OpenPR: An Open-Source Partial Reconfiguration Tool-Kit for Xilinx FPGAs

Ali Asgar Sohanghpurwala

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Peter M. Athanas, Chair

Patrick Schaumont

Mark Jones

December 20, 2010

Blacksburg, Virginia

Keywords: FPGA, partial-reconfiguration, open-source

Copyright 2010, Ali Asgar Sohanghpurwala

OpenPR: An Open-Source Partial Reconfiguration Tool-Kit for Xilinx FPGAs

Ali Asgar Sohanghpurwala

The Xilinx Partial Reconfiguration tool kits have been instrumental for performing a wide variety of research on Xilinx FPGAs. These tool kits provide a methodology for creating rectangular partial reconfiguration modules that can be swapped in and out of a static baseline design with one or more PR slots. This thesis presents a new PR toolkit called OpenPR that, for starters, provides similar functionality to the Xilinx PR tool kits. The distinguishing feature of this toolkit is that it is being released as open source, and is intended to be customizable to the needs of researchers. OpenPR has been designed to be easy to use, extensible, portable, and compatible with a wide range of Xilinx software and devices. Aside from supporting the slot-based PR paradigm, OpenPR also provides a solid base for further research into partial reconfiguration and FPGA productivity oriented design tools.

Dedication

To my mother Nafisa, whose love and guidance will be sorely missed. This work, unworthy as it is, is dedicated to her memory.

Acknowledgments

Thanks to my academic adviser, Dr. Peter Athanas, for the opportunity to work in the Configurable Computing Lab, and for the invaluable advice both academic and otherwise. The CCM Lab is one of the best "work" environments I've experienced, thanks for tolerating my idiosyncrasies and providing an environment where I could thrive.

Thanks to Dr. Mark Jones, on whom I could always rely on for sound and insightful advice on whatever topic I brought to his office, be it Graduate School, Embedded Systems, or even how to do a proper paper review.

Thanks to Dr. Patrick Schaumont whose sharp insight and and sage advice always steered me in the right direction. I also thoroughly enjoyed his class on Secure Hardware Design.

Thanks to my wife Fatema, and my father Aliakbar, whose love and support I can always count on. Without their constant nagging it is unlikely this work would have ever been completed.

I am eternally grateful to Dr. Govind Rao and Dr. Yordan Kostov for taking a chance on me at a time when few others were willing to. The opportunities they presented me at UMBC's

Center for Advanced Sensor Technology reignited my passion for computer engineering and steered my career towards Embedded Systems Design. Thank you for encouraging me to pursue a master's degree.

Thanks to Matt French and Dr. Neil Steiner of Information Sciences Institute East for providing such great learning opportunities in our collaborative projects. The amount I learned from Dr. Steiner about writing development tools, Xilinx FPGAs, and general C++ development practices is incredible.

Further thanks go to Dr. Neil Steiner and Aaron Wood for their contributions to the Torc tool set. Without their work on the Torc logic and wiring databases, OpenPR would not have been possible. Aaron also provided invaluable insight and code for the first iteration of the OpenPR route-blocker.

Thanks to our Torc collaborators from BYU. Their contributions to Torc, and incredibly in-depth knowledge of Xilinx Virtex architectures greatly assisted the development of OpenPR.

Thanks to ISI East and Harris Corporation for funding portions of my research.

Thanks to Dr. Cameron Patterson who has been a veritable treasure trove of information on FPGAs and all things related. I have learned a great deal from him about HDL design, run-time reconfiguration, and Xilinx FPGAs in general.

Thanks to all my CCM Lab colleagues who have helped me along the way. Working with you has been a great privilege, and I hope our paths cross again in the future. Thanks to Tony Frangieh, who has been a true friend, his support and companionship has helped me

greatly during my time in the CCM Lab. Thanks to everyone who has helped me along the way including Dr. Jorge Suris, Dr. Stephen Craven, Matt Shelburne, Adolfo Recio, Charles Irick, Prabhaav Bhardwaaj, and Jacob Couch among others.

Further thanks go to Tony Frangieh for being the first OpenPR guinea pig, and providing his video filter application as a test case. Additionally thanks to Jacob Couch for allowing the use of his XDL Netlist manipulator utility in OpenPR.

Thanks to my employer X-COM Systems/Bird Technologies Group, especially my supervisor Romel Haq, for being so understanding of my oddball schedule as I struggled to finish this work.

Thanks to my friends from my undergraduate career including Ashwin, Idi, Neil, Wesley, Chris, Ammar, and Adnan. You helped me maintain sanity through a grueling Computer Engineering program.

Thanks to all my friends and family who have supported me during dark periods in my life. Your support has kept me going.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

With each generation of Xilinx Field Programmable Gate Arrays (FPGAs), advancements are made in logic density, hard intellectual property (IP) availability, speed and power consumption[1, 2, 3]. However, the predominant design methodology for these chips remains largely unchanged. Typically an engineer will build a monolithic design, simulate to verify functionality, then synthesize the design targeted towards a specific FPGA platform. This process can be very inefficient, especially when minor changes or bug fixes are needed. Development difficulty and lengthy turn-around time has been acknowledged by many as a barrier to more widespread usage of FPGAs[4]. As a result, many configurable computing research projects have been focused on improving development tools. A number of these

projects have been based upon the various versions of Xilinx's partial reconfiguration tool kits. The last free version of these tools was the Xilinx Partial Reconfiguration Early Access Software Tools for ISE version 9.2i (here on referred to as the Xilinx PR Toolkit). With the 12.x ISE series, Xilinx has released a more usable and stable version of the PR Toolkit as a paid license option.

All versions of the Xilinx PR Toolkit operate on the same underlying principles of "slot-based" partial reconfiguration. A "slot" in this case is a fixed geometric region – fixed in size and fixed in position for a selected device. Multiple partial designs can be created that can then populate a given slot one at a time. The partial designs are constrained to only use the resources available within the designed slot. A design can be no bigger than the slot that it targets, and the unused "white space" of a design that does not completely fill the slot cannot be recovered. Also, a core designed for one particular slot cannot be used in another slot on the device. Previous versions of the Xilinx PR Toolkit even required the designer to manually place routing terminals to allow signals to pass between partial modules and the static design. This was a tedious and mistake-prone task.

One feature of the Xilinx PR Toolkits has been support for run-time partial reconfiguration. Run-time partial reconfiguration allows the user to reprogram portions of the FPGA while the rest of the system implemented on the FPGA is still operational. An example of this might be a design implemented on a single FPGA with both a microprocessor and a reconfigurable co-processor[5]. The microprocessor could continue communicating with the outside world while the co-processor functionality is changed. The FPGA reconfiguration in

this case is transparent to the user, with no system down-time.

Many significant research projects have explored new areas in reconfigurable computing by building upon the Xilinx PR Toolkit. These include notable efforts where the details of hardware reconfiguration from the system designer's perspective are abstracted away. An example of this is the work from Fahmy et al. where hardware processing components are made to appear as software components in the run-time system, enabling their inclusion in adaptive applications [6]. Full system autonomy was demonstrated via partial reconfiguration in [7]. Here, the system realizes a significant reduction in size compared to static implementations and an equally impressive reduction in reaction times compared to alternative solutions. In [8], the authors present a partially reconfigurable FPGA-based architecture and methodology for increasing wireless sensor network agility. Their findings show a reduction in power consumption and while an improvement in performance compared to a microprocessor-based system. In [9], the authors show how an FPGA system based on an Open Source OpenRISC 1200 microprocessor can take advantage of partial reconfiguration to perform the secure transfer of the data needed to run an application. The authors of [10] propose a Secure Reconfiguration Controller (SeReCon) that provides secure run-time management of designs downloaded to the dynamically partially reconfigurable Xilinx device as a strategy to protect the design intellectual property.

1.2 Motivation and Objectives

Previous releases of Xilinx's PR Toolkit have been maligned for difficulty of use and the inflexibility of the slot-based model[11]. Even within those rigid confines, however, researchers have shown creativity in their approaches to partial reconfiguration with varied experimental outcomes. Access to a more flexible PR Toolkit might help the community pursue advances in design productivity, autonomous computing, and FPGA fabric reuse. In fact, it would be preferable if modifications could be made to the PR Toolkit itself to better suit the desired application area.

With the goal of providing researchers with an extensible, open-source PR toolkit, work began on the OpenPR project. The initial release of the OpenPR toolkit is nearly functionally identical to the Xilinx PR Toolkit. Being so, it is intended to help create a baseline static design with one or more partial reconfiguration slots, as well as partial modules that fit in those slots. When complete, the designer is provided a bitstream file for the static design along with one or more smaller partial bitstream files for the overlay functions. The distinguishing feature is that it is being released as open source. Because of this alone, it is possible for individual researchers to extend this toolkit to accommodate the specific needs of their project, possibly overcoming limitations imposed by the vendor tools.

To verify that OpenPR's functional objectives were met, several slot-based reconfigurable designs were attempted on several different target platforms. One design implemented run-time-reconfigurable video filters on Xilinx's XUPV5 development board. The static design took a video signal from the boards VGA input, streamed it through the reconfigurable

region, and pushed the filtered signal out through the DVI output. Several swappable video filters were implemented to fit in the reconfigurable slot. This was an important test design as it had previously been implemented with Xilinx's 9.2 PR patch, making it ideal for a comparison. OpenPR compared favorably to the Xilinx 9.2 PR patch with similar compile times, ease of use, and functionality. Further details of this test will be discussed in Chapter 6.

OpenPR differs from previous PR tools like *JBits*[12] in that it does not depend on any Xilinx proprietary information. All information used in the OpenPR toolkit is based on information that has been openly released by Xilinx such as the Xilinx Description Language (XDL) format, user guides, and software manuals. This is important because it means that there are no legal barriers to freely distributing and modifying the toolkit.

Unlike the Xilinx PR tool kits which have generally been tied to a specific ISE release, OpenPR aims to be version-agnostic. This is possible because OpenPR utilizes a common subset of the Xilinx ISE functionality that should not change drastically between releases. Researchers can switch between ISE versions as desired to take advantage of new features or architecture support.

1.3 Thesis Organization

This thesis first offers a short background into hardware acceleration, FPGA design tools, partial reconfiguration, and open-source software in Chapter 2. Then Chapter 3 explores

the various components that make up the toolkit. A detailed step-by-step analysis of how the OpenPR process is used to create a baseline static design and partial modules follows in Chapter 4. Issues of code portability, extensibility, and maintainability are discussed in Chapter 5. Finally, results are presented in Chapter 6 with the conclusion in Chapter 7.

Chapter 2

Background and Related Work

This section provides some basic background on what FPGAs are and how they are used along with a survey of FPGA development tools and other related work.

2.1 FGPA Background

As computing technology advances, engineers endeavor to solve ever more complex problems while simultaneously shrinking our computing devices. This creates new power efficiency and performance requirements resulting in scenarios where traditional general purpose processors (GPPs) prove inadequate. Von Neuman based GPP architectures provide incredible flexibility, allowing them to easily compute almost any problem. The trade-off for this flexibility is that their sequential nature can result in inefficient computation for many data parallel or task parallel applications[13]. In areas such as networking, signal processing,

wireless communications, security, and others, GPPs are often unable to satisfy performance or power-efficiency constraints. For these applications it is often necessary to employ a hardware accelerator. Hardware accelerators can be specialized programmable processors such as digital signal processors (DSPs) and general purpose graphics processing units (GPGPUs), or they can employ custom logic in application specific integrated circuits (ASICs) or configurable computing devices such as FPGAs[14]. These solutions typically trade programming flexibility for increased parallelism, resulting in greater performance and/or power efficiency for certain target applications[15].

Hardware accelerators are typically selected based on the performance requirements, form factor, and power budget for the intended application. DSPs are widely used in embedded systems as they offer good performance in signal processing applications with relatively low power consumption. In applications where power consumption and form factor are less important than overall cost and performance, GPUs are being increasingly employed.

Applications that require the absolute best performance and power-efficiency often require custom digital logic in the form of ASICs or Configurable Computing devices. ASICs offer the best possible performance and power-efficiency. Unfortunately, ASIC designs also have the longest development cycles with increasingly large fabrication costs[16]; a 65nm mask can cost $1.5 million[17]. Configurable computing devices can offer performance approaching that of ASICs with shorter development times and significantly less cost in low-volume applications. FPGAs are fine-grained reconfigurable devices that offer an intermediate option between the flexibility of GPPs and the optimal performance of ASICs. Modern FPGAs are

a collection of Configurable Logic Blocks (CLBs), and various specialized hardware IPs connected with extremely flexible routing architectures. This architecture allows FPGAs to compute a large variety of problems much faster than general purpose hardware while maintaining reprogrammability. The high overhead of the configurable logic and routing in FPGAs means that systems implemented on ASICs will generally perform better and have a lower per-unit cost in high volume applications[16]; however, the shorter design cycle, reconfigurability, and significantly lower initial cost than an ASIC design make FPGAs a popular option for implementing custom logic.

2.2 Traditional FPGA Development Flow

FPGAs can be utilized to produce fast, low-power hardware accelerators at reasonable costs, but developing for them is a complex and difficult process relative to software development. This section provides an overview of the traditional design flow for designs targeted towards Xilinx FPGAs. While based on the Xilinx development tools, this flow should generally apply to other FPGA vendors.

2.2.1 Design Entry

Design entry is one of the most flexible steps in the FPGA design process. While the most prevalent form of design entry is Hardware Description Language (HDL) code, there are many alternatives. These include graphical design entry methods such as schematic, block,

and finite state machine (FSM) diagram creation as well as the incorporation of third-party IP, and automatically generated cores.

2.2.2 Simulation

Verifying design functionality usually involves simulating the entire design with a complex test bench that fully exercises all modules in a design. There are several popular options for FPGA design simulation including tools from Mentor Graphics, Cadence, Aldec, and Xilinx themselves.

2.2.3 Synthesis

The synthesis process takes HDL code or logic schematics and generates files describing the design in terms of generic digital logic elements such as look up tables (LUTs), flip-flops, and logic gates. Synthesis tools are very mature and very good at taking a complex design and optimizing it for area, power, or performance. When synthesis is completed, the Xilinx Synthesis Technology (XST) utility writes the generic netlist to a Native Generic Circuit Description (NGC) file. Third-party synthesis tools such as Synopsys's Synplify usually generate industry standard EDIF Netlist files (.EDN).

2.2.4 Translation

The Translate stage takes synthesized modules (NGC or EDN files), physical macros (.NMC files), as well as any User Constraints (UCF) files and generates a single Native Generic Database (NGD) file. This operation is performed by the Xilinx *NGDBuild* utility. This is generally the last stage where the designer is required to provide any input.

2.2.5 Technology Mapping

Technology mapping transforms a technology-independent netlist into one that only contains logic cells supported by the target FPGA architecture[18]. For Xilinx FPGAs this means converting the generic NGD netlist into one that supports the target architecture's Basic Elements of Logic (BELs) such as look up tables (LUTs) and flip-flops. Xilinx's *MAP* utility reads in the NGD file and maps the contained netlist, producing a Native Circuit Description (NCD) file and a Physical Constraints (PCF) file.

2.2.6 Packing

In modern Xilinx FPGAs, BELs such as LUTs and flip-flops are grouped into Configurable Logic Blocks (CLBs). The process of grouping several LUTs and registers into one logic block is called packing[19]. Timing-driven packing is performed by the Xilinx *MAP* utility after the design has been mapped into BELs. The NCD file is then updated with the packed design.

2.2.7 Placement and Routing

Placement and routing are perhaps the two steps in the design flow with the most impact on timing closure and compile time. With the incredibly large number of logic resources and rich routing architectures of modern FPGAs, place and route tools must reduce a staggering number of implementation choices down to an optimal design that achieves timing closure. Packed logic resources need to be located such that the design is routable, user placement and timing constraints are satisfied, and design rule checks (DRC) are not violated. For the Virtex-4 and earlier devices, the placer was invoked from the Place and Route (*PAR*) utility. Since the release of the Virtex-5 architecture, placement has been performed by *MAP*. When the placer is finished, it updates the NCD file with the fully placed design. Given a placed NCD file and the PCF file, the Xilinx router produces a fully routed NCD file. The Xilinx router is invoked by the *PAR* utility.

2.2.8 Bitstream Generation

Finally the fully routed design must be converted into a bitstream that can be downloaded to an FPGA. Xilinx's *Bitgen* utility takes the fully routed NCD file and extracts the necessary configuration data for the entire device. *Bitgen* also performs the final DRC checks, ensuring that the design cannot damage the FPGA or the target platform. The whole process usually takes several minutes or more depending on the device size.

2.3 Related Work: Partial Reconfiguration Tool Kits

Aside from the well-known slot-based PR toolkit from Xilinx, there have been other attempts at PR tool kits from both Xilinx and the research community.

2.3.1 *JBits* and its Progeny

One of the more notable PR toolkit efforts was Xilinx's *JBits* effort, that offered a completely novel paradigm for FPGA development[12, 20, 21]. *JBits* provided a Java Application Programming Interface (API) for accessing the Xilinx configuration bitstream. Whether the bitstream resided on the host machine, or on an operational FPGA, *JBits* allowed the designer to make design adjustments directly in the configuration bitstream. While *JBits* was incredibly powerful, it had several limitations that prevented its widespread use. Specifically it required explicit instantiation of all logic and routing, an in-depth knowledge of the low-level architecture, and its low-level interface precluded the use of standard tool facilities such as timing analysis and logic optimization[20]. What ultimately killed off *JBits* though was its reliance on proprietary Xilinx data. When Xilinx decided to stop supporting the project, further development was rendered impossible.

JBits spawned several significant research projects which aimed to leverage its power while ameliorating some of its limitations. One limitation of *JBits* was its incomplete wiring coverage; *JBits* made a trade-off between wiring coverage and simplicity[22]. In [22], the author presents a *JBits* compatible Alternate Wire Database (ADB) which provides 100% wiring

coverage for supported devices. The ADB project provided designers with a comprehensive wire database without a large performance sacrifice. The ADB API and data structures are abstract enough to support many different Xilinx architectures. In fact OpenPR uses a database based on ADB as its tile and wire database. The JHDLBits project that built upon both *JBits* and ADB was presented in [23]. JHDLBits aimed to provide an open-source FPGA development environment that combined *JBits*' run-time bitstream control with the high-level design abstractions provided by BYU's JHDL project[23].

2.3.2 ReCoBus

Slot-based reconfigurable designs are susceptible to internal fragmentation since smaller modules can leave slot resources unused. One solution proposed in [24] is to represent reconfigurable regions of the FPGA as a fine-grained 2D tile grid where small fragments of modules can be placed. For this solution to be viable, a high-throughput, flexible communication system is needed. In [24] the authors present a structured communication architecture that allows module fragments to be placed anywhere within the reconfigurable grid. This concept was expanded upon in [25] which explained the difficulties of migrating the tool to the Virtex 5 architecture. A ReCoBus executable is available from the authors' website free for non-commercial use, but source code is not available.

2.3.3 Wires On Demand

Wires on Demand was another project that aimed to tackle the internal fragmentation problem and inflexibility of the slot-based PR flow. The solution proposed in [11] was to dynamically allocate resources from a pool. A dynamic module library would store partial bitstreams containing configuration information for module logic along with wrapper structures providing structurally consistent routing anchor points for module ports. The reconfigurable region is a 2D grid referred to as the sandbox. Modules can be dynamically placed within the grid at run-time and a simple channel router makes the necessary connections between modules. This approach allows the designer to build a full system in a span of several minutes using a vast library of precompiled modules. The Wires on Demand tool-flow was never intended for public release.

2.4 Related Work: Open-Source FPGA Design Tools

2.4.1 Architecture-Unaware Tools

There is a long history of open-source FPGA design tools that are architecture unaware. These are generally academic efforts targeting nonexistent, abstract FPGA architectures in order to research algorithms or design methodologies. Open-source tools are available that cover nearly every step of the development flow described in section 2.2 including simulation[26], synthesis[27], mapping[28], as well as packing, placement, and routing [29].

While these efforts represent significant contributions to the research community, their inability to build designs for commercial FPGA architectures leaves a huge gap in the open-source FPGA development research community.

2.4.2 Tools Supporting Xilinx FPGAs

Several new open-source tool sets are being introduced that will help bridge the gap between tools that support theoretical FPGA architectures and ones that could target actual devices. Rather than reproducing the excellent research done in [27, 28, 29], the tools discussed below provide hooks into the Xilinx development flow that would allow FPGA CAD tools to support Xilinx FPGAs.

Torc

Torc is an open-source infrastructure and tool set for Xilinx FPGA design that was jointly developed by University of Southern California's Information Sciences Institute East (ISI East), Virginia Tech's Configurable Computing Machines Laboratory (CCM Lab), and Brigham Young University's Configurable Computing Laboratory (BYU CCL). The Torc tool-set has the capability to "(1) read, write, and manipulate generic netlists, (2) read, write and manipulate physical netlists, (3) provide exhaustive wiring and logic information for commercial devices, and (4) read, write and manipulate bitstream packets[30]." Torc builds greatly upon previous work, tracing its roots all the way back to ADB as presented in [22]. More

recently, Torc derives from a set of unreleased tools developed and used internally by the team responsible for Torc. This unreleased predecessor formed the base upon which OpenPR is built. Specifically OpenPR relies upon the logic and wiring database and the bitstream manipulation capabilities provided by this Torc predecessor. As a member of the team, the author of this thesis was responsible for much of the bitstream functionality in the Torc predecessor. When OpenPR is released to the Public, it will be based upon the official version of Torc released by ISI East.

RapidSmith

RapidSmith is a framework and set of Java utilities for manipulation of Xilinx XDL files developed by the BYU CCL. RapidSmith provides a software API allowing users to read, write, and manipulate XDL files[31]. Since XDL files can be converted to and from Xilinx's proprietary NCD physical netlist format, RapidSmith allows indirect access to the physical netlist of a design. As demonstrated in [31], RapidSmith can be used to analyze and create designs and design tools targeted towards Xilinx FPGAs.

Chapter 3

PR Flow Components

The OpenPR toolkit, like the Xilinx PR tool kits, is intended to augment the utilities provided within a contemporary ISE release. A slot-based PR toolkit consists of utilities that aid in the generation of the static bitstream and multiple partial module bitstreams. Specifically the toolkit must provide the functionality outlined below:

1. Identification of a dynamically reconfigurable geometric region.

2. Exclusion of all logic and routing resources in the dynamic region from being used by the Static design.

3. The identification and location of routing terminals (via bus macros) that are spatially consistent in both the static design and all partial designs.

4. Confinement of all routing and logic associated with the partial module to the specified

geometric region.

5. Generation of Partial Bitstreams for each partial module that allows reprogramming of the dynamic region.

The remainder of this section describes the aforementioned components, how they are implemented in OpenPR, and the challenges encountered in implementing them.

3.1 Identification of a Dynamically Reconfigurable Geometric Region

The first step in creating a slot-based partial reconfigurable design is to floor-plan and define the dynamic region. In the Xilinx PR Toolkit dynamic regions are defined by AREA_GROUP constraints. These constraints define the dynamic region as a rectangular range of resources that reside within the region. The user can either manually add these constraints to the User Constraints File (UCF) or use Xilinx's PlanAhead tool to floor-plan the region and generate the necessary constraints. Once an AREA_GROUP is defined the MODE can be set as RECONFIG, telling the Xilinx PR tools to treat this geometric region as a reconfigurable area. The OpenPR tools presented here retain the AREA_GROUP constraint in defining the geometric reconfigurable region. This allows the designer to floor-plan the static and partial designs in PlanAhead, making it much easier to allocate the dynamic region while avoiding conflicts and resource scarcity.

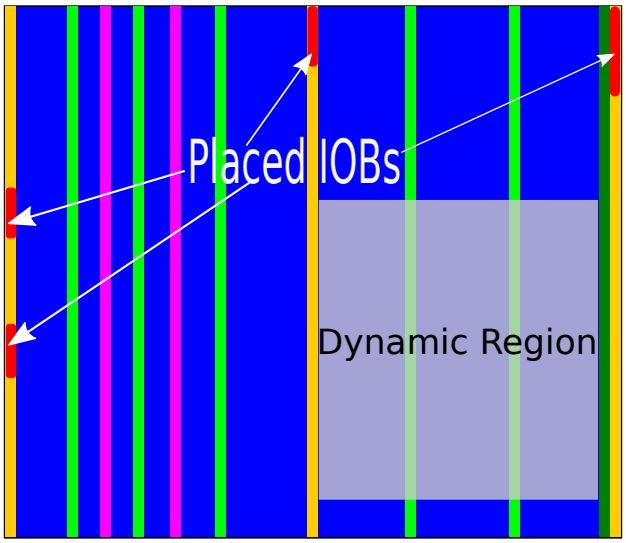


Figure 3.1: Floor-Planned Static Design

3.2 Exclusion of Dynamic Region Resources From Static Design

In run-time reconfigurable designs it is critical that reconfiguration of the dynamic region does not disturb operation of the static design. Since Xilinx Virtex devices offer glitch-less reconfiguration [32], the main issue is that resource conflicts between static and partial designs must be prevented. There are two possible approaches to avoiding resource conflicts. One is to keep track of which resources in the static design are contained within the dynamic region and prevent them from being used in any of the partial modules. The other is to prevent the placer and router from using any of the dynamic region's resources in the static design. The Xilinx PR toolkit takes a mixed approach to this problem. When an AREA_GROUP is marked as a reconfigurable region, the placer enforces exclusion of any

static logic from that region. A different approach is taken for routing exclusion where global routes are allowed to pass through the region and the arcs contained within the dynamic region are saved to a text file. This file is then used as an input to the partial bitstream generation process, ensuring that no routing conflicts are created. For the OpenPR toolkit it was decided that the static design should be prohibited from using any resources within the dynamic regions. This was mostly due to the limited granularity of bitstream configuration data available in Xilinx's published documentation.

3.2.1 Logic Exclusion

Without the benefit of a PR-aware placer, it was necessary to find some way to constrain the standard ISE placer. While there are documented constraints in the standard ISE tools that disallow placement of unrelated logic within an AREA_GROUP, experiments with several versions of the tools confirmed that these constraints are often ignored. In a run-time reconfigurable design, exclusion of static logic from the dynamic region is a condition that must never be violated. The ISE tools offer a CONFIG PROHIBIT constraint that disallows usage of specific sites, or a rectangular range of sites. If prohibiting a site creates an intractable placement problem, the ISE tools emit an error, halting the build process. Applying this constraint to a rectangular region is straightforward, and can be done from within PlanAhead. Employing this constraint within a PR flow is more difficult since exceptions must be made for the routing terminals, which have logic resources residing within the dynamic region. The initial approach was to enumerate all non busmacro sites

within the region and prohibit them individually. It was quickly discovered that the Xilinx tools are limited in their ability to parse very large UCF files. A more scalable approach was then taken that analyzed each tile column within the dynamic region and generated a prohibit constraint for the largest contiguous tile spans in the column. An example of how the *siteBlocker* utility splits the dynamic region into prohibit ranges is illustrated in Figure 3.2. The generated constraints are then appended directly to the static design's UCF.

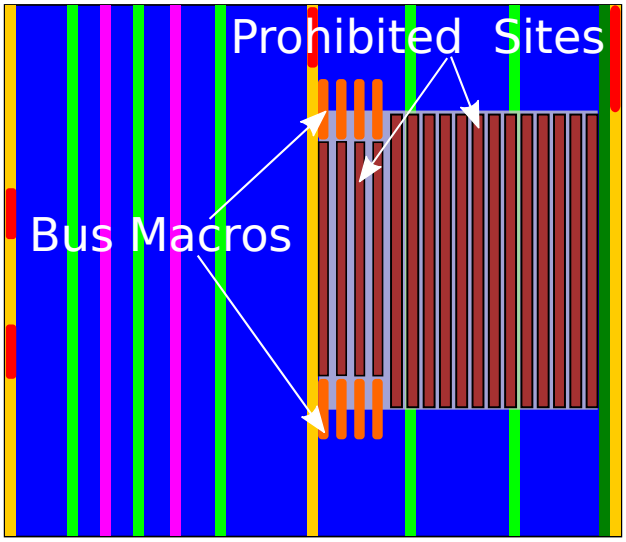


Figure 3.2: Dynamic region split into smaller prohibit ranges

3.2.2 Routing Exclusion

The Xilinx PR flow router allows nets in the static design to cross dynamic region boundaries. If these nets use wiring resources within the region, the router for the PR modules is told not to use them. OpenPR takes a different strategy, where the static design router is prevented from using any resource inside the dynamic region. To do this, OpenPR enumerates all

wires that pass from the static region into the dynamic region and prohibits them from being used in the static design. Figure 3.3 shows examples of wires that would need to be excluded from the static design. The *routeBlocker* utility enumerates all wiring resources that cross the dynamic region boundary and identifies which Programmable Interconnect Points (PIPs) are used to enable them. Unfortunately there is no constraint available in the standard tools to disallow the use of specific routing resources. Instead, the OpenPR tools implement a brute-force method of ensuring that routing does not enter a specified region. A single temporary net is introduced into the design that enables all the PIPs identified in the previous step. The ISE router is then invoked in re-entrant routing mode, routing the static design without modifying the blocking net. This concept was referred to as an *anti-core* in the *JBits* days [12]. Implementing this approach while using Xilinx's *PAR* utility as the router proved impossible. Many different approaches were trialed without success including UCF and PCF constraints, saving blocking routes as a hard macro, and generating directed routing constraints from *FPGA Editor*. *PAR*'s first goal is to meet timing, and as such it ignores many user constraints when attempting to achieve timing closure. The hard macro and directed routing approaches did not work because *FPGA Editor* is able to detect that the blocking route does not actually connect any elements of the circuit and removes the blocking PIPs from the macro or DIR constraint. Ultimately the only feasible solution was to use *FPGA Editor's* greedy router that does obey the PCF LOCK constraints that prevent the router from altering a net's routing. This same constraint was tried with the *PAR* router, but it either ignored the constraint or hung infinitely trying to force itself through

the dynamic region. This is unfortunate because the greedy router generally offers worse performance in both run-time and timing closure than *PAR*'s timing-driven router.

While there has not been any published research on how to manipulate the Xilinx router in this manner, there has been some research investigating fault-tolerant routers that avoid using specific routing resources. One approach to this as discussed in [33] is to route the design with a common routing algorithm such as the pathfinder algorithm, and re-route any nets using defective routing resources with an incremental routing algorithm. This approach is being considered for future versions of OpenPR that could leverage open-source routers from Torc[30] or VPR[34].

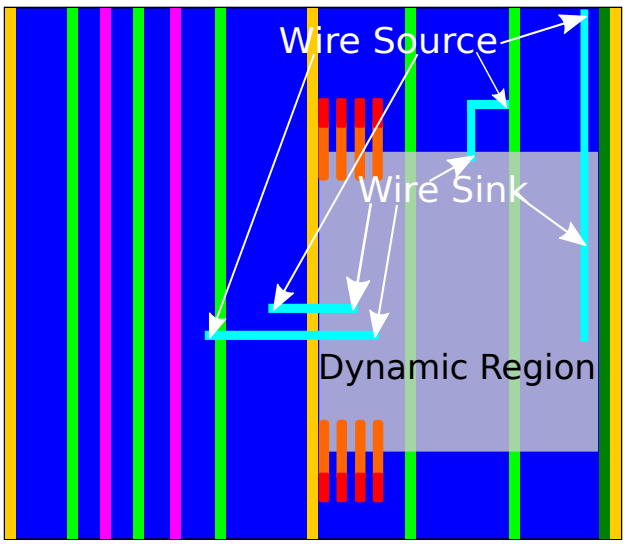


Figure 3.3: Examples of wires that need to be excluded from static design

3.3 Identification and Location of Routing Terminals

In a slot-based reconfigurable design the dynamic region's input and output terminals need to be spatially consistent with that of the static design. In Xilinx's PR tool kits bus macros are employed to this end. Bus macros are are hard macros placed in fixed locations at the regions inputs and outputs, usually on the perimeter of the dynamic region. In the ISE 9.2 version of the toolkit, manual placement of the bus macros was necessary. This was a tedious process which was maligned by researchers [11]. With the 12.1 release of the PR Toolkit, Xilinx introduced automatic placement of bus macros in a manner transparent to the user. The OpenPR tools retain the bus macro concept, and can automatically place these macros without user intervention. Currently, data-flow through the region is limited to a vertical top to bottom flow; however, this can be easily changed in the future with new bus macros and bus macro placement strategies. The *placeMacros* utility reads in the region's AREA_GROUP constraint from the UCF file and attempts to place input macros starting from the top-left and output macros starting from the bottom-left. Placement constraints for the bus macros are generated and appended to the UCF file. An example of a design with automatically placed bus macros is shown in Figure 3.4.

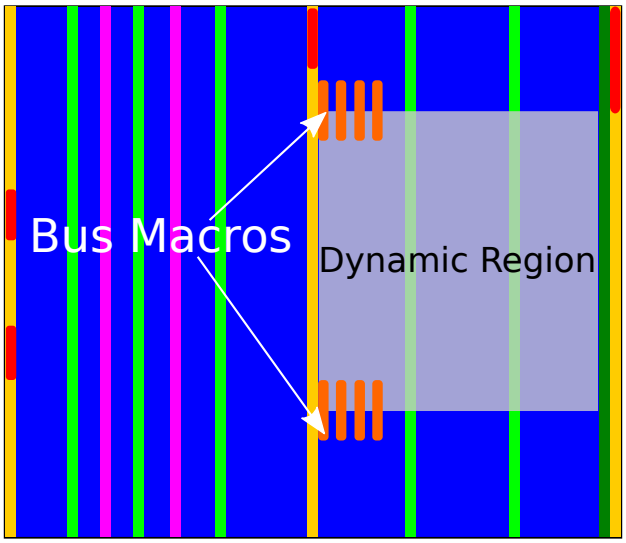


Figure 3.4: Static design with automatically placed bus macros

3.4 Confinement of All Partial Module Logic and Routing to Dynamic Region

All resources needed in the partial module must be contained within the dynamic region, otherwise the partial module will not function when placed in the slot. Confining the placement of logic in the partial module is simply done using the Xilinx AREA_GROUP constraints. The more difficult problem is constraining a core's routing to reside within the specified geometric region. At first this problem was approached as the inverse of the previously mentioned routing exclusion problem; the *routeBlocker* utility should prevent the router from using any wires that leave the dynamic region. Thus the same approach was taken; all wires which cross the region boundary were reserved before routing the design. After trying some test designs it became apparent that this approach was too restrictive, the ISE router

was unable to route even simple partial modules effectively. Since the route blocker for the static design ensures that the static design does not use any wire that crosses the region boundary, the partial design should be able to use these wires without conflict. The only remaining routing limitation then is that the partial module can only use routing resources whose PIP configuration bits reside in tiles within the dynamic region. Thus the behavior of the *routeBlocker* utility was modified for partial designs. All wiring resources within the dynamic region were traced back to the PIPs that could enable them, and those PIPs that were outside the region were reserved. This approach reserves far less routing resources, giving the router far more flexibility in routing the partial module.

3.5 Generation of Partial Bitstreams

To perform a run-time reconfiguration, the designer needs a partial bitstream that only reprograms the dynamic region. The OpenPR toolkit uses Xilinx's *Bitgen* program to generate a full bitstream for the partial design, and extracts only the configuration frames corresponding to the dynamic region, creating a partial bitstream. The granularity of this bitstream generation tool is limited by the Xilinx documentation on the bitstream format. In [35] Xilinx documents the bitstream header, configuration registers, and frame addressing scheme. This is enough information to be able to parse a bitstream at the frame level. In Xilinx Virtex 4, 5, and 6 devices one frame spans the height of a single clock region. Consequently, OpenPR is limited to reconfigurable regions that span multiples of clock regions, which means a gran-

ularity of 16 frames for Virtex 4 and 20 frames for Virtex 5. In addition to the configuration data for logic and routing within the dynamic region, the generated partial bitstream must also include clocktree information for the partial design. The *partialGenerator* utility combines the dynamic region configuration frames with the GCLK configuration frames to create a partial bitstream that can dynamically reconfigure the reconfigurable module slot.

Chapter 4

The OpenPR Process

Much of the criticism of previous Xilinx PR Flows has been focused on the difficulty of use and the need for much manual intervention[11]. OpenPR was designed with ease of use as a primary goal. Much effort was put into the automation of the design process. The designer mainly concerns themselves with the instantiation of busmacros into the modular HDL design, floor-planning the dynamic region, and setting up the project configuration file. Other details of the flow are automated and transparent to the designer. This chapter first discusses some of the tools used to automate the tool-flow and then details the process of creating a PR design using OpenPR.

4.1 Design Automation Tools

This section describes the various tools and technologies used to automate the OpenPR design process.

4.1.1 OpenPR XML Project File

A primary goal of the OpenPR tools is to minimize the amount of manual intervention required to build a run-time reconfigurable system. In pursuit of this goal, a file format was created that stores both configuration details and design implementation state. This OpenPR project file (.PROJ) stores project configuration and implementation progress, preventing redundant operations from being performed during the design process. Extensible Markup Language (XML) was chosen as the base format for the OPR project file because it is easily parsable (by both humans and computers), portable, and of course extensible[36]. Currently the user configures the initial project file with the necessary design parameters; however, this could be easily automated in the future. At each step in the flow, the file is incrementally updated with information. A sample file is shown in Figure 4.1. This sample shows a typical project file that includes XML tags for file locations, design parameters, and design status.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE boost_serialization>
<boost_serialization signature="serialization::archive" version="5">
<myProj class_id="0" tracking_level="0" version="0">
        <designName>top</designName>
        <projectPath></projectPath>
        <staticPath>/home/user/genesys/counter_sandbox_genesys</staticPath>
        <partialPath>/home/user/genesys/counter_partial_genesys</partialPath>
        <isPartial>0</isPartial>
        <ucfPath>/home/user/genesys/counter_sandbox_genesys/src/system.ucf</ucfPath>
        <dynamicAGName>counter</dynamicAGName>
        <busMacroPrefix>busmacro_xc5v_async_vert</busMacroPrefix>
        <busMacroNames class_id="1" tracking_level="0" version="0">
                <count>0</count>
                <item_version>0</item_version>
        </busMacroNames>
        <passThroughNetName>data_temp</passThroughNetName>
        <passThroughNet2></passThroughNet2>
        <clkNetNames class_id="1" tracking_level="0" version="0">
                <count>1</count>
                <item_version>0</item_version>
                <item>clk_BUFGP</item>
        </clkNetNames>
        <busWidth>8</busWidth>
        <deviceName>xc5vlx50t-ff1136-2</deviceName>
        <regionDefined>0</regionDefined>
        <yMin>-1</yMin>
        <yMax>-1</yMax>
        <xMin>-1</xMin>
        <xMax>-1</xMax>
</myProj>
</boost serialization>
```

Figure 4.1: Example project file

4.1.2 Bazaar Distributed Revision Control System

Bazaar is a distributed revision control system maintained by Canonical Ltd. Because it is a distributed version control system (VCS), Bazaar repositories can reside on a centralized server or on a user's machine. Code can be easily branched from and merged back into a Bazaar repository making it a good option for a project being worked on by one designer or a small team[37]. In OpenPR, Bazaar is used more as a productivity tool than a VCS. Using it in this manner is possible due to the distributed nature of the Bazaar VCS. A centralized VCS like Subversion (SVN) would not work well here. Using SVN would require the user to setup a server on their machine or on a remote machine. If a user wanted to save their progress, they would have to merge their branch back onto the server where it could possibly conflict with another user's branch. Similarly, in a multiple-user environment, significant coordination between users would be required to ensure that the repository remains organized; even when working on unrelated projects. Also, merging branches using SVN can be a complicated task,

especially when using older versions of SVN[38]. Using a distributed VCS allows OpenPR users to organize working directories at their discretion, save progress quickly and easily, and merge experimental branches painlessly. While Bazaar was chosen for its Windows support, ease of use, and Launchpad integration[39], other distributed VCS systems such as Git[40] or Mercurial[41] would likely work just as well.

The OpenPR utilities expect the design directories to be organized in a specific way. This was a necessary trade-off to ease automation of the design process. Instead of placing the burden of creating the proper file hierarchy on the designer, OpenPR leverages Bazaar to do this automatically. A read-only reference design is provided with the flow. This reference design contains the proper folder hierarchy as well as a stock Makefile and OPR file. The designer then branches and customizes the reference repository to create a standalone Bazaar repository for their design. This branch can now be used as both a working directory and a reference. With all configuration files, HDL sources, precompiled cores, and constraint files being included in the Bazaar repository, creating a new design for the same platform is as simple as branching the working repository.

4.1.3 GNU Make Utility

GNU's *make* utility allows the controlled generation of executables and other files from a collection of source files. Rules for generating the target files are coded into a "Makefile." For each target, the Makefile specifies source file dependencies and a list of commands necessary for generating the target. The build process can be incremental where sub-targets are

compiled from source files, and then the final target is compiled from the sub-targets. This is a common flow is software development where object files are built from source files, and are subsequently linked together into an executable or library.[42].

OpenPR uses Makefiles and the *make* utility to incrementally build static and partial bitstreams for each design. This is a natural fit as the Xilinx implementation flow is a largely sequential process where each step depends upon files produced in the previous step. The OpenPR Makefiles contain rules that outline the dependencies and commands necessary for executing each step of the build process described in Section 2.2.

4.1.4 Xilinx PlanAhead

Xilinx's *PlanAhead* utility is a powerful tool used for floor-planning, building, analyzing designs. Recent releases of the Xilinx PR toolkit have used PlanAhead as the GUI interface to the PR flow[43].

Using the OpenPR flow does not require PlanAhead, but it is the recommended tool for floor-planning the dynamic region and verifying that the region contains enough resources for the partial modules.

4.2 Steps Common to Both Static and Partial Module Designs

This section offers a detailed description of PR design implementation process in OpenPR. An overview of the OpenPR flow is shown in Figure 4.2.

4.2.1 Checking Out Reference Repositories

The first step in creating an OpenPR design is to branch Bazaar repositories for both static and partial designs. This is done by issuing the commands shown below:

```
$ bzr branch <openpr/reference_repos>/reference_sandbox ~/my_opr_sandbox
$ bzr branch <openpr/reference_repos>/reference_partial ~/my_opr_prm_0
$ bzr branch <openpr/reference_repos>/reference_partial ~/my_opr_prm_1
```

Branching from an existing reference design has several advantages. It provides the designer with a pre-built directory structure compatible with OpenPR, necessary busmacro files, and blank templates for OPR, HDL and UCF files. Due to the automated nature of the OpenPR flow, source and intermediate files need to be in predictable locations. A newly branched sandbox repository will have the directory structure and template files shown below:

The repository for the partial modules is has a similar directory structure. The primary difference is in the Makefile and the default OPR project file.

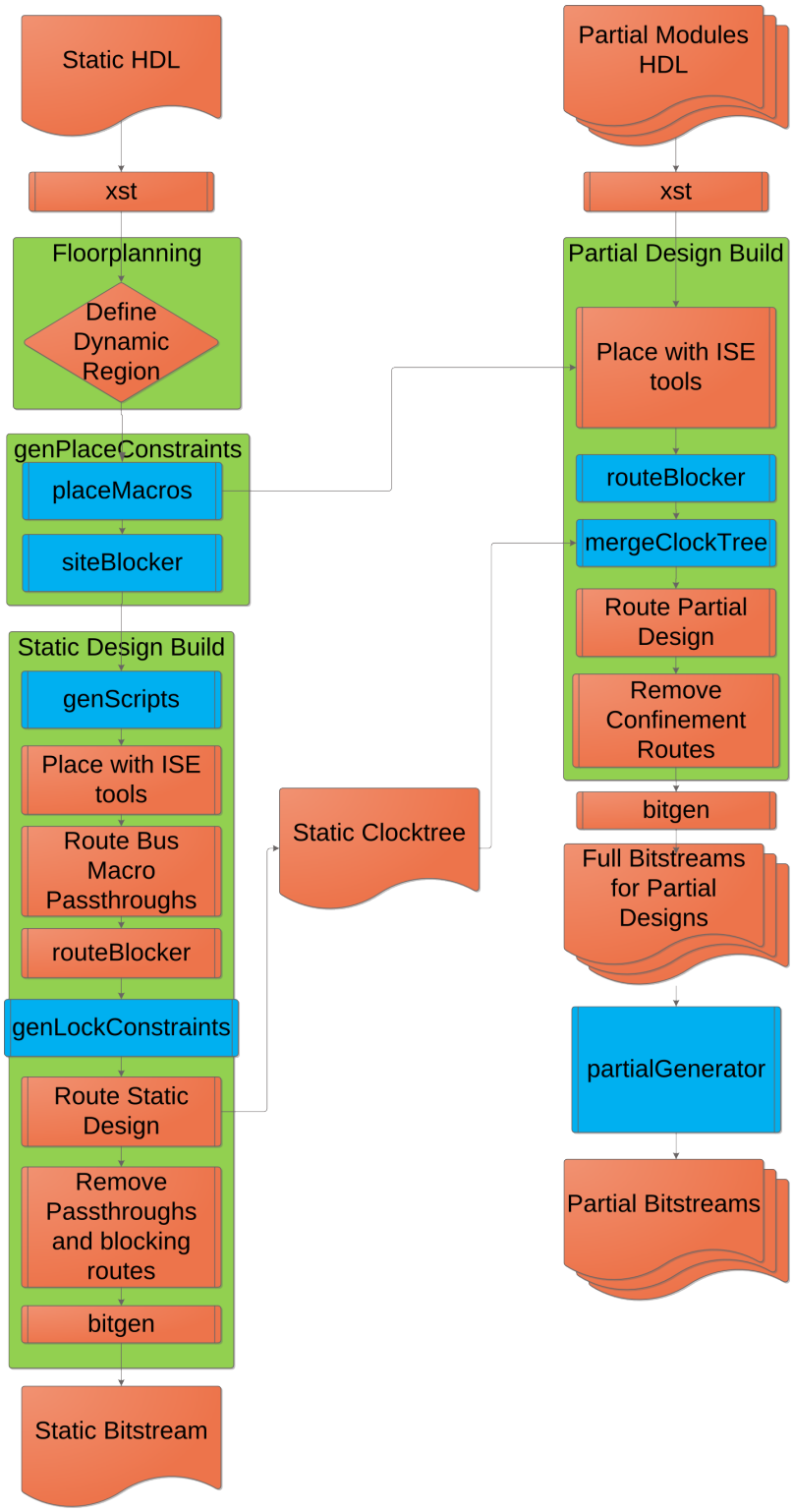


Figure 4.2: Overview of OpenPR design process

```
my_opr_sandbox/
├── Makefile ..... Static Design Makefile
├── top.proj ..... OpenPR Sandbox Project File
└── src/
    ├── sandbox.vhd ..... Static Design Top Level
    ├── system.ucf
    └── precompiled/
        └── busmacro_* ..... Pre-built Bus Macros
```

Figure 4.3: Example Directory Structure for Static Design

```
my_opr_prm_0/
├── Makefile ..... Partial Module Makefile
├── top.proj ..... OpenPR Partial Module Project File
└── src/
    ├── sandbox.vhd ..... PR Module Top-level With Bus Macros
    ├── prm.vhd ..... PR Module Implementation
    ├── system.ucf
    └── precompiled/
        └── busmacro_* ..... Pre-built Bus Macros
```

Figure 4.4: Example Directory Structure for Partial Design

4.2.2 HDL Entry and Synthesis

With the reference repositories checked out, the next step is to create the HDL code. The top-level entity and all supporting VHDL/Verilog files should be kept in the src directory. The default top-level entity is "top"; if this is changed, the Makefile's *TOP* variable needs to be set accordingly.

One of the major differences between a regular static design and an OpenPR design is the

need to declare and instantiate bus macros in the top-level entity. Unlike the Xilinx 9.2PR Toolkit, each bus macro must be defined as a unique component following a set naming scheme. Each bus macro component has a name prefix composed of the busmacro target architecture, type, and direction. The component name is made unique by appending a bus macro index value to the end. Indexing starts from 0 and increments by 1 for each bus macro, following the pattern used by the *placeMacros* utility to automatically place the macros. The name of the instantiated macro name is simply the component name with the macro index value appended a second time. Table 4.1 gives naming examples for different data widths assuming a macro width of 8 bits.

Bus macros will need to be instantiated in the top-level of both the static design and the PR

Table 4.1: Example illustrating bus macro naming conventions

Data Width	**Component Names**	**Instantiation Names**
8	busmacro_xc5v_async_vert_input_0	busmacro_xc5v_async_vert_input_0_0
16	busmacro_xc5v_async_vert_input_0	busmacro_xc5v_async_vert_input_0_0
	busmacro_xc5v_async_vert_input_1	busmacro_xc5v_async_vert_input_1_1
20	busmacro_xc5v_async_vert_input_0	busmacro_xc5v_async_vert_input_0_0
	busmacro_xc5v_async_vert_input_1	busmacro_xc5v_async_vert_input_1_1
	busmacro_xc5v_async_vert_input_2	busmacro_xc5v_async_vert_input_2_2

module designs. There are three sets of std_logic_vector signals for each pair of input/output bus macros. One set drives all inputs to the dynamic region, the second set passes signals

from input bus macros to output bus macros, and the final set of signals are the outputs from the dynamic region.

Before floorplanning the design it is a good idea to synthesize the sandbox and the partial modules. Resource usage and timing information from the synthesis report provides valuable information that will assist in floorplanning the design. To build the design, the following commands should be issued:

```
$ cd my_opr_sandbox && make xst
$ cd ..
$ cd my_opr_prm_0 && make xst
$ cd ..
$ cd my_opr_prm_1 && make xst
```

4.2.3 Floor-Planning

The first step in floor-planning is to assign static resources such as input/output pins, clock buffers, and hard IP blocks as required by the target platform. With these constraints in place, the next step is to define a dynamic region in an empty area of the chip. Using PlanAhead, the designer can draw a pblock defining the dynamic region; doing so generates an AREA_GROUP constraint in the UCF file which contains user constraints for the design. PlanAhead also allows the designer to check that the region has sufficient resources for a given PR module. Once this is done, the designer can export the UCF from PlanAhead and

continue with the build process. The floor-planned design should appear similar to Figure 4.5a.

4.2.4 OpenPR Project File Configuration

Before a PR design can be built, the OPR project file needs to be configured for the target platform and design. Table 4.2 below shows the fields that need to be configured when starting an OpenPR build.

The default values for some of these fields when starting an OpenPR build is shown in Table 4.3.

4.3 Static Design Flow

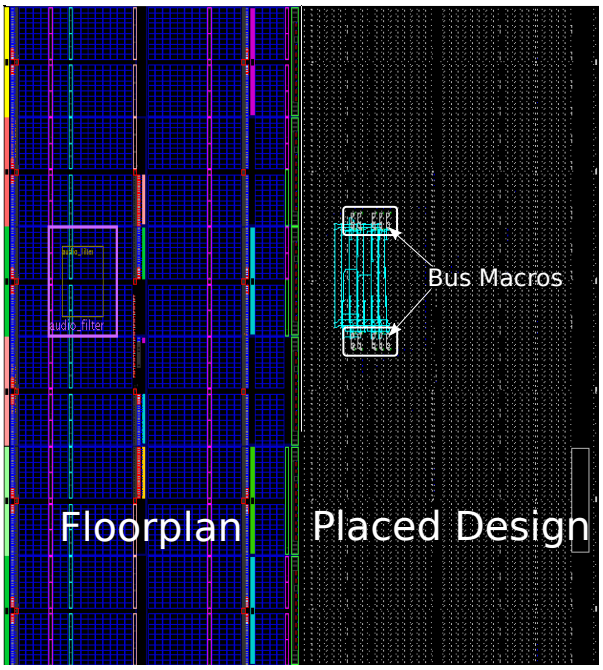
Once the OpenPR project file is configured, all source files have been compiled into Xilinx's synthesized netlist format (NGC), and the floor-planned design is constrained by a UCF, the designer can create the bit file for the static design with the command:

```
$ make static
```

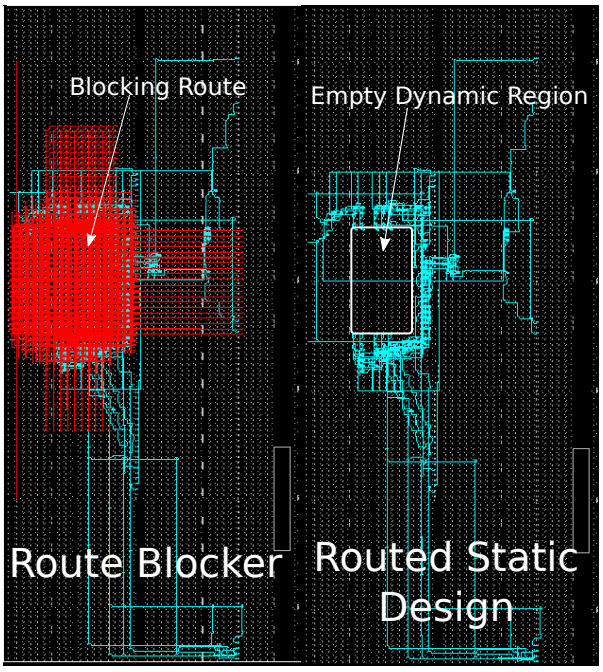
The Makefile fully automates the construction process, invoking a series of OpenPR utilities and Xilinx utilities. The remainder of this section examines the lower-level OpenPR steps and utilities that accomplish this.

Table 4.2: OpenPR Project File Configuration

Field	**Explanation**
designName	This field should match the top-level design entity
staticPath	Full path static design directory
partialPath	Full path partial design directory
isPartial	Boolean value indicating whether project is a sandbox or a partial module
ucfPath	Full path to UCF file with pin/timing constraints for static design
dynamicAGName	Name of dynamic region AREA_GROUP as defined in UCF file
busMacroPrefix	Naming prefix for busmacro defining target architecture and busmacro type
passThroughNetName	For sandbox, identify nets passing through dynamic region connecting input macros to output macros in the static design; for partial design, identify nets connecting virtual IO pins to dynamic region inputs/outputs
clkNetNames	Collection of clock nets in the design.
busWidth	Width of dynamic region datapath
deviceName	Full device name including package and speed grade
regionDefined	Boolean value indicating whether OpenPR region is defined yet
x/y Min/Max	Coordinates defining vertices of dynamic region



(a) PlanAhead Floor-plan (b) Placed Static Design



(c) Blocking Routes (d) Final Static Design

Figure 4.5: Screenshots showing progression of OpenPR static design flow.

Table 4.3: OpenPR Project File Default Values

Field	**Default Value**
isPartial	0 for sandbox, 1 for partial
ucfPath	$(staticPath)/src/system.ucf
busMacroPrefix	For Virtex 5, asynchronous, vertical data-flow: bus-macro_xc5v_async_vert
clkNetNames	clk_BUFGP
busWidth	8
deviceName	xc5vlx50t-ff1136-2
regionDefined	0
x/y Min/Max	-1 before building design

4.3.1 Generate Placement Constraints

The OpenPR *genPlaceConstraints* utility parses the UCF and the OpenPR project file and generates the necessary placement constraints for the static design. This entails both choosing locations for the bus macros and generating CONFIG PROHIBIT constraints to exclude dynamic region sites from the static design. Generated constraints are then appended to the UCF, and Xilinx's *NGDBuild* and *MAP* utilities are invoked to create a placed static design.

4.3.2 Automatic Script Generation

The router used by the OpenPR toolkit is the point-to-point router provided by Xilinx's *FPGA Editor*. *FPGA Editor* can be scripted from the command line using a proprietary scripting language. The OpenPR *genPTScripts* utility automatically generates FPGA Editor scripts to route and unroute the temporary pass-through nets in the design.

4.3.3 Route Pass-Throughs

Using an *FPGA Editor* script generated in the previous section, all pass-through nets in the design are pre-routed. These nets are pre-routed for two reasons. Firstly, because they pass through the dynamic region and would be unroutable with blocking routes in place. Secondly, these routes will be removed from the final design and need not be constrained in any way. At this point the static design will resemble Figure 4.5b.

4.3.4 Create Blocking Routes

Once the static design is fully placed with all pass-throughs routed, the *routeBlocker* utility is run, using one of the designated pass-through nets as a blocking net. The end result will be a dense structure that when instanced into the design will force the Xilinx router to avoid it during the final assembly. Following the procedure outlined in Section 3.2, *routeBlocker* enumerates all routing resources that cross the dynamic region border and adds them to the blocking net. Then the physical netlist of the placed static design is converted to a textual

XDL file representation, and the blocking net is merged into it. The modified XDL file is then converted back to the proprietary NCD format using the Xilinx *XDL* utility. At this point the static design looks similar to Figure 4.5c. Note that in this figure the blocking routes appear to enlarge the desired dynamic region, yet this is not the case. Wire segments that have one endpoint within the dynamic region and the other endpoint outside must also be blocked as explained in Section 3.2.

4.3.5 Generate Lock Constraints for Blocking Routes

The *genLockConstraints* utility automatically generates LOCK constraints for the blocking routes and busmacro nets, and appends them to the physical constraints file (PCF) generated by the Xilinx *MAP* utility. These constraints prevent *FPGA Editor* from unrouting or changing any of these nets.

4.3.6 Routing Design and Removing Pass-Throughs

With the blocking net locked down, *FPGA Editor* is invoked to route all remaining unrouted nets. Once the static design is routed, blocking nets and pass-through nets can be removed from the design. The resulting final static design will look similar to Figure 4.5d.

4.4 Partial Module Design Flow

Creation of the Partial bitstream follows a similar flow to that explained in the previous section. For the sake of brevity, this section will only discuss the aspects of the flow that differ from those explained in the previous section.

As with the static design, the Makefile fully automates the construction process, invoking a series of OpenPR and Xilinx utilities. The designer can build the partial bitstream by issuing the following commands:

```
$ cd my_opr_prm_0 && make partial
```

4.4.1 HDL Considerations

It is important to remember that the goal here is to trick the Xilinx tools into treating our partial design as a full, valid design. As such, the partial module requires a top-level wrapper with virtual I/O ports that connect directly to the bus macro inputs and outputs. These ports will be packed into IOBs and placed by the *MAP* utility. Depending on the floor-plan of the design, it might be necessary to constrain these virtual pins to make them easier to route to and from the bus macro pins.

4.4.2 Merge Static Clocktree into Partial Design

The *mergeClockTree* utility extracts clock tree routing information from the static design and inserts it into the placed partial design file. Since the partial design is routed using re-entrant routing, the static clocktree will be preserved. The resulting partial design clocktree will be a super-set of the static clocktree.

4.4.3 Generate Partial Bitstreams

Finally, *genPartialBitstream* is invoked to create a partial bitstream. The generated bitstream will reprogram the FPGA with configuration data for the partial module and its clock tree.

Chapter 5

The Code

From a code standpoint, the OpenPR tools are intended to be portable, extensible, and maintainable. This chapter explains how the code design and style contribute towards these goals.

5.1 Portability

The OpenPR utilities are targeted towards all operating systems supported by Xilinx ISE. Currently, this includes Linux and Windows for 32 and 64 bit x86 architectures. Code for the OpenPR utilities is written in ANSI C++, using non-portable features such as bitfields as sparingly as possible. Care was taken to ensure that all external dependencies not part of the Standard Template Library (STL) are available on all platforms.

The initial OpenPR builds can be built without modification on Linux machines with GNU

Table 5.1: Operating systems used to test OpenPR build

Operating System	**Architecture**	**GCC Version**
Ubuntu 9.04	i686	4.2
Ubuntu 10.10	i686	4.4.5
Ubuntu 10.10	x86_64	4.4.5
Windows 7 (Cygwin)	x86_64	4.3.4

Compiler Collection[44] C++ Compiler (G++) 4.2, and the *boost* library set. With minor modifications to included Makefiles, OpenPR builds without issue using other modern G++ versions. The OpenPR compilation process has been tested on several different platforms with different versions of G++ 4.x, including Windows within the Cygwin environment. Table 5.1 lists the platforms on which the OpenPR compile process has been tested. In theory, OpenPR code should be generally compatible with tool-chains other than the GNU tool-chain; however, it is likely that the build process and perhaps some code would need to be modified.

5.2 Extendability and Maintainability

This section explains how the OpenPR architecture was intended to facilitate extensibility, maintainability, and ease of use. When discussing extensibility and maintainability for

OpenPR, there were two main requirements. The first is the addition of new features, the second is support for new architectures and devices.

The approach taken here was to modularize extended feature sets and device support, separating extra features and device specific functions from common functionality needed for all OpenPR projects. To achieve this modularity, OpenPR leverages C++ features such as inheritance[45] and polymorphism[46]. All code that is architecture agnostic and implements base functionality should reside in abstract base classes. Code that implements extra functionality or architecture specific details should be in a separate class that inherits from the abstract class, implementing or overriding the necessary functions. Internally, OpenPR utilities only access pointers to the base classes, relying on polymorphism to determine when device-specific or extended functions need to be invoked. This strategy encourages code reuse by implementing common functionality in base classes, aids extensibility by allowing the user to override only a subset of functionality as needed, and makes the project more maintainable by limiting the amount of work required to add new architectures. The example below gives an example of how this approach applied to some of the bitstream manipulation functionality of OpenPR.

The abstract device class defines how bitstreams of Xilinx devices are split into bitstream frames according to device layout. This class allows the high level bitstream class to easily map between device tiles and bitstream frames. Attributes and functions related to bitstream functionality are very much architecture and device dependent, making the device class a natural fit for inheritance. The inheritance graph in Figure 5.1 below shows the

relationships between the abstract base class, and the child classes containing architecture and device details:

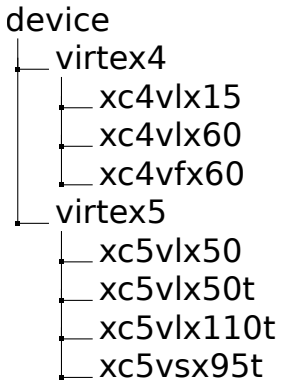


Figure 5.1: Inheritance Graph or Bitstream Description Classes

In the above graph, architecture-agnostic details are inherited from the device base class, and device-agnostic details are inherited from the architecture base class. At run-time the base device pointer is allocated an object corresponding to the device set in the OPR project file. Through the use of inheritance and polymorphism, the device pointer now stores all the attributes and functions specific to the target device. Inheritance structures such as the one shown above are used throughout OpenPR to ensure that adding new devices or features is as simple as creating a new inherited class that only implements the minimum amount of new functionality.

Chapter 6

Results

It's difficult to objectively measure the success of a tool-chain because there are so many different factors of varying importance to different people. The most important question of course is does the tool-chain work? To this end, this chapter first demonstrates OpenPR's functionality by describing test applications that have been successfully implemented using OpenPR. Then it presents a comparison of OpenPR to Xilinx PR tool kits including design build times and feature set comparisons.

6.1 Test Designs

To demonstrate OpenPR toolkit capabilities, various PR designs were implemented on popular Virtex 5 FPGA development boards. Boards used include the XUPV5-LX110T and Genesys V5-LX50T from Digilent and the V5-SX95T development kit from Avnet. Table

Table 6.1: Designs implemented using OpenPR

Design	**Development kit**	**Partial modules**
Counter	XUPV5-LX110T Genesys V5-LX50T Avnet V5-SX95T	Up-counter Down-counter Pass-through
Audio filters	Genesys V5-LX50T	Low-pass filter High-pass filter Pass-through
Video filters	XUPV5-LX110T	Grayscale filter Intensity filter MaxRGB filter Negative filter

6.1 lists the test applications, their corresponding PR modules, and the development boards they were implemented on.

6.1.1 Basic Counter Design

The first test application used to test OpenPR is a basic 8-bit counter. An 8-bit dip-switch is connected to the dynamic region inputs, and a Light Emitting Diode (LED) is connected to each of the dynamic region outputs. This application can switch between three PR modules. The first module is a simple pass-through, each bit of the dip-switch will

either enable or disable a single LED. The other two designs implement 8-bit counters that count in opposing directions. In both counter designs, the dip-switch inputs can enable or disable the corresponding bit of the 8-bit counter. This application was valuable in verifying OpenPR functionality. When all three modules worked, it proved that inputs were properly routed to the PR region, outputs were properly routed from the PR region, and that run-time reconfiguration of the FPGA is functioning correctly. This design was implemented successfully on all target platforms.

6.1.2 Audio Filtering Design

While the counter proved basic functionality of OpenPR, the tool-chain must be able to build more complex, realistic designs. The next attempted design implemented audio filters in the reconfigurable region. The target platform for this design was the Digilent Genesys V5 board which features a Virtex 5 LX50T FPGA and an AC97 audio codec chip. A VHDL AC97 audio codec controller was borrowed from [47] and incorporated into the base design. In this application, an audio signal is fed into the line-in input of the Genesys board, and filtered audio is fed out of the headphone jack. The AC97 controller's sampling frequency is fixed at 48kHz. During each sampling period, the AC97 controller generates 16-bit audio packets for each audio channel. These audio packets are filtered by whichever module is instantiated in the PR region. Once again, three PR modules were implemented. The first was a simple pass-through, the second was a low-pass filter, and the third was a high-pass filter. The low-pass and high-pass filters were implemented using *Coregen's* finite impulse response (FIR)

filter generator. The low-pass filter had a cut-off frequency of 500Hz and the high-pass a cut-off frequency of 2kHz. Operation of the filters was verified by generating signals with frequencies between 10Hz and 10kHz and observing the amplitude of the filtered output on an oscilloscope. This set of partial bitstreams worked well with some glitches. Occasionally the partial bitstreams had to be reprogrammed multiple times before the filter operated correctly. It is unclear what caused these issues, but we have observed similar issues when using *Coregen* filters with run-time-reconfigurable designs in the past.

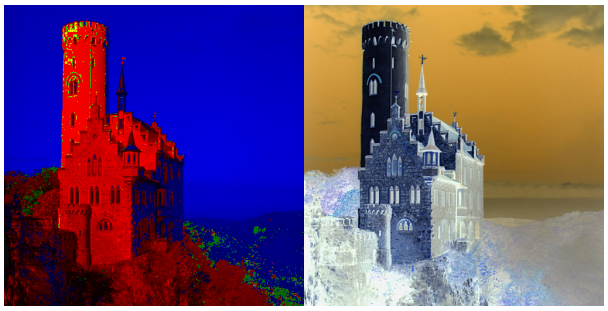
6.1.3 Video Filtering Design

A video filtering design was implemented on the XUPV5-LX110T development board using OpenPR. The application processes in real-time a video signal in RGB format, applies a filter to that signal, and outputs the results to a display. Four video filters were implemented as partial modules: grayscale, intensity, maxRGB, and negative. The static design contains logic that extracts the RGB signal, feeds it to the filter, and outputs the filtered video. Figure 6.1 shows a screenshot of the input signal, and Figure 6.2 illustrates what the signal looks like after being filtered by each PR module.

A series of FPGA Editor screenshots illustrating the concept of how the slot-based PR model applies to an actual design is shown in figures 6.3 and 6.4. The static design is shown in Figure 6.3, with the empty dynamic region slot highlighted. Screenshots showing the logic within each PR module that fits in the slot is shown in Figure 6.4.

Figure 6.1: Original image

(a) Grayscale filter (b) Intensity filter



(c) MaxRGB filter (d) Negative filter

Figure 6.2: Screenshots showing the outcome of the video filters

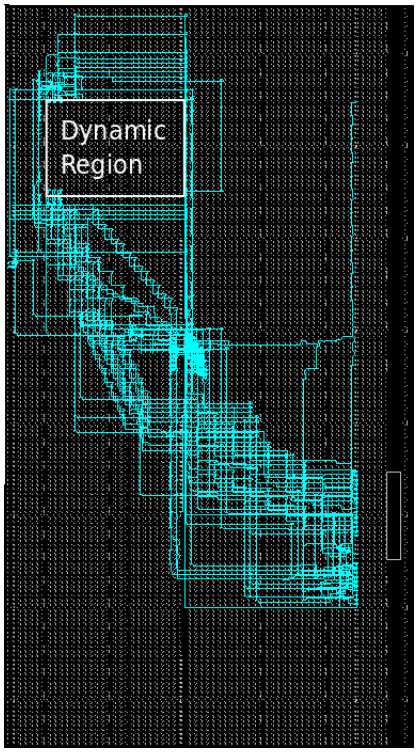
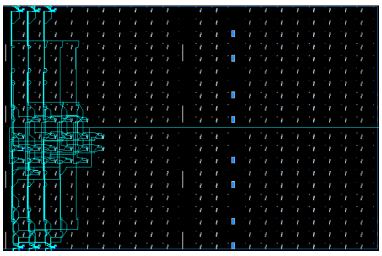


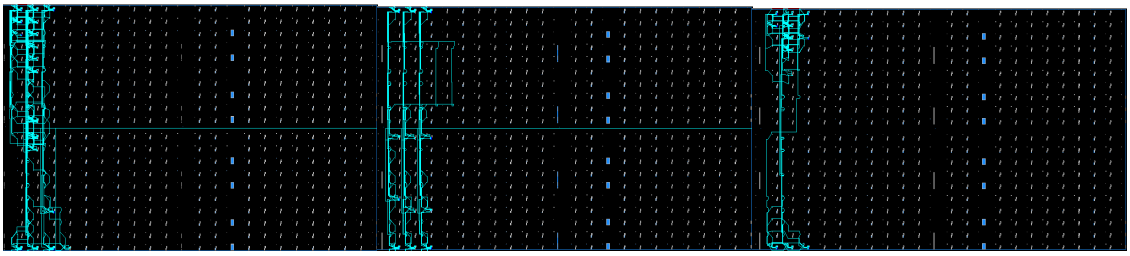
Figure 6.3: FPGA Editor Screenshot of Static Design for Video Filter Application

6.2 Performance

An important aspect of any CAD tool is the amount of time it takes to build a design. In order to evaluate the speed of the OpenPR build process, the video filtering application was built using both OpenPR and the Xilinx 9.2 PR Toolkit. Table 6.2 compares partial module build time in both tool kits. The OpenPR developers have focused on functional completeness and bitstream generation times presented in Table 6.2 represent mostly unoptimized code. Even so, OpenPR bitstream compilation times are competitive with the Xilinx 9.2 PR toolkit, with less than 10% increase in the time to generate a bitstream. With some effort put into optimization, it should be possible to make the OpenPR build process faster than that of



(a) Negative Filter



(b) Max-RGB Filter (c) Intensity Filter (d) Grayscale Filter

Figure 6.4: FPGA Editor Screenshots of PR Modules

the Xilinx PR flow. It is important to note however that compared to the Xilinx PR flow, OpenPR has almost no design rule checking (DRC) capability for generated bitstreams and uses a faster but less competent router.

6.3 Feature Comparison

OpenPR was developed with different design goals and use cases in mind than the Xilinx PR tool kits. While the initial OpenPR release only implements the traditional slot-based method of partial reconfiguration, the hope is that releasing an open-source PR toolkit will allow researchers to develop more powerful tools on their own. However, OpenPR still provides a feature-competitive tool-set to build slot-based PR applications. Table 6.3

Table 6.2: Bitstream generation times with Xilinx 9.2i PR and OpenPR (Video Filters)

Module name	**Xilinx 9.2i PR**	**OpenPR**
Video filters static	4m11s	4m33s
Grayscale filter	4m13s	4m15s
Intensity filter	3m53s	4m12s
MaxRGB filter	4m23s	4m11s
Negative filter	4m13s	4m11s

provides a feature by feature comparison of OpenPR to the Xilinx PR tool kits.

The remainder of this section discusses the advantages and disadvantages of the OpenPR tool-flow outside of the context of the slot-base reconfigurable design.

6.3.1 OpenPR Advantages

While OpenPR does provide a case for use as a slot-based partial reconfiguration tool kit, its main potential lies in its flexibility and extensibility. OpenPR is intended to be used as a base for research into new partial reconfiguration and FPGA productivity paradigms.

One of the biggest advantages for research in partial reconfiguration is OpenPR's ability to support FPGA architectures that Xilinx does not. For example, Spartan 6 support could be added to OpenPR without a great deal of effort. While this architecture does not have an Internal Configuration Access Port (ICAP) for self-configuration at run-time, OpenPR could

Table 6.3: Comparison of OpenPR to Xilinx PR tool kits

Feature	**OpenPR**	**Xilinx 9.2 PR**	**Xilinx 12.3 PR**
Automatic Bus Macro Placement	Yes	No	Yes
Custom Bus Macro Support	Yes	Yes	No
Cost	Free	Unavailable	Variable
Extensible	Yes	No	No
Device Support	Virtex Expandable	Virtex	Virtex
DRC	No	Yes	Yes
Router Performance	Poor	Good	Great

allow researchers to pursue PR research on low-cost platforms through external configuration interfaces.

OpenPR is unique in its ability to be used as a platform for new research in partial reconfiguration as well as FPGA productivity. OpenPR provides several capabilities that will allow researchers to break the boundaries of traditional work-flows for FPGA design:

1. The ability to prevent the Xilinx placer from using certain parts of the chip. To this author's knowledge OpenPR would be the only tool available that generates the necessary constraints in an automated matter.

2. OpenPR allows users to create recipes that automatically place pre-compiled hard

macros. This is currently used for automatic busmacro placement, but there is no technical reason why it could not be used to automatically place pre-compiled modules as well.

3. The ability to constrain the Xilinx router, forcing it to avoid certain regions of the chip. To this author's knowledge, OpenPR is the only freely available tool that would give researchers this capability.

4. Perhaps most significant is the bitstream manipulation and generation capabilities of OpenPR. OpenPR gives users to ability to read and write full or partial bitstreams, copy and paste configuration frames from different bitstreams, intelligently manage clock-tree frames, and of course generate partial bitstreams given a defined dynamic region.

Between the features unique to OpenPR, and the features available in the Torc tools, including many contributed by this author, researchers will now have an opportunity to create custom FPGA tools targeting partial reconfiguration, run-time reconfiguration, compile-time FPGA productivity, and run-time FPGA productivity.

The traditional Xilinx tools treat the FPGA as a "sea of gates" with a flattened hierarchy of low-level components. While this approach results in highly optimized design, it also engenders long compile times and inefficient design techniques. Treating the FPGA as a platform to implement pre-compiled components could result in greater design productivity. There are many successful examples of this approach in the Software domain, such as the

Linux kernel[48], Java[49], and Visual C#[50]. Xilinx has put more effort recently into a hierarchical design flow with the 12.3 PR Tool-kit[43] and the Partitions Flow[51]. However, neither of these flows gives designers the ability to build complete designs using modular components at compile-time or at run-time. The PR Tool-kit limits users to fixed-size slots that can be swapped one at a time. The Partitions Flow allows designers to designate specific design instances as "partitions", isolating their implementation details from the rest of the design. This flow even allows the designer to export and import instance implementations into the design. However, these operations work at the instance level, and keep track of global rather than relative logic and routing resource usage. This greatly limits the ability to build new designs based on previously compiled partitions. To get closer to the flexibility of software platforms, FPGA design tools should strive for the ability to arbitrarily place and route pre-compiled modules at run-time or compile-time. While something similar to this was proposed in [11], those tools were never intended for public release. OpenPR could form a base for research into new modular tool-flows. It is uncertain why Xilinx has not applied more resources to pursue such a flow. At this point the research community bears the responsibility of proving that this is a viable and efficient design methodology, and hopefully commercial support will follow.

One feature that makes it well-suited to all these areas is OpenPR's ability to create a sandbox region completely devoid of logic and routing. This capability of the Xilinx 9.2 PR Toolkit is what made projects like [52, 11] possible. However, as noted in [52] the `ROUTING=CLOSED` constraint that enabled this functionality did not always work when used

on Virtex 4 based designs. Additionally, this functionality does not work at all for Virtex 5 based designs in either Xilinx PR 9.2 or Xilinx PR 12 tool kits due to the single-slice bus macros employed; since these macros have inputs that reside within the region, the router must enter the region to connect to them.

When this capability is combined with the ability to automatically place hard macros, and copy and paste bitstream modules, OpenPR makes a strong case for inclusion as part of a compile-time FPGA productivity project.

6.3.2 OpenPR Disadvantages

The main disadvantage of OpenPR is its use of the *FPGA Editor* router. This router was intended to route small portions of a design after *FPGA Editor* was used to make manual changes . As such, this router is a point-to-point greedy router that is not timing-driven. If OpenPR were able to use the timing-driven router included in the *PAR* utility, OpenPR would have a much stronger case as an alternative to the Xilinx PR 12 tools. Currently, faster designs that easily achieve timing closure with Xilinx PR 12 might fail timing when built with OpenPR. OpenPR also does not perform DRC checks after the placement and packing stage; this could lead to dangerous situations for the target FPGAs. In its current incarnation, OpenPR is command-line based. To facilitate widespread adoption, it may be necessary to develop GUI interfaces to OpenPR.

Chapter 7

Conclusion

This thesis presented OpenPR, an open-source, slot-based, partial-reconfiguration toolkit targeted towards Xilinx FPGAs. As presented, the toolkit supports Virtex 4 and Virtex 5 FPGAs using the Xilinx ISE 11.x and 12.x series. The discussion started with a background in FPGA design tools, along with a summary of research projects related to partial-reconfiguration and open-source FPGA design tools. This was followed by a technical introduction to the components that make up OpenPR and a detailed walk-through of the OpenPR design process. The final sections discussed the portability, extensibility, functionality, and performance of the OpenPR tool-set.

OpenPR has been shown to be a successful implementation of the slot-based partial reconfiguration paradigm. While there is room for improvement in design performance, device support, and support for new PR design paradigms, current incarnation of OpenPR provides a solid base for further development.

7.1 Future Work

The version of OpenPR presented here offers support for a limited number of devices, uses a non-timing-driven router, and implements the slot-based PR paradigm which many consider to be lacking[11]. The real potential of OpenPR lies in its ability to be expanded, adding support for new devices and paradigms. One major criticism of OpenPR is the performance of the *FPGA Editor* router it uses to route the static design and the partial modules. Further work could be done to use Xilinx's *PAR* utility or the Torc Pathfinder router[30] to improve the timing performance of OpenPR designs. The next step is likely to add support for newer Xilinx architectures such as the Virtex and Spartan 6 series. Adding Spartan 6 support could be especially significant as Xilinx currently does not support partial-reconfiguration with these chips. Having this support could be a killer feature for OpenPR. Further work could be done to advance the functionality of OpenPR to support PR paradigms beyond the slot-based model, or to use OpenPR as the base for tool-flows that advance FPGA productivity. In fact researchers involved in the development of OpenPR are already working on an FPGA productivity tool-flow utilizing some of OpenPR's capabilities.

Alongside the technical opportunities provided by OpenPR, there is a unique opportunity to advance the development of open-source FPGA design tools. There is potential to create an entirely open-source tool-flow by combining long-standing projects such as VPR[53] with exciting new projects such as RapidSmith[31] , Torc[30], and of course OpenPR. By engaging the community in tool development, it is hoped that open-source FPGA development tools as a whole will reach the point where they can rival and force innovation from their proprietary

brethren.

Bibliography

[1] A. Cosoroaba, "Achieving Higher System Performance with the Virtex-5 Family of FPGAs," *Architecture*, vol. 245, pp. 1–12, 2006. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp245.pdf

[2] Frédéric Rivoallon, "Achieving Breakthrough Performance in Virtex-4 FPGAs," 2006. [Online]. Available: http://www.xilinx.com/support/documentation/white_papers/wp218.pdf

[3] Xilinx, "Virtex 6 Product Brief," 2009. [Online]. Available: http://www.xilinx.com/publications/prod_mktg/Virtex6_Product_Brief.pdf

[4] P. Thompson, "Manycore Processors Can Replace FPGAs," 2010. [Online]. Available: http://www.dsp-fpga.com/articles/id/?4632

[5] A. Lagger and A. Upegui, "Self-reconfigurable pervasive platform for cryptographic application," 2006.

[6] S. A. Fahmy, J. Lotze, J. Noguera, L. Doyle, and R. Esser, "Generic Software Framework for Adaptive Applications on FPGAs," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE, 2009, pp. 55–62. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290954

[7] M. French, E. Anderson, and D.-I. Kang, "Autonomous System on a Chip Adaptation through Partial Runtime Reconfiguration," in *2008 16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, April 2008, pp. 77–86. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4724891

[8] R. Garcia, A. Gordon-Ross, and A. D. George, "Exploiting Partially Reconfigurable FPGAs for Situation-Based Reconfiguration in Wireless Sensor Networks," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE, 2009, pp. 243–246. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290914

[9] J. Castillo, P. Huerta, V. Lopez, and J. Martinez, "A Secure Self-Reconfiguring Architecture Based on Open-Source Hardware," in *2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05)*, no. ReConFig. IEEE, 2005, pp. 10–10. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1592492

[10] K. Kepa, F. Morgan, K. Kościuszkiewicz, and T. Surmacz, "SeReCon : a Secure Dynamic Partial Reconfiguration Controller," in *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, 2008, pp. 292–297.

[11] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and J. Graf, "Wires on Demand: Run-Time Communication Synthesis for Reconfigurable Computing," *2007 International Conference on Field Programmable Logic and Applications*, pp. 513–516, Aug. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4380705

[12] C. Patterson and S. A. Guccione, "JBits" Design Abstractions," *FCCM*, 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=1058426.1058888

[13] K. Micha, "Secure Intellectual Property Management in Reconfigurable Computing Systems," Dissertation, National University of Ireland, Galway, 2010.

[14] V. Kindratenko, "Overview of Hardware Accelerators," in *Third International Workshop on High-Performance Reconfigurable Computing Technology and Applications*, 2009. [Online]. Available: http://wsdmhp09.hpcl.gwu.edu/kindratenko.pdf

[15] D. Curd, "Power Consumption in 65 nm FPGAs," 2007. [Online]. Available: www.xilinx.com/support/documentation/white_papers/wp246.pdf

[16] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, "Reconfigurable Computing: Architectures and Design Methods," *IEEE Proceedings* -

Computers and Digital Techniques, vol. 152, no. 2, p. 193, 2005. [Online]. Available: http://link.aip.org/link/ICDTEA/v152/i2/p193/s1&Agg=doi

[17] M. LaPedus, "Open-Silicon to Drive Down Mask Costs," 2007. [Online]. Available: http://www.eetimes.com/showArticle.jhtml?articleID=198900081

[18] D. Chen and J. Cong, "DAOmap: A Depth-Optimal Area Optimization Mapping Algorithm for FPGA Designs," *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pp. 752–759, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1382677

[19] G. Ni, J. Tong, and J. Lai, "A New FPGA Packing Algorithm Based on the Modeling Method for Logic Block," *2005 6th International Conference on ASIC*, pp. 788–791, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1611445

[20] S. Guccione, D. Levi, and P. Sundararajan, "JBits : Java Based Interface for Reconfigurable Computing," 1999.

[21] S. Singh and P. James-Roxby, "Lava and JBits: From HDL to Bitstream in Seconds," *FCCM*, 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=1058909

[22] N. J. Steiner, "A Standalone Wire Database for Routing and Tracing in Xilinx Virtex , Virtex-E , and Virtex-II FPGAs," Ph.D. dissertation, 2002.

[23] A. Poetter, "JHDLBits: An Open-Source Model for FPGA Design Automation," Ph.D. dissertation, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.9810&rep=rep1&type=pdf

[24] D. Koch, C. Beckhoff, and J. Teich, "Minimizing Internal Fragmentation by Fine-Grained Two-Dimensional Module Placement for Runtime Reconfiguralble Systems," *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, pp. 251–254, Apr. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290911

[25] D. Koch, C. Beckhoff, and J. Torrison, "Fine-Grained Partial Runtime Reconfiguration on Virtex-5 FPGAs," *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 69–72, May 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5474067

[26] T. Gingold, "GHDL," 2010. [Online]. Available: http://ghdl.free.fr/

[27] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon, *Odin II - An Open-Source Verilog HDL Synthesis Tool for CAD Research.* IEEE, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5474055

[28] J. Cong, "RASP," 2004. [Online]. Available: http://cadlab.cs.ucla.edu/software_release/rasp/htdocs/

[29] "VPR and T-VPack 5.0.2," 2009. [Online]. Available: http://www.eecg.utoronto.ca/vpr/

[30] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc : Towards an Open-Source Tool Flow," in *Nineteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, no. 1, 2011.

[31] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapid Prototyping Tools for FPGA Designs : RapidSmith," in *The 2010 International Conference on Field-Programmable Technology*, 2010, pp. 2–5.

[32] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," *2006 International Conference on Field Programmable Logic and Applications*, pp. 1–6, 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4100950

[33] E. Keller and S. Mcmillan, "An FPGA Wire Database for Run-Time Routers," in *5th Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD)*, vol. 95124, Laurel, MD, 2002.

[34] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose, "Vpr 5.0: Fpga cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, ser. FPGA '09. New York, NY, USA: ACM, 2009, pp. 133–142. [Online]. Available: http://doi.acm.org/10.1145/1508128.1508150

[35] Xilinx, "Virtex-5 FPGA Configuration User Guide (UG191)," 2008. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug191.pdf

[36] M. Royal, "XML: An Introduction," 2000. [Online]. Available: http://xml.gov/presentations/gsa/marion_royal_intro_to_xml.PPT

[37] Canonical, "Bazaar User Guide," 2010. [Online]. Available: http://doc.bazaar.canonical.com/bzr.2.2/downloads/pdf-en/bzr-en-user-guide.pdf

[38] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*, 2nd ed. Beijing: O'Reilly, 2008, vol. 5. [Online]. Available: http://http//svnbook.red-bean.com/

[39] Canonical, "Why Switch to Bazaar?" 2009. [Online]. Available: http://doc.bazaar.canonical.com/migration/en/why-switch-to-bazaar.html

[40] S. Chacon, "Git the fast version control system," 2010. [Online]. Available: http://git-scm.com/

[41] "Mercurial," 2010. [Online]. Available: http://mercurial.selenic.com/

[42] R. M. Stallman, R. McGrath, and P. D. Smith, "GNU Make," *Syntax*, 2010. [Online]. Available: http://www.gnu.org/software/make/manual/make.pdf

[43] D. Dye, "Partial Reconfiguration of Virtex FPGAs in ISE 12," 2010.

[44] "GCC, the GNU Compiler Collection," 2010. [Online]. Available: http://gcc.gnu.org/

[45] J. Solle, "Polymorphism," 2010. [Online]. Available: http://www.cplusplus.com/doc/tutorial/polymorphism/

[46] ——, "Friendship and Inheritance," 2010. [Online]. Available: http://www.cplusplus.com/doc/tutorial/inheritance/

[47] Javier Valcarce, "VHDL Macro: DC97," 2009. [Online]. Available: http://www.javiervalcarce.eu/wiki/VHDL_Macro:_DC97

[48] L. Torvalds, "The Linux Kernel Archives," 2010. [Online]. Available: http://www.kernel.org

[49] "Java," 2010. [Online]. Available: http://www.java.com

[50] "Visual C#," 2010. [Online]. Available: http://www.msdn.microsoft.com/en-us/vcsharp/default

[51] Xilinx, "UG748: Hierarchical Design Methodology Guide," 2010. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/Hierarchical_Design_Methodology_Guide.pdf

[52] J. Carver, N. Pittman, and A. Forin, "Relocation of FPGA Partial Configuration Bit-Streams for Soft-Core Microprocessors," in *WoSPS: Workshop on Soft Processor Systems*, 2008. [Online]. Available: http://www.eecg.toronto.edu/wosps08/papers/carver.pdf

[53] V. Betz and J. Rose, "VPR : A New Packing , Placement and Routing Tool for," in *Field-Programmable Logic and Applications*. Springer, 1997, pp. 213–222. [Online]. Available: http://www.springerlink.com/index/1673HWLMM7720606.pdf