OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs

Ali Asgar Sohanghpurwala*, Peter Athanas*, Tannous Frangieh* and Aaron Wood†
*Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061
Email: asohangh,athanas,tannous@vt.edu
†Information Sciences Institute East
Email: awood@east.isi.edu

***Abstract*—The Xilinx Partial Reconfiguration Early Access Software Tools for ISE 9.2i has been an instrumental package for performing a wide variety of research on Xilinx FPGAs, and is now superseded with the corresponding non-free add-on for ISE 12.3. The original package was developed and offered by Xilinx as a downloadable add-on package to the Xilinx ISE 9.2 tools. The 9.2i toolkit provided a methodology for creating rectangular partial reconfiguration modules that could be swapped in and out of a static baseline design with one or more PR slots. This paper presents a new PR toolkit called OpenPR that, for starters, provides similar functionality as the Xilinx PR Toolkit, yet is extendable to explore other modes of partial reconfiguration. The distinguishing feature of this toolkit is that it is being released as open source, and is intended to extend to the needs of individual researchers.**

I. Introduction

The Xilinx Partial Reconfiguration Early Access Software Tools for ISE 9.2i (here on referred to as the Xilinx PR Toolkit) has been a useful contribution to the reconfigurable computing community, and has served as the basis for many research projects. This toolkit, while not formally released with the ISE toolkit, was intended to augment the modular design flow. The underlying principle is that this toolkit enables 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 with this toolkit that can one-by-one populate a given slot. 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.

There have been many significant research projects that have explored new areas in reconfigurable computing using 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 [1] where hardware processing components are made to appear as software components in the runtime system, enabling their inclusion in adaptive applications. Full system autonomy was demonstrated via partial reconfiguration in [2]. The system presented here can potentially support hundreds more filters than a similar static implementation, and provides a reaction time improvement of 26-43x compared to non-autonomous systems.

In [3], 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 an improvement in performance compared to a microprocessor-based system. In [4], the authors show how an FPGA system based on an Open Source Open-RISC 1200 microprocessor can take advantage of partial reconfiguration to perform the secure transfer of the data needed to run an application. The authors of [5] propose a Secure Reconfiguration Controller (SeReCon) that provides secure runtime management of designs downloaded to the dynamically partially reconfigurable Xilinx device as a strategy to protect the design intellectual property. All of these endeavors are quite diverse in their strategies, uses of partial configuration, and experimental outcomes. The inflexibility of the slot-based model and the difficulty of use of previous Xilinx PR kits has been cited as a limiting factor in more widespread utilization of Partial Reconfiguration[6]. It is possible that researchers could achieve even greater impact in the domain of partial reconfiguration if the community had more control over the partial design creation process.

An alternative PR toolkit, called OpenPR is presented in this paper. OpenPR 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, and to help create the partial designs themselves. 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. OpenPR differs from previous PR tools like JBits[7] 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 paper first presents the central component capabilities of

OpenPR in Section III. A detailed step-by-step analysis of how the OpenPR process is used to create a baseline static design and a partial module is presented in Section IV. Sample results are presented in Section VII.

II. Related Work

A. 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 Xilinx development flow, including simulation[8], synthesis[9], mapping[10], as well as packing, placement, and routing [11]. 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 CAD tool research community.

B. Contemporary Extensions to Xilinx Tools

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 [9], [10], [11], the tools discussed below provide hooks into the Xilinx development flow that would allow FPGA CAD tools to support Xilinx FPGAs.

1) 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[12]." Torc builds greatly upon previous work, tracing its roots all the way back to the Alternate Wire Database (ADB) as presented in [13]. 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. When OpenPR is released to the Public, it will be based upon the official version of Torc released by ISI East.

2) 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[14]. 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 [14], RapidSmith can be used to analyze and create designs and design tools targeted towards Xilinx FPGAs.

III. Components

The OpenPR toolkit is written primarily in C/C++, and like the Xilinx PR Toolkit, is intended to augment the utilities provided within a contemporary ISE release. It is intended not to be tied to a particular version of the ISE tools. Also, like the Xilinx PR Toolkit, there are multiple components to the OpenPR toolkit to be used in the formation of the static design and the construction of the individual partial designs. This toolkit has been targeted primarily towards Virtex 4 and Virtex 5 devices, but is sufficiently flexible to accommodate other Xilinx architectures. The Xilinx PR Toolkit provides several important functions that are otherwise unavailable in the standard ISE modular flow:

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 discusses these functions and how they are implemented in OpenPR.

A. Identification of a dynamically reconfigurable geometric region

The first step in creating a slot-based partial reconfigurable design is to floorplan 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 the 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 floorplan 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 in this paper retain the AREA_GROUP constraint in defining the geometric reconfigurable region. This allows the user to floorplan the static and partial designs in PlanAhead, making it much easier to allocate the dynamic region while avoiding conflicts and resource scarcity. Figure 1 shows an example floorplanned static design with a dynamic region.

B. 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 the Xilinx Virtex devices offer

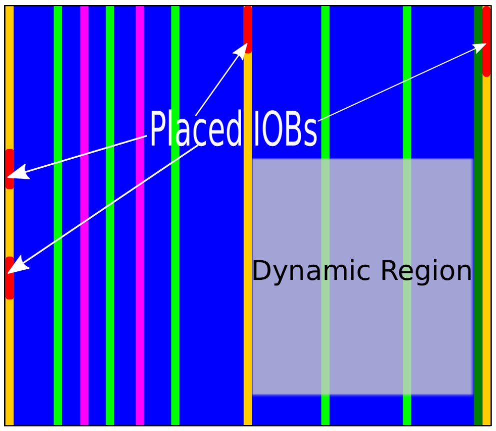


Fig. 1: Floorplanned static design

glitchless reconfiguration [15], the main issue is to prevent resource conflicts between static and partial designs. 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. Alternatively, 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.

Without the benefit of a PR-aware placer, it was necessary to find some way to constrain the standard ISE placer. There are documented constraints in the standard ISE tools that disallow placement of unrelated logic within an AREA_GROUP, however experiments with several versions of the tools confirmed that these are not always obeyed. Proper exclusion of static logic from the dynamic region requires a hard constraint that will always be followed. 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 throw an error. To ensure that no sites within the dynamic region were included in the static logic, a utility was developed to enumerate all sites within the candidate dynamic region and generate the necessary prohibit constraints. This utility uses a database similar to the Alternate Wire Database (ADB) [13] to store tile and site information that has been built from Xilinx XDLRC data. The automatically generated PROHIBIT constraints are then appended to the static design's constraint file.

The strategy adopted to block the router from entering the dynamic region was to enumerate all wires that pass from the static region into the dynamic region and prohibit them from being used in the static design. The aforementioned tile and wiring database is analyzed to enumerate all the wires that cross the dynamic region boundary and identify which Programmable Interconnect Points (PIPs) need to be enabled to use them, as shown in Figure 2. 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 cross a specified boundary. This is accomplished by introducing a single net into the design that is strategically routed in a way that consumes all available routing resources that could enter a designated area. By doing so, the ISE router is forced to only use resources outside of the dynamic region. This is accomplished by creating a temporary net that enables all the PIPs identified by the OpenPR route blocking utility. Then the router is allowed to run in a reentrant routing mode that will route around the blocking net. This concept was referred to as an *anti-core* in the JBits days [7].

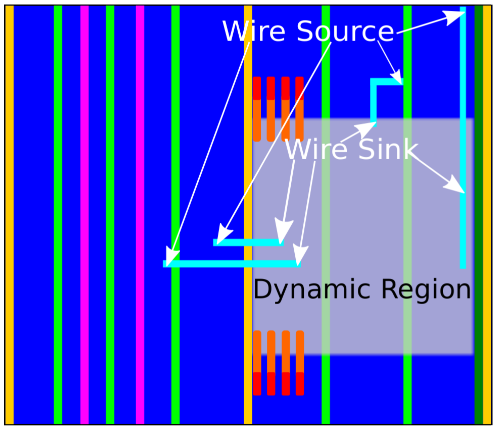


Fig. 2: Examples of wires that need to be excluded from static design

C. 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 the Xilinx 9.2 PR Toolkit, Xilinx employed bus macros which are hard macros placed in fixed locations at the regions inputs and outputs around the perimeter of the dynamic region. In this ISE 9.2 version of the toolkit, manual instantiation and placement of the bus macros was necessary. With the 12.3 release of the PR Toolkit, Xilinx introduced automatic instantiation and placement of bus macros in a manner transparent to the user. The bus macro concept is retained in the OpenPR toolkit. OpenPR macros are instantiated in the user's HDL code, and are placed automatically by OpenPR. The user must instantiate bus macros using the OpenPR naming convention where each bus macro has a standard name prefix with a unique integer suffix. OpenPR analyzes the boundaries of the PR region to find eligible sites for bus macro placement. Placement constraints are generated for the eligible sites and appended to the User Constraint File (UCF). Depending on the type and position of the eligible site, symbolic links are created in the build directory linking the appropriate hard macro type to the corresponding bus macro instance. Aside from the initial instantiation in the HDL code, this process is tranparent to the user. Figure 3 illustrates a static design with automatically placed bus macros.

Currently, dataflow through the region is limited to a vertical top to bottom flow, however this can be easily changed in the future with the addition of new bus macros and placement strategies.

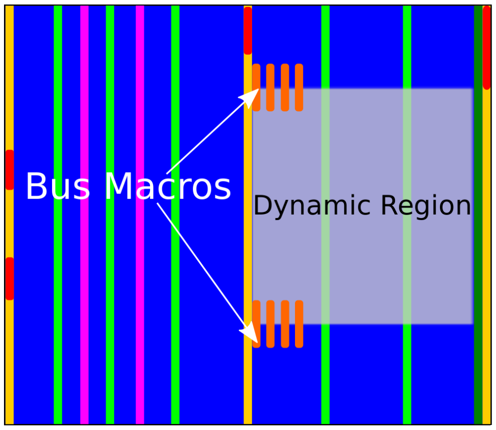


Fig. 3: Static design with automatically placed bus macros

D. 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 route blocker should prevent the router from using any wires that leave the dynamic region. Thus the same approach was taken, using the wiring database we enumerate all the wires that cross the region boundary and reserve them before routing the design. After trying some test designs it was determined that this might be too restrictive. 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 routing limitation then is that the partial module can use routing resources only if the PIP configuration bits that enable them lie inside the dynamic region. This approach reserves less routing resources, giving the router far more flexibility in routing the dynamic core.

E. Generation of Partial Bitstreams

To perform a runtime 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 generates from it a partial bitstream that only reconfigures the dynamic region. The granularity of this bitstream generation tool is limited by the Xilinx documentation on the bitstream format. In [16] 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; thus with the requirement that the reconfigurable region span multiples of clock regions, the OpenPR partial bitstream generator can generate a partial bitstream that only reconfigures the dynamic region. 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.

IV. The OpenPR Process

This section presents a step-by-step walk-through of the process of creating a static base design and a partial design. For the interest of brevity, this process is somewhat simplified setting aside some detail. An overview of the OpenPR flow can be seen in Figure 4.

A. OpenPR XML Project File

One goal of the OpenPR tools is to minimize the amount of manual intervention required to build a runtime reconfigurable system. In pursuit of this goal an XML based project file format was created. The user creates an initial file with some design parameters and at each step in the flow, the file is updated with additional information. A sample file is shown in Figure 5. This sample file shows a typical project file that includes the top level design name (<*designName*>), the static design path (<*staticPath*>), the path to the UCF (<*ucfPath*>), the prefix name of bus macros (<*busMacroPrefix*>), the name of the clock net(s), and the device name (<*deviceName*>) among other design parameters. XML was chosen because of its extensibility and ease of use. The use of XML means that the OpenPR project file can be easily modified by hand, by the tools, or with a third-party utility.

B. Floorplanning

Once the UCF for the static design has been created with the proper pin locations and timing constraints, Xilinx's PlanAhead utility can be used to floorplan the reconfigurable design. Using PlanAhead, the designer can draw the region as an AREA_GROUP on the FPGA in an area where there are no resources needed by the static design. The designer should then synthesize all of the partial modules and use the same PlanAhead project to verify that all modules will fit in

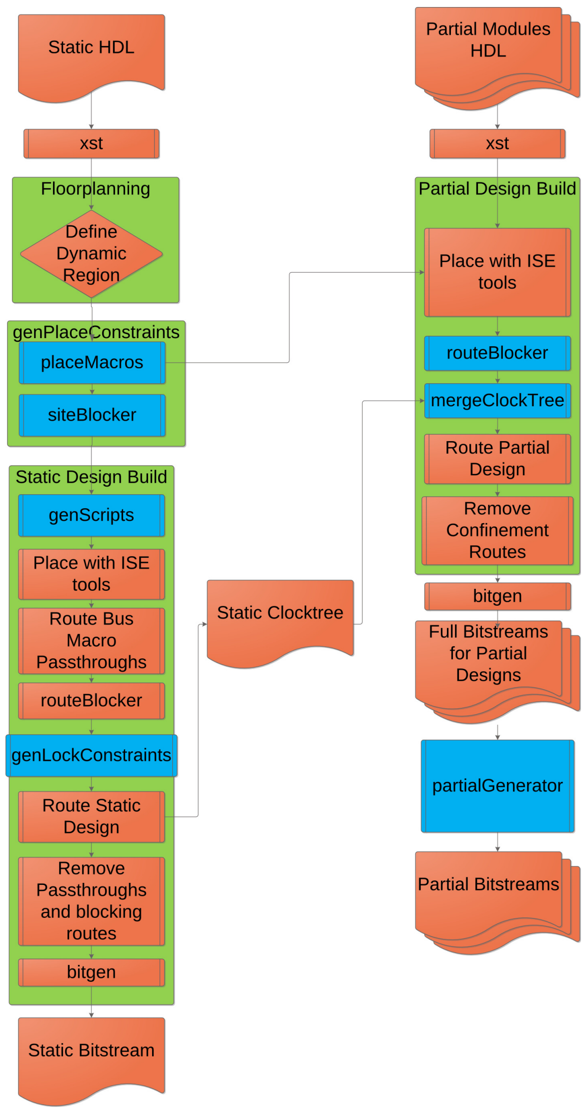


Fig. 4: Overview of OpenPR design process

the region. Once this is done, the designer can export the UCF from PlanAhead and continue with the build process. Manually editing the UCF to floorplan the design can be done instead. The Floorplanned design should appear similar to Figure 6a.

C. Static Design Flow

Once the OpenPR project file is created, all HDL source files have been compiled down to NGCs, and the floorplanning information is generated with PlanAhead or manually with a text editor, the designer can create the bit file for the static design simply with the command:

```
make static
```

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

```
<?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>
  <staticPath>/counter_sandbox</staticPath>
  <isPartial>0</isPartial>
  <ucfPath>/counter_sandbox/src/system.ucf</ucfPath>
  <dynamicAGName>counter</dynamicAGName>
  <busMacroPrefix>bm_xc5v_async_vert</busMacroPrefix>
  <passThroughNetName>data_temp</passThroughNetName>
  <clkNetNames class_id="1" tracking_level="0" version="0">
   <count>1</count>
   <item>clk_BUFGP</item>
  </clkNetNames>
  <deviceName>xc5vlx50t-ff1136-2</deviceName>
 </myProj>
</boost_serialization>
```

Fig. 5: Example project file

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 the CONFIG PROHIBIT constraints to exclude the dynamic region sites from the static design. Generated constraints are then appended to the UCF for the static design and Xilinx's *map* utility is run to place the static design.

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 toolkit automatically generates *FPGA Editor* scripts based on parameters in the OpenPR project file.

3) Route Passthroughs: Using an *FPGA Editor* script generated in the previous subsection, all passthrough nets in the design are pre-routed. These nets are pre-routed because they pass through the dynamic region and would be unroutable with the blocking routes in place. At this point the static design will resemble Figure 6b.

4) Create Blocking Routes: Once the static design is fully placed with all passthroughs routed, the *routeBlocker* utility is run, using one of the designated passthrough nets as a blocking net. The end result will be a dense structure that when instantiated into the design will force the Xilinx router to avoid it during the final assembly. Following the procedure outlined in Section III-B, *routeBlocker* enumerates all routing resources that cross the dynamic region border and uses them in the blocking net. This net is merged back into the static design XDL file and and NCD is generated. At this point the static design looks similar to Figure 6c. Note that in this figure that the blocking routes appear to enlarge the desired dynamic region, yet this is not the case. Segments that have one endpoint within the dynamic region and the other endpoint outside must also be blocked.

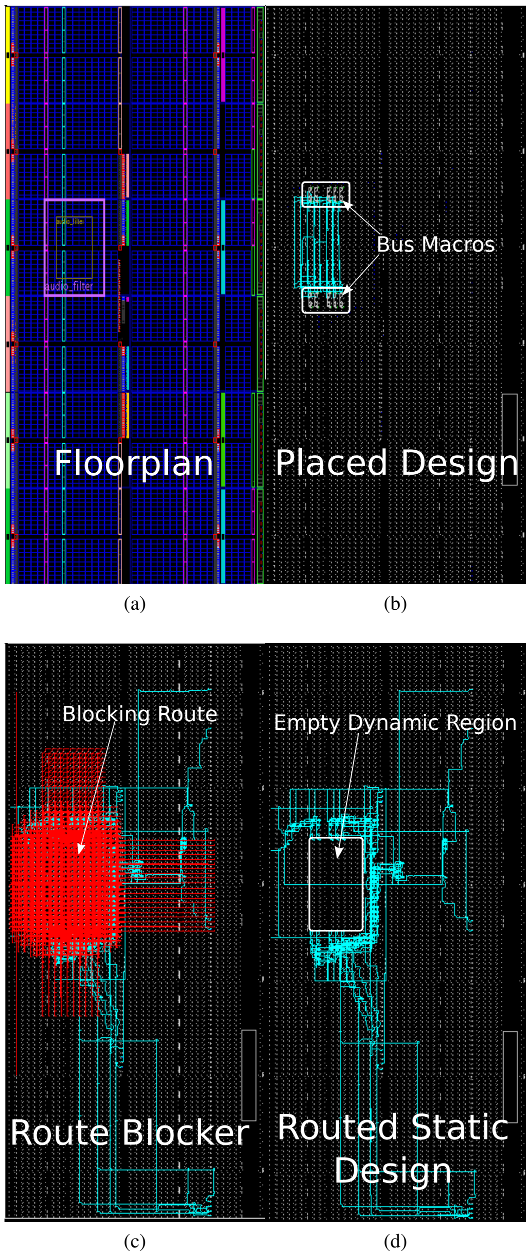


Fig. 6: Screenshots showing progression of OpenPR static design flow. a) static design floorplan in PlanAhead b) static design with bus macros placed c) static design with blocking route in place d) fully routed static design with empty dynamic region

5) Generate Lock Constraints for Blocking Routes: Using the defined passthrough nets from the OpenPR project file, LOCK constraints are automatically generated and appended to the Physical Constraints File (PCF). These constraints prevent *FPGA Editor* from unrouting any of the passthrough nets, including the blocking net.

6) Routing Design and Removing Passthroughs: With the blocking net locked down, *FPGA Editor* is invoked to route all remaining unrouted nets. Once the static design is routed, the blocking nets and passthrough nets can be removed from the design. The resulting final static design will look similar to Figure 6d.

D. Partial Module Design Flow

Once the base static design is complete, the process of creating the partial designs that can populate the slot in the static design is straight-forward:

```
make partial
```

As with the static design, the Makefile fully automates the construction process, and invokes a series of OpenPR utilities and Xilinx utilities. The remainder of this subsection examines the lower-level OpenPR steps and utilities that accomplish this.

1) Merge Static Clocktree into Partial Design: The *merge-ClockTree* utility extracts the clock tree routing information from the static design and inserts it into the placed partial design file. Since the partial design is routed using reentrant routing, the static clocktree will be preserved. The resulting partial design clocktree will be a superset of the static clock-tree.

2) Generate Partial Bitstreams: Finally, a partial bitstream is generated to reprogram the FPGA with configuration data for the partial module and its clock tree. Dynamic region co-ordinates are read from the OpenPR project file and bitstream frames corresponding to the tiles within the dynamic region are written to the partial bitstream along with the corresponding clocktree configuration frames.

V. Implementation Choices

One of the major reasons C++ was chosen as the primary language for OpenPR was performance. This work uses a tile and wire information database similar to the one presented in [13] where the author posited that “performance and memory overhead would be improved if C++ was used instead of Java.” All OpenPR utilities were written in portable, object-oriented C++. While targeted towards gcc on the Linux platform, efforts were made to ensure that all code was portable and only used cross-platform libraries such as *boost*. Building and running the OpenPR tools on other Xilinx supported platforms such as Windows (using Cygwin) should be fairly trivial.

The abstract interface employed by the tile and wiring database is expressive enough to describe devices from all post-Virtex Xilinx architectures. This means that adding support for new devices should only require parsing the XDLRC data to create new device information databases. When possible, inheritance was used to organize device and architecture specific functionality and attributes. Hierarchical organization makes it easier to add new devices and architectures without having to rewrite major portions of code. Developers wishing to extend OpenPR’s functionality can inherit the existing OpenPR classes and override only the functionality they wish to change.

TABLE I: Comparison of OpenPR to Xilinx PR tool kits

Feature	OpenPR	Xilinx 9.2i PR	Xilinx 12.3 PR
Bus Macro Placement	Automatic	Manual	Automatic
Cost	Free	Unavailable	Price varies
Extensible	Yes	No	No
Device Support	Virtex / Expandable	Virtex	Virtex
Bitstream DRC	No	Yes	Yes
Router Performance (Timing Closure)	Poor	Good	Great

TABLE II: Designs implemented using OpenPR

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

VI. Comparison to Xilinx PR Flows

OpenPR was developed with different overall goals than the Xilinx PR tool kits. While the initial OpenPR release implements the slot-base partial reconfiguration methodology, OpenPR is intended to be a starting point for research into new FPGA design paradigms. Even so, OpenPR offers a competitive feature set when used for slot-based PR designs. Table I compares OpenPR to Xilinx 9.2i and 12.3 PR toolkits.

The main disadvantage of OpenPR is the current dependence on the *FPGA Editor* router employed. Designs that meet timing closure easily with the Xilinx PR tools, might not meet timing when built with OpenPR. The main advantage of OpenPR is its capability to be expanded with new features and device support. While the reliance on the *FPGA Editor* router is potentially a disadvantage for large high-speed designs, the authors are currently working on a higher-performance alternative combining Xilinx's PAR and the Torc PathFinder router[12] to replace *FPGA Editor* greedy router.

VII. Results

To demonstrate the OpenPR toolkit capabilities, three designs were implemented on three different FPGA development boards, all featuring Xilinx V5 FPGAs: the XUPV5-LX110T, the Genesys V5-LX50T, and the Avnet V5-SX95T development kits. Table II lists all three designs along with the development kits on which each design was implemented.

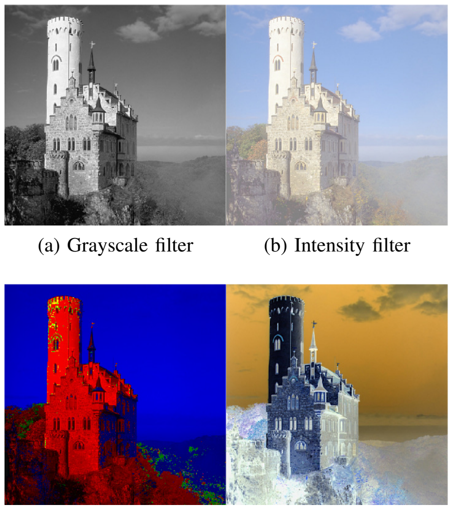
For the sake of brevity, we only detail the video filters example. Implemented on the XUPV5-LX110T development board using OpenPR, the video filters application processes in real-time a video signal in RGB format, applies a filter to that signal then outputs the results to a display. The application consists of four video filters implemented as partial modules: a grayscale, an intensity, a maxRGB, and a negative filter. Table II shows the partial modules for each of the implemented designs. The baseline static design consists of

TABLE III: 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

logic to extract the RGB signal and feed it to the filter, grab the outcome of the filter and display it, and the necessary glue logic. Figure 7 shows a screenshot of the signal fed to the video application. The outcome of the different filters partial bitstreams is illustrated in Figure 8.

Fig. 7: Original image [17]


(a) Grayscale filter (b) Intensity filter

(c) MaxRGB filter (d) Negative filter

Fig. 8: Screenshots showing the outcome of the video filters

In order to evaluate the speed of the bitstream generation process using OpenPR, the video filters application was implemented using Xilinx 9.2i PR as well. The OpenPR developers have been focusing on functional completeness and bitstream generation times presented in Table III represent mostly unoptimized code. Even so, OpenPR presents bitstream compilation times that are competitive with the Xilinx 9.2 PR flow, with less than 10% increase in the time to generate a bitstream. With some optimization work it should be possible to make OpenPR bitstream compilation times lower than the Xilinx PR flow numbers.

It is important to note some of the other disadvantages OpenPR has compared to the Xilinx PR flow. OpenPR has little design rule checking (DRC) capability beyond the DRC check performed by the Xilinx *MAP* utility after packing and placement. This should ensure that there are no DRC violations within the partial modules or within the static design. However, since there is no DRC check performed during partial bitstream generation, it is the user's responsibility to ensure that no DRC violations (multiple drivers, etc.) occur when the partial module is programmed into a running system. Also OpenPR currently uses the *FPGA Editor* router that can have shorter runtimes, but is less capable of meeting timing closure than the *PAR* router used by the Xilinx PR Flow.

VIII. Conclusions & Future Work

This paper has presented an open-source, slot-based, partial-reconfiguration toolkit. The presented toolkit has been verified to work with several different tool versions, development boards, and example designs. The toolkit is composed of several utilities written in portable object-oriented C++ tied together via scripts and Makefiles. The initial release includes minimal automation and only supports the inflexible slot-based paradigm of partial reconfiguration. However, this work provides a baseline that researchers could use to explore advancements in the areas of runtime-reconfiguration and FPGA design productivity for Xilinx FPGAs. For example, OpenPR ability to create an FPGA sandbox that excludes logic placement and routing is an appealing feature currently employed in an NSF Center for High-Performance Reconfigurable Computing (CHREC) productivity project. OpenPR is packaged in a form that makes it ready to use for traditional slot-based partial reconfiguration.

The real potential of OpenPR lies in its ability to be expanded, adding support for new devices and paradigms. Adding support for devices such as Spartan 6 could be especially significant as Xilinx currently does not support partial-reconfiguration with these chips. Further work could be done to improve the timing performance of OpenPR designs by using Xilinx's PAR utility or the Torc Pathfinder router[12]. Advancing 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 are potential areas to explore. Alongside the technical opportunities provided by OpenPR, there is a unique opportunity to advance the development of open-source FPGA design tools. 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.

IX. Acknowledgments

The authors would like to thank USC-ISI East for supporting this research. The authors would also like to thank the Brigham Young University Configurable Computing Lab for their perseverance in discovering neat new things. The authors would like to convey a special thanks to Neil Steiner and Aaron Wood from ISI for providing excellent tools for highly innovative approaches to FPGA routing.

References

[1] 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

[2] 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

[3] 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

[4] 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)*. IEEE, 2005, pp. 10–10. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1592492

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

[6] 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

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

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

[9] 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

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

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

[12] 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*, February 2011.

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

[14] 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.

[15] 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

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

[17] (2001) Castle lichtenstein. [Online]. Available: http://famtille.de/deutschland/sueden/2001_046.html