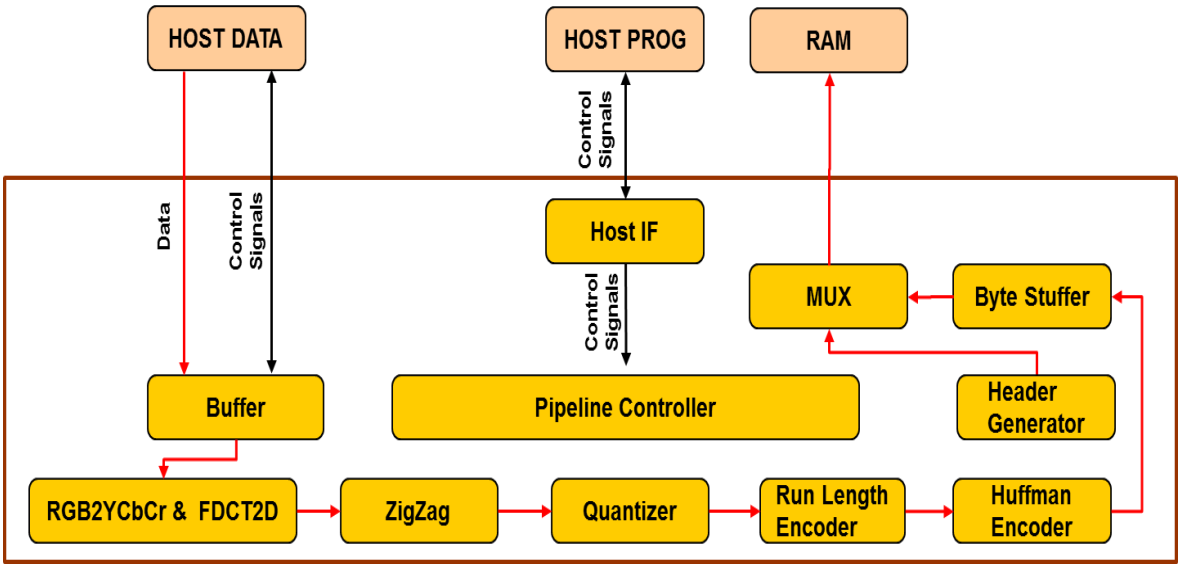

Figure 4-7. Hardware implementation of JPEG encoding process

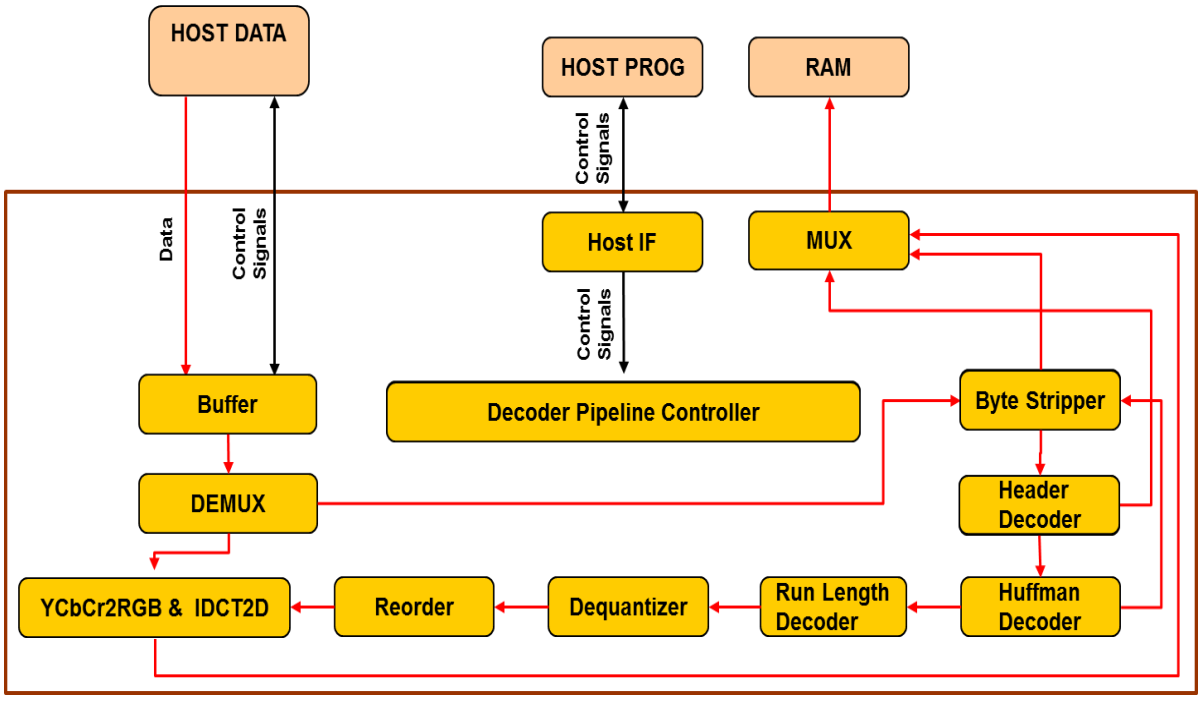

Figure 4-8. Hardware implementation of JPEG decoding process

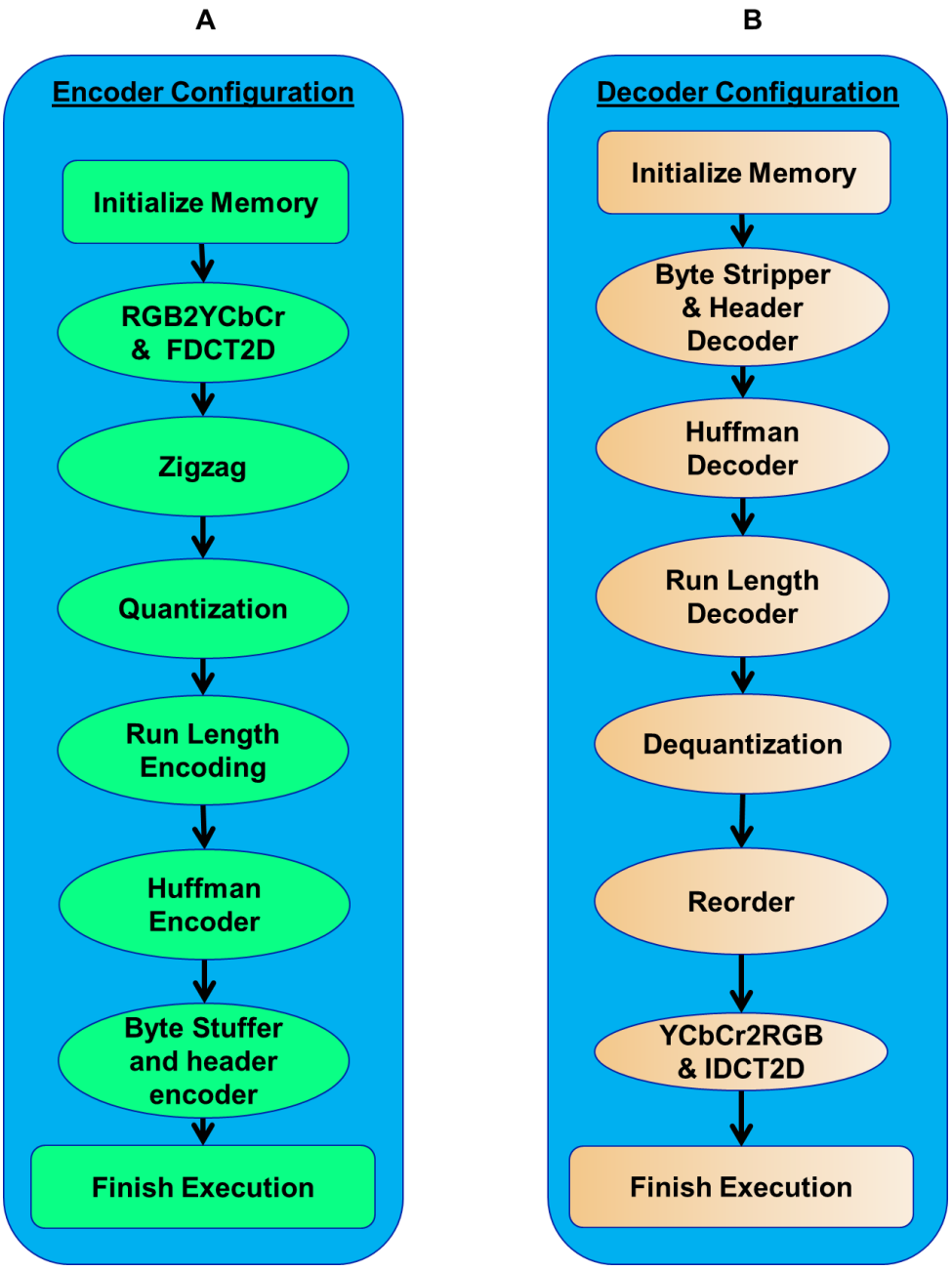

Figure 4-9. Task flow graphs (A) JPEG encoder process, (B) JPEG decoder process

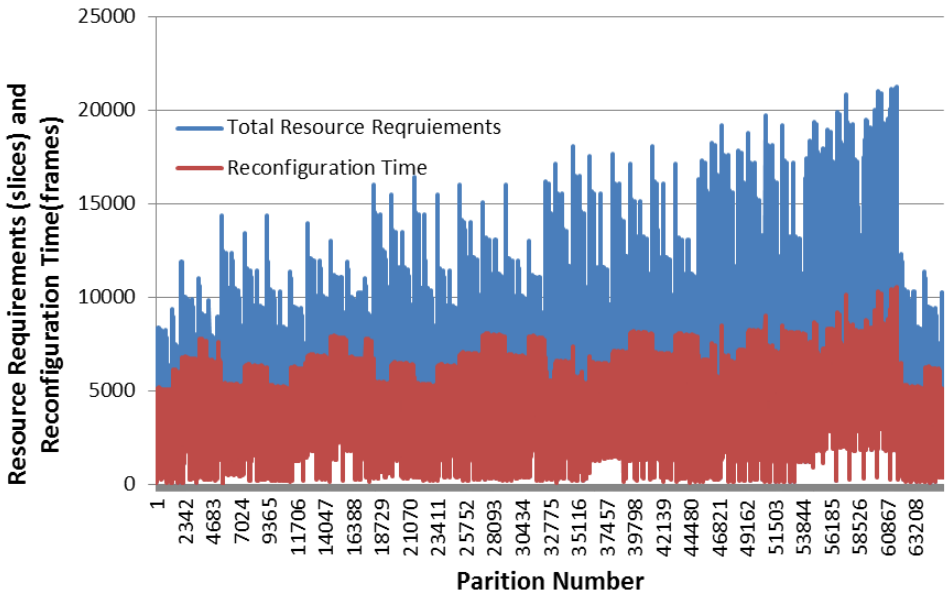

Figure 4-10. JPEG CODEC encoder configuration's resource requirements and reconfiguration time for generated HW/SW PR partitions

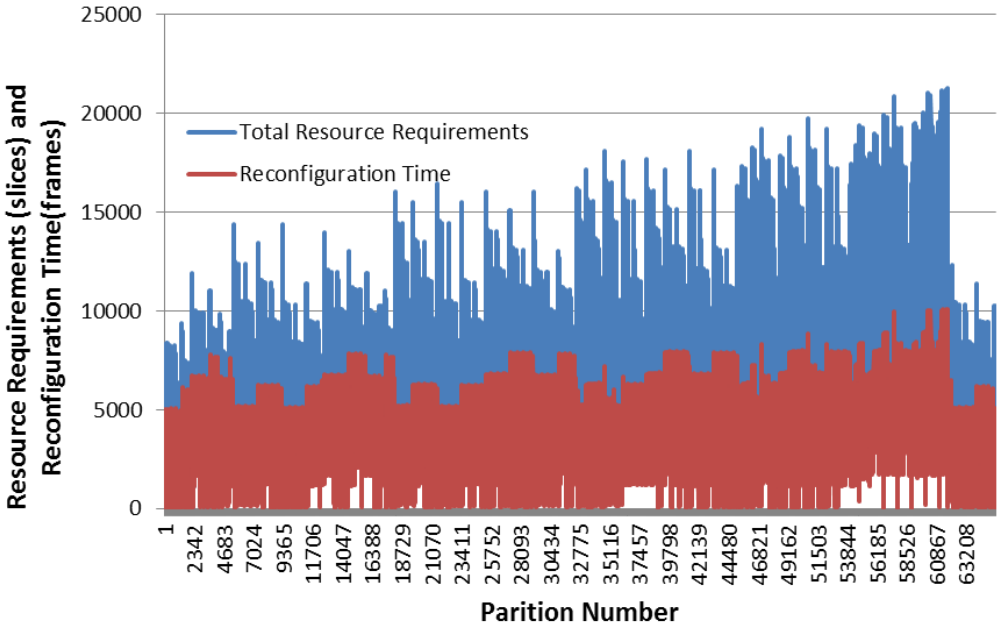

Figure 4-11. JPEG CODEC decoder configuration's resource requirements and reconfiguration time for generated HW/SW PR partitions

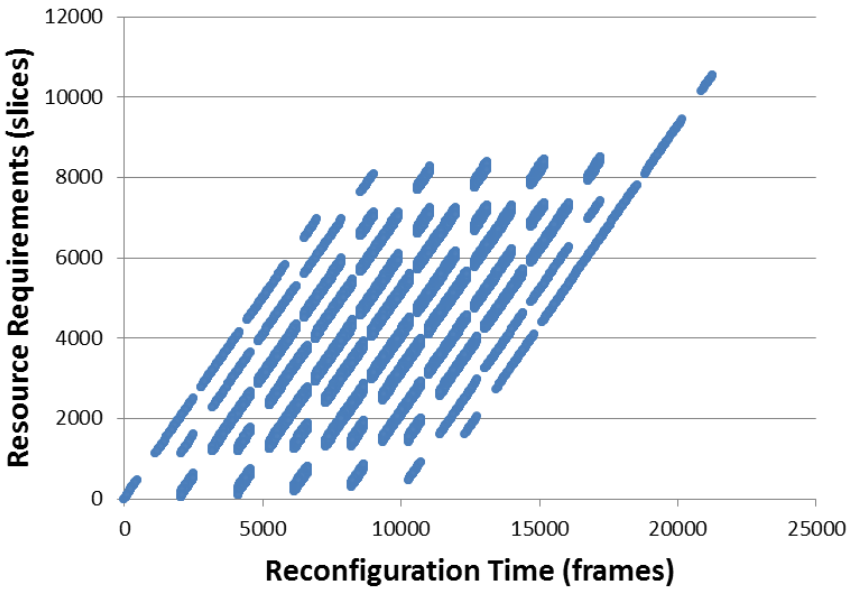

Figure 4-12. JPEG CODEC encoder configuration's resource requirements vs. reconfiguration time for generated HW/SW PR partitions

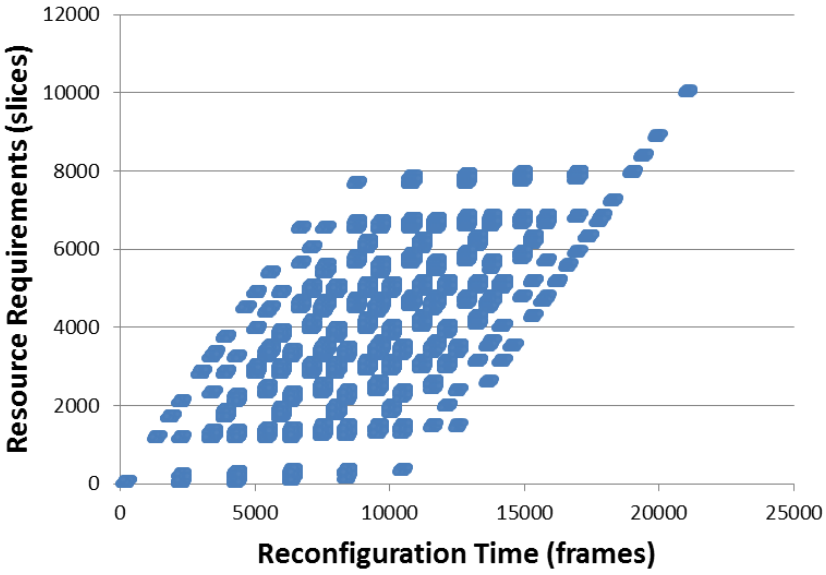

Figure 4-13. JPEG CODEC decoder configuration's resource requirements vs. reconfiguration time for generated HW/SW PR partitions

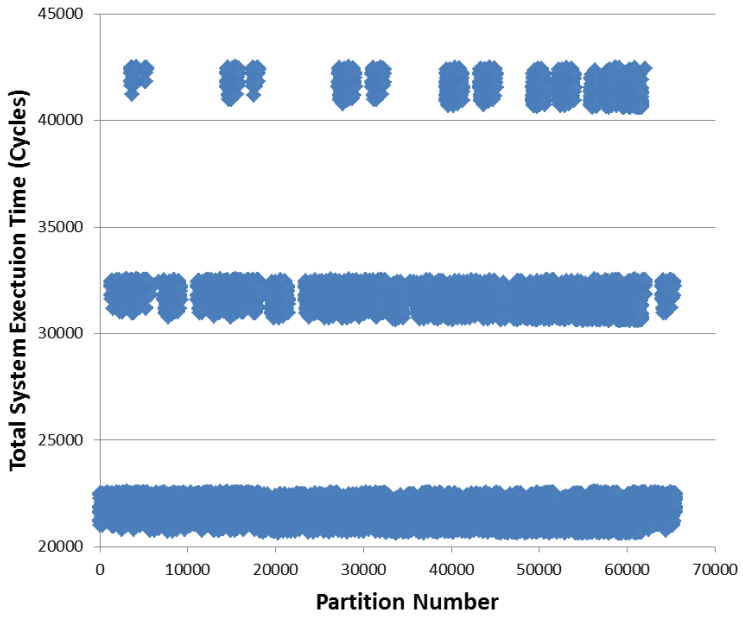

Figure 4-14. JPEG CODEC encoder configuration's total system execution time for generated HW/SW PR partitions

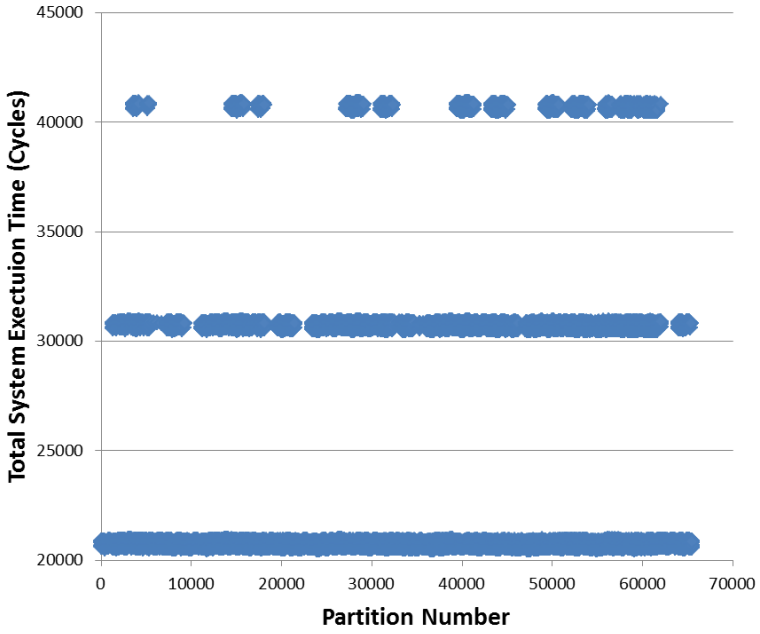

Figure 4-15. JPEG CODEC decoder configuration's total system execution time for generated HW/SW PR partitions

CHAPTER 5
DAPR DESIGN FLOW AUTOMATED FLOORPLANNING

The DAPR design flow automated floorplanning works towards improving the clock rate of a chosen HW/SW PR design partition and consists of a manual step (1) and an automated step (2). In the manual step (1), the designer sets DAPR-tool-specific annotations in the top-level VHDL design file, and sets device-specific I/O pin location constraints in a design constraints file (.dcf). Even though these are manual steps, designer effort is minimal compared to the manual steps required in Xilinx's PR design flow, and these annotations and design constraints are necessary input into the automated step (2). The automated step (2) is orchestrated by the DAPR floorplanning tool, and automates Xilinx's PR design flow's complex steps. Even though our discussions and experiments concentrate on Xilinx tools and devices, the DAPR design flow's floorplanning methodology can be easily adapted for different PR devices by updating the command line calls, which invoke a specific vendor utility.

Section 5.1 describes the floorplanning design space considerations, Section 5.2 provides an overview of the automated floorplanning methodology, Section 5.3 describes the DAPR floorplanning tool, and Section 5.4 evaluates the DAPR floorplanning tool.

5.1 Design Space Considerations

The PRR design space size depends on the number of PRRs and the PR design's total PRR resource requirements with respect to the available device resources. The PR design's total number of PRRs affects the PRR design space size since the number of PRR placement permutations with respect to each other increases as the number of PRRs increases. The PR design's total resource requirements affects

the PRR design space in two ways: (1) as the amounts of available device resources increases, the flexibility of where the PRRs can be floorplanned inside the device increases, which increases the PRR design space size; and (2) depending on the PR design, some PRRs may require special resource types available only at specific device locations, which constrains the PRR's placement to specific device locations device, which reduces the PRR design space size.

As the partition pins are floorplanned on the PRR's boundaries, the partition pin design space depends on the number of partition pins with respect to the total perimeter of the PR design's PRRs. For example, as the number of partition pins increases, the partition pin placement permutations relative to each other increases, and as the total PRR perimeter increases, the number of available locations a partition pin can be placed increased.

Considering both the PRR and partition pin design spaces results in an extremely large PR design space, where the PR design space increases exponentially as the device size and PRR sizes increase. Given such a large PR design space, manual design space exploration to find PRR and partition pin floorplans with the smallest clock frequency degradation can be a very cumbersome, tedious, or perhaps infeasible, process. Therefore, even with Xilinx's PR design flow, realizing PR benefits is challenging since a lack of sufficient expertise can result in design performance that is worse than a functionally-equivalent FR design.

5.2 DAPR Design Flow Automated Floorplanning Overview

In the DAPR design flow's floorplanning manual step (1), the designer performs several straightforward tasks to prepare the PR design for the second automated step (2). First, the designer annotates the VHDL component instantiations in the top-level

design file using standard-formatted VHDL comments (the designer must also follow Xilinx's PR design partitioning guidelines, which assumes that all PRR instantiations are defined in the top-level file). Using VHDL comments ensures that these annotations will not introduce synthesis errors or affect design portability. Optionally, the designer can also define additional PR design constraints (e.g., timing, power, area, partial bitstream size, etc.) and the DAPR floorplanning tool's effort level control value in the design constraints file.

After completion of the manual step (1), the DAPR floorplanning tool's inputs are the annotated top-level design file, all other design files, and the .dcf file. The DAPR floorplanning tool manipulates these files in order to generate a PR design floorplan and evaluates the floorplan with respect to the design constraints (power, speed, area, partial bitstream size, etc.) in the .dcf file. If the design constraints are not met, the DAPR tool uses an iterative process to improve the PR design's floorplan using a simulated annealing-based (SA) algorithm [Sait and Youssef 1999]. This iterative process's effort is controlled via the DAPR tool variable, Imax, in the .dcf file. Imax specifies the maximum number of successful iterations the DAPR tool can perform to improve a PR design's floorplan. A successful iteration generates a valid, candidate PR design floorplan that passes place and route on the FPGA while running Xilinx's PAR utility, while an unsuccessful iteration generates a PR design floorplans that fails place and route, and is thus an invalid PR design. Since the DAPR floorplanning tool's SA-based algorithm iteratively improves the PR design floorplan, each successful iteration's candidate PR design is a potential Pareto-optimal PR design, and is retained for later Pareto-optimal determination.

5.3 DAPR Floorplanning Tool

The DAPR design flow's automated floorplanning takes as input a PR design's partitioned hardware despeciation language (HDL) design files and leverages the DAPR design flow's DAPR floorplanning tool to automatically perform PRR and partition pin floorplanning. The DAPR floorplanning tool automatically performs design exploration during floorplanning using a simulated annealing (SA)-based algorithm. The DAPR floorplanning tool's SA-based algorithm iterative works towards improving the PR design's clock frequency to attain the PR design with the lowest possible clock frequency degradation. By default the DAPR floorplanning tool runs for 100 iterations (which can be modified by the user), outputs the full and partial bitstreams of the PR design with the best found clock frequency, and a Pareto-optimal set of PR design bitstreams that trade off clock frequency and average partial bitstream size. For bitstream generation, the DAPR tool currently uses Xilinx's utilities, but since the utilities are executed using command line arguments, integrating newer or different vendor utilities is simple. Also, the DAPR tool's outputs can be easily tailored to user-specific requirements by modifying the SA-based algorithms cost function.

Figure 5-1 depicts the four phases of the DAPR tool. In phase 1, information identification uses the top-level PR design's file annotations to identify the static region(s) and PRR instantiations, and PR design file names. Information extraction extracts port map connection information from the region instantiations and stores the extracted connection information into an internal data structure. Collectively phases 2 through 4 iteratively generate candidate PR designs.

Phase 2, the candidate PR design floorplan generation phase, constitutes the bulk of DAPR's work, and automatically (1) synthesizes all design files using Xilinx's

XST utility, (2) estimates the hardware resource requirements from the generated synthesis log file (.srp) and records the requirements in an internal data structure, (3) generates a connectivity information file (cif.dot) to create a connectivity graph for the PR design, (4) uses the device information library data structure (DILDS), estimated resources, connectivity information, and .dcf file to build an initial candidate floorplan, and writes the current candidate floorplan constraints to a new .ucf file. The DILDS is generated by the DAPR tool and contains the target device's resource information.

Phase 3, the candidate PR design bitstream generation phase, uses Xilinx's NGDBuild, MAP, PAR, and BitGen utilities (similar to Xilinx's PR design flow's steps (4) and (5)) to output the PR design's full and partial bitstreams.

Finally, phase 4, the candidate PR design evaluation phase, determines if the current candidate floorplan meets the specified design constraints by computing the candidate PR design's clock frequency, bitstream sizes, and power requirements from the trace report file (.twr), power report file (.pwr), and map report file (.mrp) generated by Xilinx's TRACE, XPower, and MAP utilities, respectively. If any design constraints are not met, the DAPR floorplanning tool returns to phase 2's automated PR floorplanning (4) to create a new candidate PR design floorplan using the SA-based algorithm, and then repeats phases 3 and 4. This iterative process continues until the candidate PR design floorplan meets the design constraints or for a fixed number of successful iterations Imax.

On phase 4's completion, the DAPR floorplanning tool outputs the candidate PR design that meets the required design constraints. If the specified design constraints were not met, the DAPR tool by default outputs the candidate PR design with the

maximum attainable clock frequency. The DAPR floorplanning tool also compares all candidate PR designs and outputs a Pareto-optimal set of PR designs and respective iteration values that trade off clock frequency and partial bitstream sizes.

5.3.1 Candidate PR Design Generation

In this section, we elaborate on the candidate PR design generation details, including the internal make-up of the DILDS and how the DILDS is utilized by the DAPR tool to generate candidate PR design floorplans.

The DILDS is primarily used in conjunction with the DAPR tool floorplanner to build candidate floorplans and is generated using Xilinx's PARTGen [Xilinx UG628 2013] utility. PARTGen is a Xilinx command line tool that creates files containing architectural details about supported Xilinx devices [Xilinx UG628 2013]. Figure 5-2 shows a snippet of one of PARTgen's generated files (partlist.xct) created for the Virtex-5 LX110T device. The Virtex-5LX110T partlist.xct file contains information on the device's number of CLB rows and columns, slices per CLB, number of block rams and block ram locations with respect to CLB columns, etc. We refer the reader to [Xilinx UG628 2013] for more details on the partlist.xct file format.

The DAPR tool parses out information crucial for floorplanning from PARTgen's generated files to create the DILDS. For a target device, the DILDS contains entries specifying the XY coordinate locations of the CLBs, BRAMs, FIFOs, and DSPs. This resource type and location information enables an easy way to identify and allocate FPGA resources during the PR design floorplanning stage. Resources can be allocated for a PR design's PRRs by translating the DILDS entries into proper .ucf file syntax. The allocation syntax for a PRR is the PRR instance name, as defined in the top-level VHDL

file, followed by the required resource's type and the XY coordinates on the FPGA fabric (see [Xilinx UG702 2013] for syntax details).

Currently, the DAPR tool's DILDS generation has been tested and verified to work with all Virtex-4, Virtex-5, and Virtex-6 LX series devices. However, extending DILDS generation for other devices is conceptually identical and only requires target device- and resource-specific considerations.

5.3.2 Candidate PR Design Floorplan Generation

The DAPR tool floorplanning algorithm's goal is to generate a candidate PR design floorplan that meets all of the design constraints within a designer-specified number of successful iterations, which enables designers to control the design exploration time. The DAPR tool leverages the DILDS and an internal data structure containing the PR design's resource requirements to automatically generate candidate PR design floorplans. Candidate PR design floorplans are generated and improved using our SA-based PRR floorplanner algorithm and our SA-based partition pin floorplanner algorithm. Our PRR floorplanner algorithm begins PRR floorplanning from the device's bottom left corner, which is an arbitrary starting location and does not affect the overall behavior of the algorithm since the first few floorplans evaluated by an SA-based algorithm are random. Our partition pin floorplanner algorithm begins partition pin floorplanning from the center of the PRRs, since typically, pins placed in the center of rectangular modules (in our case PRRs) will result in lower half-perimeter wire lengths (HPWLs), which may result in a lower total wire length floorplan [Sait and Youssef 1999]. A similar assumption cannot be made during PRR floorplanning since PRRs may require different hardware resources and can be spread across the FPGA device. Both algorithms prioritize improving clock frequency over other design constraints to minimize

clock frequency degradations, which is the main concern for PR designer's desiring high-performance embedded RC systems, but the priority can be easily modified by updating each algorithm's cost function.

Typically, the DAPR tool's floorplanning algorithm should run the PRR floorplanner longer than the partition pin floorplanner because changes in the PRR floorplan usually affects the total wire length of the PR design more significantly than partition pin floorplanning. To set the number of successful iterations the DAPR tool's PRR floorplanning algorithm runs, we use the variables Imax and Ipp. Imax is the number of successful iterations the DAPR tool's floorplanning algorithm runs, and Ipp is the number of iterations the DAPR tool's partition pin floorplanning algorithm runs. Therefore, the number of successful iterations the DAPR tool's PRR floorplanning algorithm runs is Imax minus Ipp. Both Imax and Ipp are set in the .dcf file.

After a candidate PR design floorplan is generated and written to the .ucf file, the respective candidate PR design is generated and evaluated in phases 3 and 4, respectively. Section 5.3.3 describes the DAPR tool's SA-based PRR floorplanner algorithm and Section 5.3.4 describes the DAPR tool's SA-based partition pin floorplanner algorithm.

5.3.3 PRR Floorplanner Algorithm

Figure 5-3 depicts DAPR's PRR floorplanner algorithm, which takes as input the set of all PRMs and static modules (S), each region's maximum resource requirements (R), total number of PRRs (N), the number of iterations reserved for partition pin floorplanning (Ipp), the maximum number of successful iterations (Imax), and the DILDS.

Lines 3-14 initialize the PRR floorplanner algorithm. Line 3 creates the current successful iteration number Icurr and current best clock frequency design number found Ibest within Icurr successful iterations. Line 4 creates lists to store the clock frequency Clk_freq_list[] and bitstream sizes Bit_size[] of the currently evaluated design. Line 5 creates lists to store the initial floorplan solution E[], the best floorplan solution E_best[], and the new floorplan solution E_new[]. A perturbation function alters E[] to create E_new[]. Line 6 creates the X_global and Y_global variables, which set the starting XY coordinates where the floorplanner begins placing PRRs on the device. For example, a default starting value of zero for X_global and Y_global means that the floorplanner begins placing PRRs from the left most corner of the device. Lines 7-9 creates Int_frag_x, Int_frag_y, and White_space, which enable allocating extra resources during floorplanning to mitigate routing congestion in case of PAR utility failure. Int_frg_x and Int_frg_y allocate extra columns and rows of hardware resources, respectively, inside a PRR, and White_space allocates extra resources between PRRs. Line 10 creates Rand_range and Rand_min, which are used to set Irand (line 11). Irand sets the random number of successful iterations to perform before calculating the initial temperature T0 required by the SA portion of our PRR floorplanner algorithm. Line 12 creates Successful_iter and Error_count, which are used to flag and count the number of consecutive PAR failures, respectively, during the PAR step in the candidate PR design bitstream generation phase. Line 13 creates M, which sets the move/perturbation type performed used to update a PRR floorplan. Line 14 creates E[], which contains the list of all PRRs, N, in the form of a normalized polish expression (e.g., 12V3V4V. . .NV). In a normalized polish expression, the operands (1,2,3…..N)

correspond to the PRRs and the operators “V” or “H” represent cut types and correspond to vertical or horizontal placements of PRRs with respect to each other [Sait and Youssef 1999]]. The normalized expression is generated using Norm_Polish().

Lines 15-50 perform the initial randomized PRR placements. PRR_placement() (line 16) generates and writes the corresponding floorplanning constraints to a .ucf file using S, R, E[], X_global,Y_Global, and the DILDS. Bit_generation() (line 17) reads the .ucf file, which runs Xilinx’s NGDBuild, PAR, MAP, and BitGen to generate the design’s full and partial bitstreams, and a routed native circuit description file (.ncd). If NGDBuild, PAR, MAP, and BitGen are successful, Bit_generation() is successful (line 18), Icurr is incremented (line 19), and the current design’s bitstream sizes and clock frequency are stored in Bit_size[] and Clk_freq_list[], respectively (line 20). Design_evaluation() generates the current design’s clock frequency using Xilinx’s TRACE command line tool [Xilinx UG628 2013]. If Bit_generation() fails, Error_count is incremented (line 23). Five consecutive PAR errors force the algorithm to allocate extra resources during floorplanning by incrementing Int_frg_x, Int_frg_y, and White_space (lines 26-28) and resetting Error_count to 0 (line 29). Next, E[] is altered using Select_perturbation(M) (lines 31-44), where the value of M corresponds to the perturbation type: ‘1’ swaps two adjacent operands (PRRs) (line 33); ‘2’ inverts an operand chain by flipping V to H and H to V (line 34); ‘3’ swaps two adjacent operand and operators (lines 35-42) while ensuring that the solution expression does not violate the property of a normalized polish expression and the expression’s balloting property [Sait and Youssef 1999]] (line 38 and 40); and ‘4’ changes the PRR floorplan starting location. M is initially set to ‘1’ and is randomized each iteration, except when the current iteration is twice the number

of PRRs (lines 45-47), when M is set to ‘4’. Setting M to ‘4’ every time the number of iterations is multiple of 2N ensures that for each starting location, every operand-to-operator permutation has been explored.

Lines 51-59 compute and initialize additional variables required by the SA algorithm. Line 52 calculates the average clock frequency variations found during the initial random region placement using Average_cost() and stores this average cost change in Δavg. Lines 53 and 54 initialize the solution acceptance probability P to a high value of ‘0.99’ and a recommended temperature reduction rate λ of ‘0.85’ [Sait and Youssef 1999]. Line 55 resets E[] to the initial normalized polish expression (12V3V4V. . .nV). Line 56 initializes E_best[] as E[]. Line 57 sets counters for the number of PRR floorplans explored that decreased the PR design’s clock frequency (an uphill move), the number of rejected moves Reject, and total number of moves MT to ‘0’. Finally, in line 58, the average temperature T0 is calculated as:

$$T_0 = \frac{\Delta avg}{ln(P)} \tag{5-1}$$

Lines 60-94 drive our SA-based PRR floorplanning algorithm for exploring the PRR floorplans. Lines 63-65 perturbs E[], writes to the .ucf file, and performs bitstream generation, respectively, similarly to random PRR placement, increments the current iteration, Icurr, and total number of moves if the current iteration is successful. Again, if Bit_generation() is successful (line 65), Icurr and the total number of moves MT is incremented (lines 67 and 68), and the current design’s bitstream sizes and clock frequency are stored in Bit_size[] and Clk_freq_list[], respectively (line 69). In line 70, our SA-based PRR floorplanning algorithm’s cost function is defined, which is the clock frequency variation ΔCost_change from the previous iteration’s clock frequency. Lines

71-82 show that if the current iteration's clock frequency is higher than the previous iteration's clock frequency (line 71), the new PRR floorplan is accepted (line 74), otherwise, the new PRR floorplan is accepted with probability P:

$$P = e^{-\frac{\Delta avg}{T}} \quad (5\text{-}2)$$

If the current design's clock frequency is the best clock frequency so far (line 75), the floorplan and iteration number is stored in E_best[] and Ibest, respectively (lines 76 and 77). If the current iteration perturbation changed the floorplan's starting location variables X_global and Y_global, and the floorplan was rejected, X_global and Y_global are restored to the previous iteration's starting location values to ensure that a PRR floorplan's starting location that may lead to the global maxima is retained (line 81). Similar to random PRR placements, if Bit_generation() fails, lines 83-91 ensure that Error_count is incremented (line 84), and if five consecutive PAR errors occur, the algorithm allocates extra resources during floorplanning by incrementing, Int_frg_x, Int_frg_y, and White_space (lines 87-89) and resets Error_count to 0 (line 90). Lines 92-96 vary M similarly to random PRR placement by randomly choosing M (line 95) expect when Icurr is a multiple of 2N (line 92), where M is set to "4" (line 93).

After N uphill moves or when the total number of moves MT is greater than 2N (Line 97), the temperature T is decreased by a factor of 0.85 (line 98) to reduce the acceptance probability of an uphill move (line 71). Next, algorithm execution returns to line 60 to continue the SA-based PRR floorplanning algorithm exploration.

At the end of the PRR floorplanner algorithm (line 100), the best clock frequency Clk_freq_list[Ibest] and the corresponding iteration number Ibest have been determined

and the algorithm completes. Then the best PR design's partition pin floorplan is manipulated using our partition pin floorplanner algorithm.

5.3.4 Partition Pin Floorplanner Algorithm

Figure 5-4 depicts DAPR's SA-based partition pin floorplanner algorithm, which takes as input the initial partition pin floorplan PPinit, the corresponding clock frequency PPclk and PRR floorplan PPplace[], the maximum number of partition pins required PPmax, the connectivity information C, the initial temperature T0, the random number of iterations Irand performed to calculate the initial temperature, and the number of iterations designated for partition pin floorplans Ipp. The initial temperature for partition pin floorplan is calculated similar to our DAPR PRR floorplanner algorithm. A random number of partition pin placements are performed, then the corresponding average clock frequency change Δavg is computed, and finally, T0 is calculated using Equation 5-1.

Lines 3-11 initialize the partition pin floorplanner algorithm. Line 3 creates the current design iteration number Icurr and current best design iteration number Ibest. Line 4 creates lists to store the current design's clock frequency Clkfq[] and bitstream sizes Bit_size[],respectively. Line 5 creates lists to store the initial floorplan solution E[], the best floorplan solution E_best[], and the new floorplan solution E_new[]. A perturbation function creates E_new[] by altering E[]. Line 6 initializes swp to PPmax, which is used to select the different perturbation methods. Line 7 initializes the solution acceptance probability P to a high value of 0.99, a recommended temperature reduction rate, λ of 0.85, and the temperature T to the initial temperature T0. Lines 8-11 initialize E[] to the initial solution PPinit[], E_best[] to E[], Uphill, Reject, MT to '0', and the initial perturbation M to '1'.

Lines 12-58 drive partition pin floorplanning exploration. A perturbation function (lines 15-39) explores new partition pin floorplans, then the corresponding candidate PR design is generated (lines 40 and 41), and accepted or rejected with probability P (line 47- 54). P is derived using Equation 5-2.

The perturbation function explores new partition floorplans E_New[] using E[] according to the value of swp, which regulates the perturbation move value M. If swp is greater than 0, swp modifies the current partition pin floorplan by performing swaps between existing partition pin placements for each respective PRR, and decrements swp (line 17-26). If swp equals 0, M is set to '4', and the perturbation function places partition pins around the PRRs randomly, while resetting the value of swp back to PPmax (lines 27-29). Lines 31-39 set the value of M by comparing the current value of PPmax and swp where, if swp≥(2*PPmax/3), the input partition pins are randomly swapped, if swp < (2 * PPmax / 3) and swp > (PPmax / 3), the output partition pins are randomly swapped, and if swp < (PPmax/3) and swp ≥ equals 0, the input and output partition pins are randomly swapped interchangeably. We use this swapping method to provide ample variation in the swapping mechanism, but not too much variation such that the design solution space size varies exponentially by 2PPmax.

After the candidate PR design with the new partition pin floorplan is generated, the design is evaluated according to partition pin floorplanner cost function. The partition pin floorplanner cost function calculates the clock frequency change between the new partition pin floorplan E_New[] and the current partition pin floorplan E[], and records the value in Δcost_chnge (line 46). If Δcost_chnge < 0 (line 47), the new partition pin floorplan is accepted only if the new floorplan has the fastest clock frequency as

compared to all prior-evaluated designs, and the new partition pin floorplan is accepted as the best partition pin floorplan and written to E_Best[] (lines 50-52). Alternatively, if the new partition pin floorplan is uphill, the new partition pin floorplan is accepted with probability P and stored in E[] (lines 47-49). Initially, the acceptance probability P is close to 1 during high T values, but decreases with decreasing T. T is decreased in line 56 with the recommended temperature reduction rate of $\lambda = 0.85$ [12], if at each given temperature the total number of uphill moves Uphill or the total number of moves MT exceeds Pmax or 2*Pmax, respectively, (line 55).

At the end of the partition pin floorplanner algorithm (line 58), the best clock frequency Clkfq[Ibest] and the corresponding iteration number Ibest have been determined, in addition to a Pareto-optimal set of PR designs that trade off clock frequency and total partial bitstream size, which are outputted with Pareto(). By default, all generated candidate PR designs are stored in the DAPR tool directory and any particular design number's floorplan and corresponding bitstreams (e.g., Ibest) can be easily retrieved.

5.4 DAPR Floorplanning Tool Evaluation

The DAPR floorplanning tool evaluation is divided into two sections. Section 5.4.1 describes the test environment and associated test designs, and Section 5.4.2 discusses the results obtained.

5.4.1 Experimental Setup

We executed all experiments on a desktop PC with an Intel® Core ™ 2 Duo E6750 2.66 GHz CPU and 3.24 GB of RAM, and all experiment bitstreams were generated for the Virtex-5 LX110T FPGA test device.

For our experiments, we evaluated the DAPR tool floorplanner SA algorithm's convergence rate using two different PR designs with vastly different floorplanning design space sizes. We analyzed the convergence rate of our DAPR tool floorplanner SA algorithm's cost function with respect to the maximum number of iterations *Imax* the DAPR tool performed to improve on the initial cost. Since our algorithm's cost function measures the clock frequency change between successive PR design floorplans, analyzing the convergence rate demonstrates how quickly our DAPR tool floorplanner improves the initial PR design floorplan with respect to the best PR design floorplan found within *Imax* iterations.

The PR designs used were a data processing (DP) PR design with three PRRs and a JPEG coding/encoding (Codec) PR design [Yousuf et al. 2009] with eight PRRs. The DP PR design is a simplified DP PR design derived from an adaptive triple modular redundancy PR design described in [Yousuf et al. 2011] and can be leveraged for processing different data types. The DP PR design's PRRs were designed such that the PRRs did not require special resources to make the DP PR design generic and portable across different devices. In our case, we used the DP PR design to process fast Fourier transform (FFT) [Yousuf et al. 2009] and coordinate rotation digital computer (CORDIC) [Yousuf and Gordon-Ross 2009] data on our test device. Figure 5-5 depicts the FFT/CORDIC DP PR design, where a 1K point FFT PRM and 32-bit CORDIC PRM were assigned to PRR A, a 512 point FFT PRM and 16-bit CORDIC PRM were assigned to PRR B; and a 256 point FFT PRM and 8-bit CORDIC PRM were assigned to PRR C. The FFT and CORDIC PRMs were generated using Xilinx's core generation tool [Xilinx UG994 2014] and the PRM-to-PRR assignments was chosen logically

considering the least amount of internal fragmentation. The FFT/CORDIC DP PR design operated as follows: FFT/CORDIC data from the external memory was transferred over the IO interface to the data flow controller module, the data flow controller module directed the external data for processing to either an FFT PRM or CORDIC PRM depending on the data type, the PRR processed the external data and output the processed data to the data flow controller, and finally, the dataflow controller output the processed data to IO interfaces to be written to external memory.

The JPEG Codec PR design is a pipelined design and performed image data encoding and decoding by operating either in the encoding mode and decoding mode respectively. In the encoding mode, the JPEG Codec PR design read raw image data from the external memory, encoded the data into JPEG format, and wrote the encoded data into external memory. The decoding mode is similar, except encoded JPEG image data is read from the external memory, the data is decoded, and the raw image data is written to the external memory. Encoding mode can be executed by loading the PRMs required for encoding into the respective assigned PRRs, and decoding mode can be executed by loading the PRMs required for decoding into the respective assigned PRRs. Figure 5-6 depicts the JPEG Codec PR design operating in the encoding mode. The JPEG Codec PR design's PRM-to-PRR assignments were as follows: the RGB2YCbCr and FDCT2D PRMs, and the YCbCr2RGB and IDCT2D PRMs were assigned to PRR A; the ZigZag and Reorder PRMs were assigned to PRR B; the Quantizer and Dequantizer PRMs were assigned to PRR C; the Run Length Encoder and Run Length Decoder PRM were assigned to PRR D; the Byte Stuffer and Byte Stripper PRMs were assigned to PRR E; the Header Generator and Header Decoder

PRMs were assigned to PRR F; the Huffman Encoder and Huffman Decoder PRMs were assigned to PRR G; and the Encoder Controller and Decoder Controller PRMs were assigned to PRR H. We refer the reader to [Yousuf and Gordon-Ross 2009] for JPEG Codec PR design's operational details.

Intuitively, the JPEG Codec PR design with more PRRs should have a much larger PRR design space as compared to the FFT/CORDIC DP PR design. However, the JPEG Codec's PRR design space was actually smaller, due to three factors. First, since the JPEG Codec PR design's PRRs occupied a much larger percentage (80%) of the device in comparison to the FFT/CORDIC DP PR design (40%), not all PRR placement permutations of the JPEG Codec PR design were valid, which resulted in a smaller PRR design space. A straightforward example of an invalid PRR placement permutation occurs when a PR design's PRR floorplan is created with all PRRs placed horizontally or vertically with respect to each other, which results in a PRR floorplan crossing the device's horizontal or vertical resource boundary and thus resulting in an invalid PRR placement permutation and floorplan. Alternately, all of the FFT/CODEC PR design's PRR placement permutations were valid. Second, since the JPEG Codec's PRRs occupied a larger percentage of the device, there were fewer PRR floorplan starting locations, and since, the FFT/CORDIC DP PR design occupied a smaller percentage of the device and there was more flexibility in the PRR floorplanning. Third, the JPEG Codec PR design's PRR design space is reduced due to special DSP resource requirements by three of the eight JPEG Codec PR design's PRRs. Specialized resource requirements can constrain PRRs' floorplans to specific locations

in the device thus reducing the overall PRR design space. The FFT/CODEC DP PR design's PRR floorplans were uninhibited due to no specialized resource requirements.

Considering the PRR design space reductions, the FFT/CORDIC DP PR design's PRR design space was approximately 62 times larger than the JPEG Codec PRR design space—approximately 44,640 and 720 valid PRR floorplans, respectively.

5.4.2 Results and Analysis

Table 5-1 shows the FFT/CORDIC DP and JPEG PR designs' clock frequency convergence rates with respect to the highest achievable clock frequency over Imax iterations and associated DAPR tool automated exploration time for 10 different test cases. Each test case had a different combination of Imax and Ipp. We chose Imax to ensure that the DAPR tool exploration time was limited to a reasonable PR design exploration time of a few days, and we chose Ipp to observe the overall effects of the DAPR tool partition pin floorplanning algorithm on the highest achievable clock frequency. Ipp was set in the range of 10% to 40% of Imax to provide the DAPR tool sufficient partition pin floorplanning exploration time but not too time much since the DAPR tool's floorplanning algorithm should run the PRR floorplanner longer than the partition pin floorplanner. The DAPR tool's automated exploration time for each test case was recorded internally by the DAPR tool using the PC desktop's system clock and reported at the end of DAPR tool's execution.

As expected, the convergence rate results revealed that the FFT/CORDIC DP PR design with a larger PRR design space had a lower average convergence rate as compared to the JPEG PR design with a smaller PRR design space. Analyzing the FFT/CORDIC DP PR design's convergence rate further revealed that floorplan perturbations that moved the PRR floorplan towards the device's global clocking

resources [Xilinx UG347 2011] in the center of the device typically resulted in PR designs with higher clock frequencies. Since the FFT/CORDIC PR design was built using a generic DP PR design with no special resource requirements, we logically concluded that the PR designs with no special resource requirements floorplanned near the global clocking resources for similar PR designs will typically result in higher clock frequency PR designs. Analyzing the JPEG PR design's convergence rate revealed that PRR permutations that floorplanned adjacent pipelined PRRs next to each other and the overall PRR floorplan near the global clocking resources typically resulted in a higher clock frequency designs.

We analyzed the DAPR tool's execution time compared to the average number of successful iterations required before the results showed diminishing returns. To determine this number of iterations, we compared the current iteration's clock frequency with respect to the highest clock frequency found within the current number of successful iterations performed for all of the test cases. Figure 5-7 and Figure 5-8 depict the current iteration's clock frequency and current highest clock frequency found, respectively, versus the number of successful iterations for the FFT/CORDIC DP PR design (A) and the JPEG Codec PR design (B) for test cases 1, 2 and 3, which are representative of all other test cases since these test cases revealed similar results. Figure 5-7 shows the variation in the current iteration's clock frequency and Figure 5-8 shows the convergence of the current best PR design's clock frequency.

As expected, Figure 5-8 shows that the greatest PR floorplan improvements occurred during the first several successful iterations, therefore a PR designer can perform rapid design prototyping in only a few DAPR tool successful iterations (i.e.,

Imax approximately between 12-20 on average). If further improvements are desired, a PR system designer could specify a larger Imax value, however, larger Imax values increases the DAPR tool execution time significantly. As shown in Table 5-1, the total DAPR tool execution for the largest value of Imax = “500” was 80 and 125 hours for the DP PR design and JPEG PR designs, respectively. Although, the DAPR tool runtime is significant for large Imax values, the DAPR tool runtime is completely automated with no PR designer intervention necessary. A comparable manual design space exploration process would require performing manual floorplanning and manual invoking the Xilinx PR flow steps, which would require nearly constant PR designer effort during design space exploration, which the DAPR tool eliminates.

Even though setting a larger Imax value has a higher probability of achieving PR floorplan improvements, a faster method would be to execute multiple instantiations of the DAPR tool in parallel. Executing multiple instantiations increases the overall PR design space explored within Imax iterations and the more instantiations of DAPR tool are executing, the greater probability a better PRR floorplan will be found.

Finally, the DAPR tool also outputs the Pareto-optimal set of a PR designs that tradeoff average partial bitstream size and clock frequency. The Pareto-optimal set enables PR system designers to quickly evaluate different design tradeoffs based on design goals/constraints/requirements while examining only a small set of potential Pareto-optimal designs. As an example, Figure 5-9 illustrates the DP PR design’s test case 1’s clock frequency verses the average partial bit stream size for all successful PR designs with the Pareto-optimal designs highlighted (square design points). For this example, only 9% of the design space constitutes Pareto-optimal designs, thus

significantly reducing the number of PR designs a PR designer must consider. Other test cases can also be evaluated in a similar fashion.

Table 5-1. FFT/CORDIC DP and JPEG PR designs' clock frequency convergence rates with respect to the highest achievable clock frequency over Imax iterations and associated DAPR tool automated exploration time for 10 different test cases

			Clock Frequency Convergence Rate with Respect to Highest Achievable Clock Frequency within Imax Iterations (%)		DAPR Tool Automated Exploration Time (Hours)	
Pair No.	Ipp	Imax	FFT/CORDIC DP PR Design	JPEG PR Design	FFT/CORDIC DP PR Design	JPEG PR Design
1	10	100	3.81	2.18	16	24
2	20	100	0.128	5.02	16	24
3	30	100	1.45	6.8	16	24
4	40	100	5.64	9.75	16	24
5	20	200	2.68	10.8	30	46
6	40	200	3.51	11.8	30	46
7	60	200	6.94	4.7	30	46
8	50	300	5.25	15.7	50	75
9	100	300	3.35	8.5	50	75
10	100	500	2.11	11.6	80	125
Convergence Rate =			3.4868	8.685		

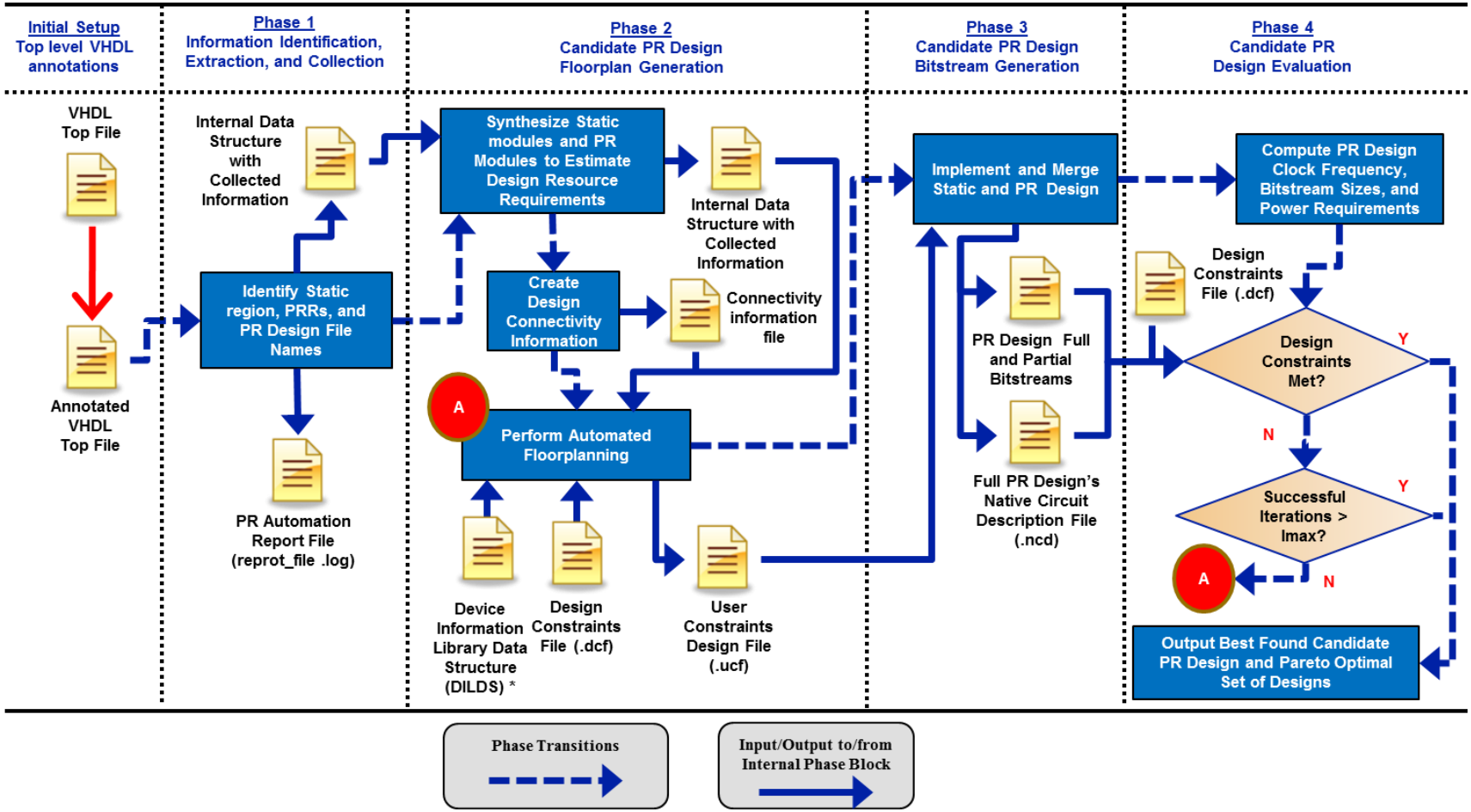

Figure 5-1. The four DAPR tool phase

```
partVIRTEX XC5VLX110Tff1136 NA.die xc5vlx110tff1136.pkg \
    NCLBROWS=160 NCLBCOLS=54 STYLE=VIRTEX5 \
    IN_FF_PER_IOB=1 OUT_FF_PER_IOB=1 \
    NPADS_PER_ROW=1 NPADS_PER_COL=2 \
    NFRAMES=23680 NBITSPERFRAME=1312 \
    NIOBS=882 NBIOBS=640 \
    SLICES_PER_CLB=2 \
    NUM_BLK_RAMS=148 \
    NUM_BLK_RAM_COLS=5 BLK_RAM_COL_0=4 BLK_RAM_COL_1=14 BLK_RAM_COL_2=36
  BLK_RAM_COL_3=46 BLK_RAM_COL_4=54 \
    BLK_RAM_SIZE=32768x1 BLK_RAM_SIZE=16384x2 BLK_RAM_SIZE=8192x4
  BLK_RAM_SIZE=4096x9 BLK_RAM_SIZE=2048x18 BLK_RAM_SIZE=1024x36
  BLK_RAM_SIZE=512x72 \
    NUM_SEL_RAMS_PER_SLICEM=8 NUM_SEL_RAMS_PER_SLICEL=0 \
    SEL_RAM_SIZE=64x1 SEL_RAM_SIZE=32x1 SEL_RAM_SIZE=16x2 SEL_RAM_SIZE=16x1 \
```

Figure 5-2. Virtex-5 LX110T FPGA PARTGen generated partlist.xct file snippet

```
Input:  S, R, N, Ipp, Imax, DILDS
Output: Highest clock frequency PR design and corresponding PR design number
Icurr, Ibest ← 0;
Clk_freq_list[], Bit_size[];
E[], E_best[], E_new[]; #Initial solution, #Best solution, New solution after perturbation;
X_global, Y_global ← 0; #Initial PRR floorplan starting location
Int_frg_x ← 0; #Extra horizontal resources allocated within regions (internal fragmentation)
Int_frg_y ← 0; #Extra vertical resources allocated within regions (internal fragmentation)
White_space ← 0; #Extra resources allocated between regions
Rand_range ← 20, Rand_min ← 5; #Controls the range for randomized PRR placements
Irand ← int(rand(Rand_range)) + Rand_min;
Successful_iter, Error_count ← 0;
M ← 1;
E[] ← Norm_Polish(N); #12V3V4V. . .NV
Do
        User constraints file ← PRR_Placement(S, R, E[], X_global,Y_global, DILDS);
        Successful_iter ← Bit_Generation(.ucf); # Returns 0 if successful
        If Successful_iter == 0 then
                Icurr++;
                Clk_freq_list [], Bit_size[]  ← Design_Evaluation(.ncd, .bit); #Evaluate design
                Error_count = 0;
        Else
                Error_count++;
        End if
        If Error_count == 5 then #After 5 consecutive errors increase extra resource allocation
                Int_frg_x++;
                Int_frg_y++;
                White_space ++;
                Error_count = 0;
        End if
        Select_Perturbation(M);
        Case M
                M1: Select two adjacent operands e_i and e_j, E_new[] = Swap(E, e_i, e_j);
                M2: Select a nonzero length chain C; E_new[] = Complement(E[], C);
                M3: Done = False
                        while !(done) do
                            Forward Swap: Select two adjacent operand e_i and operator e_i+1;
                            if (e_i-1 != e_i+1) and (2N_i+1 < i) then done = TRUE;
                            Backward Swap: Select two adjacent operator e_i and operand e_i+1;
                            if (e_i != e_i+2) then done = TRUE;
                        End while
                            E_new [] = Swap(E, e_i, e_i+1);
                M4: Change PRR floorplan starting locations X_global, Y_global;
        End case
        If (iteration % 2*N ==0) then
                M ← 4;
        Else
                M ← Rand(4);
        End if
Until (Icurr = Irand)
If (Icurr = Irand) Then
```

Figure 5-3. DAPR tool PRR floorplanner algorithm

```
52        Δavg ← Average_Cost(Clk_freq_list []);
53        P ← 0.99;
54        λ ← 0.85;
55        E[] ← Norm_polish(S); #12V3V4V. . .nV
56        E_best[] ← E[];
57        Uphill,Reject,MT ← 0;
58  T ← T0 ← Δavg/ln (P);
59  End if
60  Do
61        Uphill, Reject, MT ← 0;
62        Do
63              Select_Perturbation(M);
64              User constraints file ← PRR_Placement(S, R, E[], X_global,Y_global, DILDS);
65              Successful_iter ← Bit_Generation(.ucf);
66              If Successful_iter == 0 then
67                    Icurr++;
68                    MT++;
69                    Clk_freq_list [], Partial_bit_size[] ← Design_Evaluation(.ncd, .bit);
70                    ΔCost_chnge ← (1/Clk_freq_list [Icurr]) – (1/Clk_freq_list [Icurr-1]);
71                    If (ΔCost_chnge < 0 or rand(1) < e^(-ΔCost_change /T)) then
72                       Begin
73                         If (ΔCost_chnge > 0) then Uphill = Uphill+1;
74                           E[] ← E_new[]; #floorplan accepted
75                         If (Clk_freq_list [Icurr]) > Clk_freq_list [Ibest]) then
76                           E_best[] ← E[];
77                           Ibest = Icurr;
78                         End if
79                       End
80                    Else Reject = Reject +1; #Reject the move
81                       Restore previous iteration X_global, Y_global
82                    End if
83              Else
84                    Error_count++;
85              End if
86              If Error_count == 5 then
87                    Int_frag_x++;
88                    Int_frag_y++;
89                    White_space ++;
90                    Error_count = 0;
91              End if
92              If (iteration % 2*N ==0) then
93                    M ← 4;
94              Else
95                    M ← Rand(4);
96              End if
97        Until (uphill > N) or (MT > 2N)
98        T ← λ * T;
99  Until (Icurr >= (Imax-Ipp))
100 Return Clk_freq_list[Ibest], Ibest
```

Figure 5-3. Continued

```
Input: PPinit, PPclk, PPplace[], PPmax, C, T0, Irand, Ipp
Output: Highest clock frequency PR design and corresponding PR design number
Icurr, Ibest ← 0;
clkFq[], Bit_size[];
E[], E_best[], E_new[]; #Initial solution, #Best solution, New solution after perturbation;
swp ← PPmax;
P ← 0.99;  λ ← 0.85; T ← T0;
E[] ← PPinit;
E_best[] ← E[];
Uphill,Reject,MT ← 0;
M ← 1;
Do
        Uphill, Reject, MT ← 0;
        Do
            Select_Perturbation(M);
            Case M
            M1:
                Select two adjacent input partition pins, ei and ej, E_new[] = Swap(E, ei, ej);
                swp--;
            M2:
                Select two adjacent output partition pins, ei and ej, E_new[] = Swap(E, ei, ej);
                swp--;
            M3:
                Select two adjacent input partition pins, ei and ej, E_new[] = Swap(E, ei, ej);
                Select two adjacent output partition pins, ex and ey, E_new[] = Swap(E, ex, ey);
                swp--;
            M4:
                Randomly place all partition pins across using the PRR using PPplace[];
                swp ← PPmax;
            End case
        If (swp => 2PPMax/3) then
                M = 1;
        Else if ((swp < 2PPMax/3) and (swp>(PPmax/3))
                M = 2;
        Else if (swp > 0 and (swp < (PPmax/3))
                M = 3;
        Else
                M = 4;
        End if
        Write E_New[] floorplan constraints to ucf file using connectivity information C;
        Bitstream_Generation(.ucf);
        if PARfail == 0 then
                DesignEvaluation();
                Icurr++;
                MT++;
                Δcost_chnge ← ((1/clkFq[Icurr]) – (1/clkFq[Icurr-1]));
                if (Δcost_chnge < 0 or rand(1) < e^(-Δcost_chnge/T)) then
                        if (Δcost_chnge > 0) then Uphill ← Uphill+1;
                                E[] ← E_new[]; #floorplan accepted
                        if (clkFq [Icurr] > clkFq [Ibest]) then
                                E_best ← E;
```

Figure 5-4. DAPR tool partition pin floorplanner algorithm

```
52                                    Ibest = Icurr;
53                              else
54                        Reject ← Reject+1;  #floorplan rejected
55          Until (uphill > N) or (MT > 2N)
56          T ← λ * T;
57  Until (Icurr >= (Ipp-PPirand))
58  return clkFq[Ibest], IBest, Pareto()
```

Figure 5-4. Continued

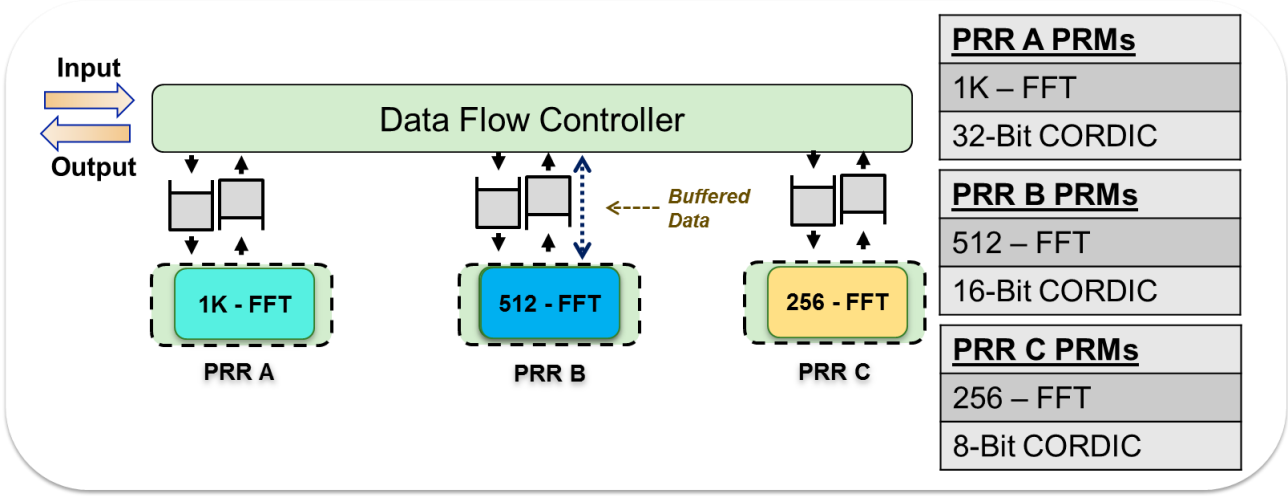

Figure 5-5. FFT/CORDIC DP PR design showing PRRs A, B, and C and associated PRMs

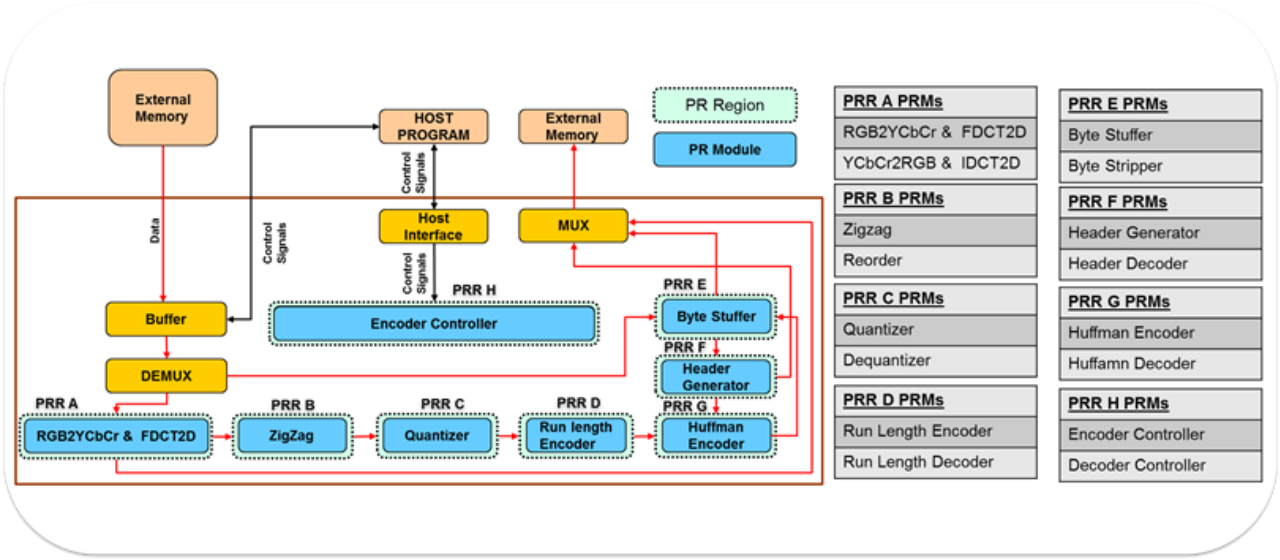

Figure 5-6. JPEG Codec PR design's encoding mode with PRRs A-H and associated PRMs

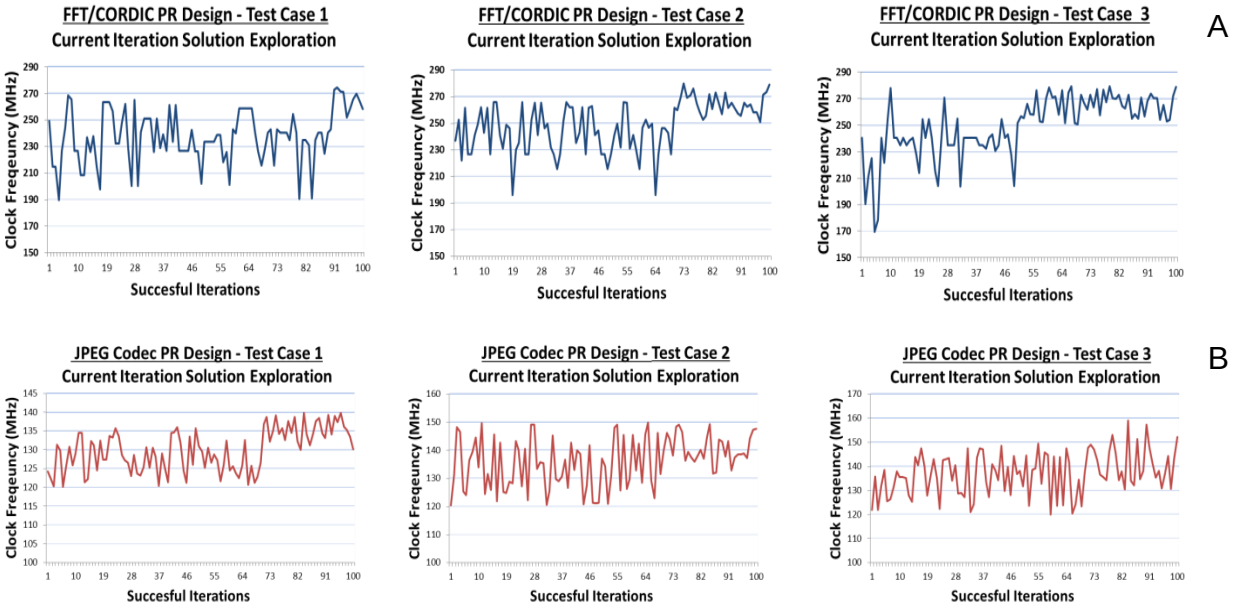

Figure 5-7. Current iteration's clock frequency versus number of successful iterations for three test cases. A) FFT/CORDIC DP PR design. B) JPEG Codec PR design

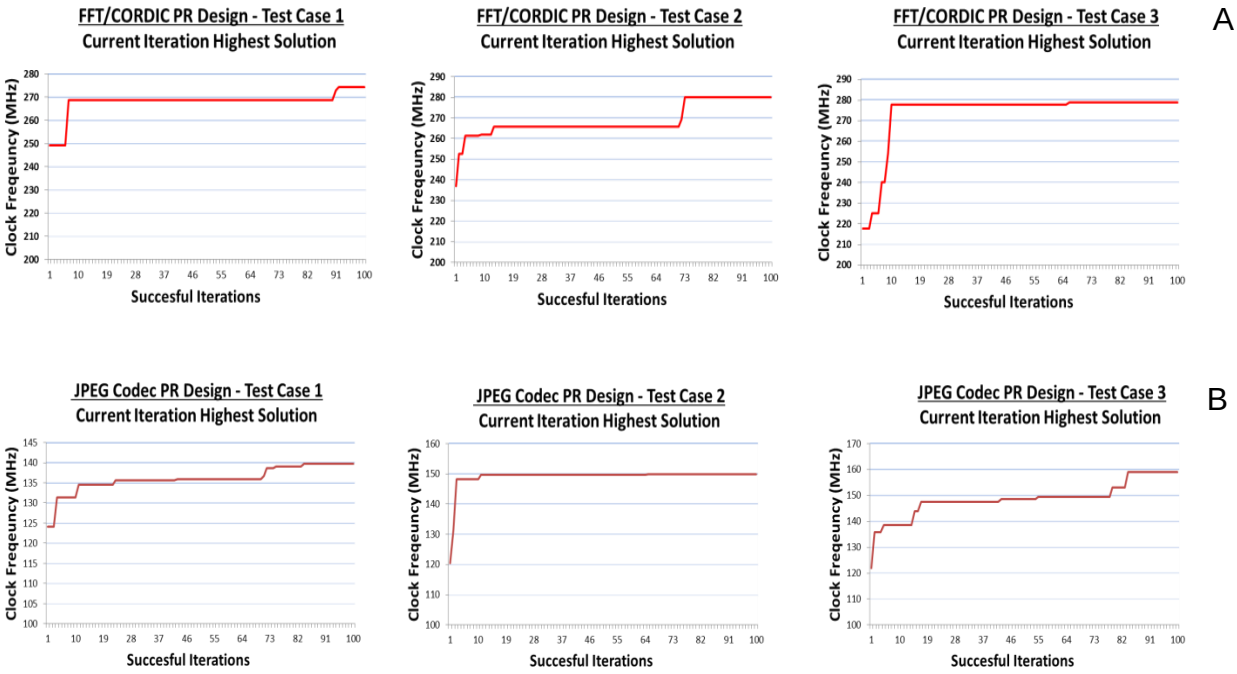

Figure 5-8. Current iteration's highest clock frequency versus number of successful iterations for three test cases. A) FFT/CORDIC DP PR design. B) JPEG Codec PR design

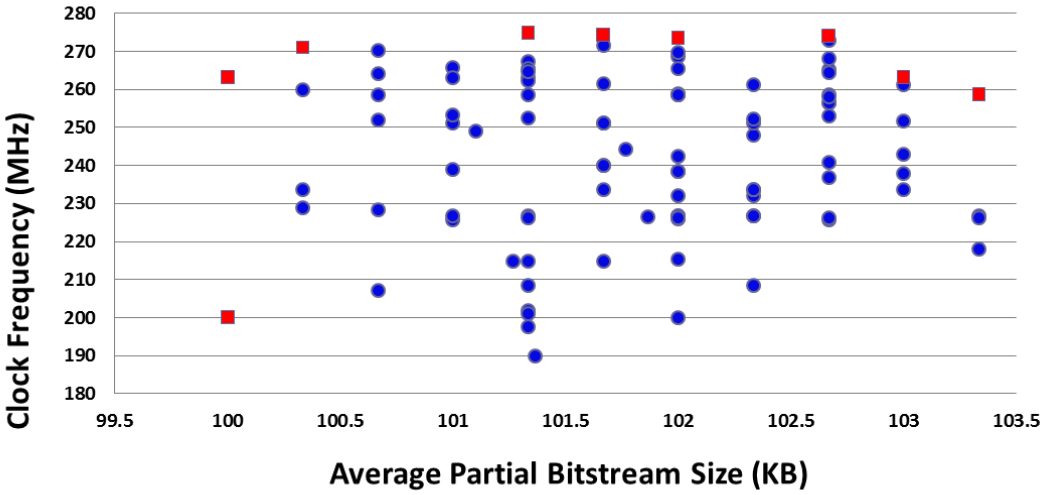

Figure 5-9. Clock Frequency versus average partial bitstream size for FFT/CORDIC DP PR design's test case 1. Square design points designate Pareto-optimal designs

CHAPTER 6
CONCLUSIONS

In this dissertation, we presented and evaluated a novel partially reconfigurable (PR) design flow and associated tools and architectures to make PR system design more amenable to a wide range of system designers. We leveraged PRs flexibility and automation techniques to iteratively improve a PR design's performance in order to design efficient PR-based HW/SW co-designed embedded systems.

In Chapter 1, we introduced the concept of PR and associated benefits achievable by leveraging PR for modern embedded system design. We also introduced the importance of HW/SW co-design in embedded system design and discussed the various challenges associated with leveraging PR during HW/SW co-design. The various challenges involved provided the motivation for easing PR for system designers.

In Chapter 2, we presented the background on current vendor supported PR design flow, discussed related work in PR design flow automation, and highlighted the contributions of our work as compared to prior work.

In Chapter 3, we analyzed Xilinx's partial reconfiguration (PR) design flow and discussed the associated performance critical steps involved and proposed the design automation for partial reconfiguration (DAPR) design flow. The DAPR design flow automates the two performance critical steps involved during HW/SW co-design of PR systems: HW/SW PR design partitioning, and PR design floorplanning. In Chapter 3, we also presented the target architecture for the DAPR design flow and associated architectural details.

In Chapter 4 we presented our automated HW/SW PR partitioning methodology, which performs an initial analysis on a given application and exhaustively determines all

possible HW/SW combinations for a particular configuration to generate the corresponding maximum resource requirements, reconfiguration time, and total system execution time for each partition.

In Chapter 5 we presented our automated floorplanning methodology, which performs automated PR design space exploration to improve the clock frequency of a PR design by iteratively exploring the PR design space using a simulated annealing (SA)-based algorithm. Floorplanning results show that the greatest clock frequency improvements are achieved during first several iterations (12-20 iterations on average).

The DAPR design flows isolates PR designers from low-level PR design complexities involved during the PR design process. This isolation reduces the overall PR design time effort and makes PR designs more amenable to PR designers. The DAPR design flow also enables PR design flexibility by allowing system designers to choose from a list of possible HW/SW co-designed PR design partitions that tradeoff reconfiguration time and hardware resource requirements. After a partition has been chosen, the provided DAPR floorplanning tool can be leveraged to improve the clock frequency of the chosen PR design. This PR design flexibility enables a system designer to easily choose a target that will satisfy system designer goals.

The DAPR design flow's key contributions include: making PR designs more accessible to a wider range of designers; facilitating rapid design prototyping; and creating high-performance systems with reduced design time effort.

Even though DAPR provides a holistic solution for HW/SW PR co-design, future work could investigate optimizations to the DAPR tool floorplanning algorithm with respect to different PR design types. For example, our analysis revealed that the

highest PR design clock frequency improvements occurred when the PR design floorplan moved towards the global clocking resources. Also, for pipelined PR designs, the highest clock frequency improvements occurred when pipelined partially reconfigurable regions (PRRs) where placed next to each other. Our findings warrant further investigation on the effects of PR design floorplanning with respect to PR designs, and this study will consequently provide additional PR design floorplanning recommendations to PR system designers. Future work could also test and extend the DAPR design flow support to all PR-enabled Xilinx devices, and incorporate the DAPR design flow with Altera's PR design flow.

LIST OF REFERENCES

Arato, P.; Juhasz, S.; Mann, Z.A.; Orban, A.; Papp, D., "Hardware-software partitioning in embedded system design," Intelligent Signal Processing, 2003 IEEE International Symposium on , vol., no., pp.197,202, 4-6 Sept. 2003

Banerjee, P.; Sangtani, M.; Sur-Kolay, S., "Floorplanning for Partial Reconfiguration in FPGAs," VLSI Design, 2009 22nd International Conference on , vol., no., pp.125,130, 5-9 Jan. 2009

Banerjee, P.; Sur-Kolay, S.; Bishnu, A., "Floorplanning in Modern FPGAs," VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on, vol., no., pp.893,898, 6-10 Jan. 2007

Beckhoff, C.; Koch, D.; Torresen, J., "Go Ahead: A Partial Reconfiguration Framework," Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on , vol., no., pp.37,44, April 29 2012-May 1 2012

Byungil Jeong; Sungjoo Yoo; Lee, Sunghun; Kiyoung Choi, "Hardware-software cosynthesis for run-time incrementally reconfigurable FPGAs," Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific , vol., no., pp.169,174, 9-9 June 2000

Bolchini, C.; Fossati, L.; Codinachs, D.M.; Miele, A.; Sandionigi, C., "A Reliable Reconfiguration Controller for Fault-Tolerant Embedded Systems on Multi-FPGA Platforms," International Conference on Defect and Fault Tolerance in VLSI Systems (DFT), Oct. 2010

Chen, F.; Craymer, L.; Deifik, J.; Fogel, A.J.; Katz, D.S.; Silliman, A.G., Jr.; Some, R.R.; Upchurch, S.A.; Whisnant, K., "Demonstration of the remote exploration and experimentation (REE) fault-tolerant parallel-processing supercomputer for spacecraft onboard scientific data processing," International Conference on Dependable Systems and Networks (DSN), June 2000.

Cheng, L.; Wong, M.D.F., "Floorplan design for multi-million gate FPGAs," Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on , vol., no., pp.292,299, 7-11 Nov. 2004

Claus, C.; Stechele, W.; Kovatsch, M.; Angermeier, J.; Teich, J., "A comparison of embedded reconfigurable video-processing architectures," Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on , vol., no., pp.587,590, 8-10 Sept. 2008

Conger, C.; Gordon-Ross, A.; George, A., "Design Framework for Partial Run-Time FPGA Reconfiguration," Engineering of Reconfigurable Systems & Algorithms (ERSA), July 2008.

Craven, S., Athanas, P., “Dynamic hardware development,” Int. J. Reconfigurable Comput., 2008, article id 901328

Di Carlo, S.; Prinetto, P.; Scionti, A., "A FPGA-Based Reconfigurable Software Architecture for Highly Dependable Systems," International Conference on Asian Test Symposium (ATS), Nov. 2009

Feng, Y.; Mehta, D.P., "Heterogeneous floorplanning for FPGAs," VLSI Design, 2006. Held jointly with 5th International Conference on Embedded Systems and Design., 19th International Conference on , vol., no., pp.6 pp.,, 3-7 Jan. 2006

Hentati, M.; Aoudni, Y.; Nezan, J.-F.; Abid, M.; Deforges, O., "FPGA dynamic reconfiguration using the RVC technology: Inverse quantization case study," Design and Architectures for Signal and Image Processing (DASIP), 2011 Conference on , vol., no., pp.1,7, 2-4 Nov. 2011

Huang, J.; Parris, M.; Lee, J.; Demara. R., “Scalable FPGA-based architecture for DCT computation using dynamic partial reconfiguration,” ACM Trans. Embed. Comput. Syst. 9, 1, Article 9 (October 2009), 18 pages.

Gonzalez, I.; Gomez-Arribas, F.J.; Lopez-Buedo, S., "Hardware-accelerated SSH on self-reconfigurable systems," Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on , vol., no., pp.289,290, 11-14 Dec. 2005

Jara-Berrocal, A.; Gordon-Ross, A., "VAPRES: A Virtual Architecture for Partially Reconfigurable Embedded Systems," Design, Automation & Test in Europe Conference & Exhibition (DATE), March 2010

Kao, C., “Benefits of Partial Reconfiguration”, Xcell Journal, Fourth Quarter 2005, pp. 65 - 67.

McDonald, E.J., "Runtime FPGA Partial Reconfiguration," Aerospace Conference, 2008 IEEE , vol., no., pp.1,7, 1-8 March 2008

Mei, B.; Schaumont, P.; Vernalde, S., ”A hardware-Software Partitioning and scheduling algorithm for dynamically reconfigurable embedded systems,”ProRisc workshop on Ckts, Systems and Signal processing, Nov 2000.

Mesquita, D.; Moraes, F.; Palma, J.; Möller, L.; Calazans, N., "Remote and partial reconfiguration of FPGAs: tools and trends," Parallel and Distributed Processing Symposium, 2003. Proceedings. International , vol., no., pp.8 pp.,, 22-26 April 2003

Ming Liu; Kuehn, W.; Zhonghai Lu; Jantsch, A., "Run-time Partial Reconfiguration speed investigation and architectural design space exploration," Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on , vol., no., pp.498,502, Aug. 31 2009-Sept. 2 2009

Montone, A., Santambrogio, M., Sciuto, D., Memik, S., “Placement and Floorplanning in dynamically reconfigurable FPGAs,” ACM Transactions on Reconfigurable Technology and Systems 3(4), 24:11-24:34 Nov 2010

Perumal, D.U.; Kumar, S.A.; Prasanth, S.; Kumar, P.V.; Kannan, M.; Vaidehi, V., "An Efficient Reconfigurable Image Compression Architecture," Signal Processing, Communications and Networking, 2007. ICSCN '07. International Conference on, vol., no., pp.265,269, 22-24 Feb. 2007

Rabozzi, Marco; Lillis, John; Santambrogio, Marco D., "Floorplanning for Partially-Reconfigurable FPGA Systems via Mixed-Integer Linear Programming," Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on , vol., no., pp.186,193, 11-13 May 2014

Sait, S. M.; Youssef, H., “VLSI Physical Design Automation: Theory and Practice”, World Scientific Publishing Company, 1st edition, 1999.

Singhal, L.; Bozorgzadeh, E., “Multi-layer Floorplanning on a Sequence of Reconfigurable Designs,” in FPL, 2006, pp. 1–8

Srinivasan, V.; Radhakrishnan, S.; Vemuri, R., "Hardware software partitioning with integrated hardware design space exploration," Design, Automation and Test in Europe, 1998., Proceedings , vol., no., pp.28,35, 23-26 Feb 1998

Stitt, G.; Lysecky, R.; Vahid, F., "Dynamic hardware/software partitioning: a first approach," Design Automation Conference, 2003. Proceedings , vol., no., pp.250,255, 2-6 June 2003

Vipin, K.; Fahmy, S.A., "Efficient region allocation for adaptive partial reconfiguration," Field-Programmable Technology (FPT), 2011 International Conference on , vol., no., pp.1,6, 12-14 Dec. 2011

Vipin, K.; Fahmy, S.A., “Architecture-aware reconfiguration-centric floorplanning for partial reconfiguration,” in Reconfigurable Computing: Architectures, Tools and Applications – Proceedings of the International Symposium on Applied Reconfigurable Computing (ARC), 2012, pp. 13– 25.

Vipin, K.; Fahmy, S.A., "Automated Partitioning for Partial Reconfiguration Design of Adaptive Systems," Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International , vol., no., pp.172,181, 20-24 May 2013

Vipin, K.; Fahmy, S.A., "An Approach to a Fully Automated Partial Reconfiguration Design Flow," Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on , vol., no., pp.231,231, 28-30 April 2013

Vipin, K.; Fahmy, S.A., “Automated partial reconfiguration design for adaptive systems with CoPR for Zynq,” in Proc. Int. Symp. FieldProgrammable Custom Comput. Mach. (FCCM), 2014

Vipin, K.; Fahmy, S.A., "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq." (2014): 1-1.

Wallace, G.K., "The JPEG still picture compression standard," Consumer Electronics, IEEE Transactions on , vol.38, no.1, pp.xviii,xxxiv, Feb 1992 doi: 10.1109/30.125072

Williams, J.; Bergmann, N.; Hodson, R., “A Linux-based Software Platform for the Reconfigurable Scalable Computing Project”, International Conference on Military and Aerospace Programmable Logic Devices (MAPLD), September 2005.

Williams, J.; Dawood, A.; Visser, S., “Reconfigurable Onboard Processing and Real Time Remote Sensing”, Transactions on Information and Systems (IEICE), May 2003

Xilinx Inc., “Command Line Tools User Guide” UG628 (v14.7) October 2, 2013. Retrieved February 15, 2015. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/devref.pdf

Xilinx Inc. “Device Control Register Bus” DS402 (v 2.9), April 19 2010. Retrieved February 15, 2015. http://www.xilinx.com/support/documentation/ip_documentation/dcr_v29.pdf

Xilinx Inc., “ML505/ML506/M L507 Evaluation Platform User Guide” UG347 (v3.1.2) May 16, 2011. Retrieved February 15, 2015.http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

Xilinx Inc., “Partial Reconfiguration User Guide,” UG702 (v14.5) April 26, 2013. Retrieved February 15, 2015. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug702.pdf

Xilinx Inc., “Virtex-5 Family Overview”, DS100 (v 5.0), Feb. 2009. Retrieved February 15, 2015. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf

Xilinx Inc., “Virtex-5 FPGA Configuration Guide”, UG191 (v 3.11), October 19, 2012. Retrieved February 15, 2015. http://www.xilinx.com/support/documentation/user_guides/ug191.pdf

Xilinx Inc., “Vivado Design Suite User Guide Designing IP Subsystems Using IP Integrator” UG994 (v2014.4) November 19, 2014.
Retrieved February 15, 2015.
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug994-vivado-ip-subsystems.pdf

Yousuf, S.; Gordon-Ross, A., “Partial Reconfiguration for Image Processing Applications,” Military and Aerospace Programmable Logic Device (MAPLD), 2009.

Yousuf, S.; Gordon-Ross, A., “DAPR: Design Automation for Partially Reconfigurable FPGAs,” in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA). CSREA Press, June 2010

Yousuf, S.; Jacobs, A.; Gordon-Ross, A., "Partially reconfigurable system-on-chips for adaptive fault tolerance," Field-Programmable Technology (FPT), 2011 International Conference on , vol., no., pp.1,8, 12-14 Dec. 2011

BIOGRAPHICAL SKETCH

Shaon Yousuf was born in Dhaka, Bangladesh in 1982. He received his Bachelor of Science in electrical and electronic engineering from Ahsanullah University of Science and Technology, Dhaka, Bangladesh in 2005 and his Master of Science in electrical and computer engineering from the University of Florida, Gainesville, Florida, United States, in 2009. He received his Doctor of Philosophy in electrical and computer engineering from the University of Florida, Gainesville, Florida, United States, in 2015. His current research interests are in the area of dynamically reconfigurable systems, design automation, floorplanning, and design partitioning.