EuTOPIA: Facilitating Processor-Based DPR Systems for non-DPR Experts

Edward Chen, William Gruver, and Lesley Shannon
School of Engineering Science, Simon Fraser University
Burnaby, BC, Canada
{ekchen, gruver, lshannon}@sfu.ca

Dorian Sabaz
Intelligent Robotics Corporation
North Vancouver BC, Canada
dorian@iroboticscorp.com

Abstract

Dynamic Partial Reconfiguration (DPR) allows the targeted Field- Programmable Array (FPGA) to selectively update portions of the programmable logic while the remainder of the fabric stays active. Previously, designers were required to manually create DPR systems with minimal support from Computer Aided Design (CAD) tools. Xilinx®'s PlanAhead™ software now automates much of the DPR system generation process for ISE-based DPR systems.

*This paper presents EuTOPIA (**EDK TO P**lanAhead **I**mplementation **A**utomation), a CAD tool that extends the automation in Xilinx's current DPR design flow to include microprocessor based designs. Given an architecture described in Xilinx's Embedded Design Kit (EDK), EuTOPIA automates the process of converting it into a PlanAhead project for a DPR design. EuTOPIA abstracts the low level details of DPR implementation so that designers need only have nominal knowledge of the underlying process to create their initial DPR system. EuTOPIA is shown to reduce implementation time by a factor of 18 times over manual implementations by experienced designers, and can be further increased for novice designers. Over the design-cycle of a product where designers may generate tens or even hundreds of different implementations of their DPR design, these time savings could be extremely significant.*

Keywords: *Field Programmable Gate Arrays, Dynamic Partial Reconfiguration, Computer Aided Design (CAD) Tools, SoC Design, Soft Processors*

1. Introduction

Field Programmable Gate Arrays (FPGAs) using SRAM technology provides designers with a programmable fabric that can be configured to implement a specific hardware circuit. This technology also allows designers to reconfigure the FPGA for a new application's circuit when the current application has been completed. In addition to reconfiguration between applications, Xilinx®'s FPGAs allow Dynamic Partial Reconfiguration (DPR) of the FPGA fabric. DPR allows selected areas of the target device to be reprogrammed while the remainder of the fabric stays active. DPR enables multiple modules to time-share the same physical area on the target device, virtually increasing its physical resources.

Another benefit of DPR systems is that they provide applications with an adaptive hardware system. The design is able to adapt to runtime system conditions and update only the necessary portions of the hardware. This has important applications in areas such as cryptography [13] and reconfigurable communication systems [8]. For modern electronics, the concept of sharing the physical hardware among different applications is appealing as it could reduce area, cost, device count and power dissipation [3,4,7,8]. Configuration time is also reduced since it is directly proportional to the physical area being reconfigured [8]. If an application does not fit on the available FPGA fabric, DPR may be used to virtually provide the necessary resources without requiring the application to stall as the entire FPGA is being reconfigured at runtime.

DPR systems mainly exist only in research and have yet to become a popular design solution for commercial products. Typically, they are extremely complex to implement and often it is possible to purchase larger devices with sufficient logic resources instead of having to use DPR to virtually increase them. However, as power becomes an important factor for many designs especially in portable electronics [4], virtually increasing the available logic resources using DPR may become a

practical solution. Xilinx has created documentation outlining the exact procedure to be followed when implementing an ISE-based DPR design [1]. Furthermore, PlanAhead™, their hierarchical floorplanning tool, can be used to automate some of the steps of the DPR design flow to reduce the design complexity. However, if the DPR design is a microprocessor-based system developed in Xilinx's EDK, additional steps are required before it can be exported to ISE. No tool support for DPR systems that include processors is currently available. As an increasing number of FPGA designs include processors, there exists a need to facilitate the generation of potential DPR-systems developed in EDK.

This paper presents EDK TO PlanAhead Implementation Automation (EuTOPIA), a CAD tool that extends the automated tool flow for DPR systems to include processor-based systems created in EDK. Our previous work provided only a brief overview of *EuTOPIA* [13]. Here, we extend the original work to include specific details on *EuTOPIA*, such as its usage and operation as well as its software architecture. From our analysis, *EuTOPIA* has been demonstrated to reduce design time for the initial DPR system by a factor of 18 times over manual implementations by experienced designers, and can be further increased for novice designers. By abstracting the low-level details, *EuTOPIA* allows designers to quickly implement a preliminary DPR system. These time savings can be extremely significant during the design life cycle where different versions of the DPR system may be created during product development. Furthermore, once *EuTOPIA* has generated the initial DPR system, experienced designers can utilize PlanAhead to tweak the initial implementation and improve performance.

The remainder of this paper begins with Section 2 describing previous research using DPR and the DPR design flow that Xilinx's CAD tools currently support to implement a DPR system. Section 3 provides an overview of how *EuTOPIA* is used to extend the automation of Xilinx's current DPR design to support systems that include processors. As *EuTOPIA* has two distinct phases, Section 4 describes the tasks performed in Phase 1 and Section 5 provides an overview of Phase 2. Section 6 provides the implementation details of *EuTOPIA* and the generated DPR systems structure. Section 7 describes a case study of a DPR system that was implemented manually and using *EuTOPIA* by both experienced and novice designers. Finally, Section 8 summarizes the conclusions and future work of this project.

2. Dynamic partial reconfiguration design

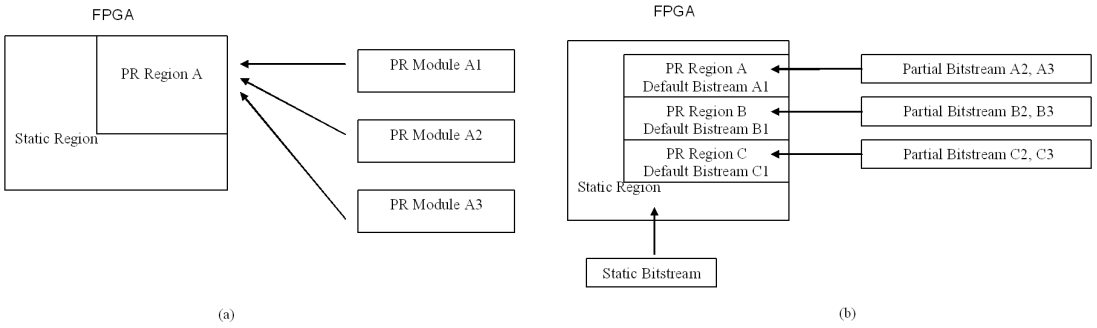

Figure 1. Relationship between (a) PR Region and its PR Modules, and (b) Partial bitstream for each PR Module and the Static Region

To clarify the discussion of DPR in this paper, it is important to first define the underlying terminology used. For this discussion, a *Partially Reconfigurable* Region (PR Region) refers to a physical area of the FPGA that can be dynamically reconfigured to implement different tasks. A Partially Reconfigurable Module (PR Module) is one of the possible tasks that can be implemented in a PR Region. These concepts are illustrated in Figure 1(a). Each PR Module has its own bitstream file, or *partial bitstream*, as does the static portion of the design (the *static bitstream*). A *full bitstream* for the FPGA comprises the static bitstream plus default partial bitstreams for each of the PR Regions shown in Figure 1(b). The remainder of this section

describes previous research applications using DPR, the steps involved in designing DPR systems, and current Computer Aid Design (CAD) tools available to implement them.

2.1. Previous work

The previous work summarized here considers both the usage of DPR in different applications and attempts to automate the DPR process.

2.1.1. Partial reconfiguration automation

To date, the only commercially available FPGAs capable of implementing a DPR design are available from Xilinx. They provide documentation to describe the concepts and complex steps required to implement such a system [1]. Xilinx has also recently introduced the PlanAhead software tool initially designed for complex systems requiring multiple designers. The target device is partitioned into different physical regions and each designer is dedicated to a particular region. The partial designs are then merged together into a final complete system.

PlanAhead can also now be used to facilitate the development of an ISE-based DPR system. It reduces the complexity of ISE-based DPR system development and offers more flexibility, automation and system performance. However, currently there is no direct support for processor-based DPR systems in PlanAhead. As processor-based systems become more common, there exists a need for designers to be able to quickly generate their designs.

2.1.2. Partial reconfiguration applications

DPR has been used in applications such as cryptography [12], network security [143], reconfigurable communication [8], and aerospace [8]. In Zeineddini [11], partial reconfiguration is used to devise a secure reconfiguration scheme that minimizes reverse engineering and bitstream cloning. Encrypted partial bitstreams are loaded via an embedded microcontroller so potential hackers are unable to obtain full configuration data of the target device. In Kshirsagar et al. [12], a general processor is combined with a reconfigurable co-processor for enhanced performance and fault tolerance. In Kao [8], partial reconfiguration is the basis for the defense industry's Joint Tactical Radio Systems (JTRS). DPR satisfies the need for a reprogrammable runtime environment and the ability to support multiple channels and network protocols simultaneously. Without partial reconfiguration, independent processing resources and hardware logic would be required for each additional supported channel. This would adversely affect space, weight and power consumption.

Kao [8] also discusses how applications designed for the aerospace industry may be subject to Single Event Upset (SEUs) that may occur from in-orbit, space-based, and extra-terrestrial applications. By comparing the stored and the actual configuration bits, the system can detect and repair SEUs using partial reconfiguration without disrupting normal operation. The system is also able to avoid completely reconfiguring itself if only minimal changes are required.

2.2. Partial reconfiguration design flow

For the purpose of this paper, the *ISE-based standard design flow* is the set of procedures used to implement a non-DPR design using ISE. The *EDK-based standard design flow* is the set of procedures used to implement a non-DPR, microprocessor-based design from EDK. Figure 2(a) shows the high level overview of the standard EDK-based design flow [9]. Modifications are required to this design flow if the processor-based design implemented in EDK is to leverage DPR. Figure 2(b) illustrates the required modifications to the standard design flow required to implement a microprocessor-based DPR system from EDK [10]. The bolded boxes highlight the modifications to the standard EDK-based design flow for DPR system implementation.

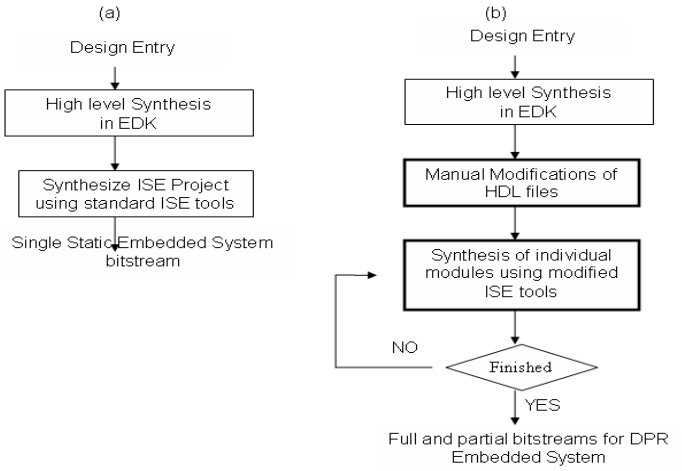

Figure 2. High level EDK-based design flow for (a) standard non-DPR systems and (b) the modified design flow for EDK-based DPR systems

The designers start by implementing a standard EDK project used for a static embedded system. Manual modifications of the HDL files are then required before the EDK project is exported to ISE for synthesis. The static module is first placed and routed. It is prohibited from using logic resources in the PR Region. However the static module can share the routing resources contained in the PR Region to achieve better system routability and performance. Each of the PR Modules is then placed and routed, and its partial bitstream is generated according to its individual User Constraints File (UCF). The UCF for a PR Module specifies, among other attributes, its physical location and timing constraints. A partial bitstream is created for each PR Module. A full bitstream containing the configuration of the entire design including a default PR Module for each PR Region is also created. This full bitstream is typically used as the default configuration when the system initializes.

2.3. Partial reconfiguration structural description

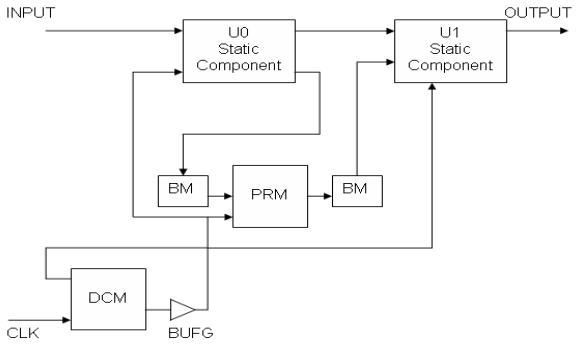

Figure 3. Top level design example

Figure 3 shows the top level design of a typical DPR system. The DPR design flow requires a strict hierarchical approach that must be closely followed during the HDL coding process. Global logic such as I/Os, clocks, and Digital Clock Managers (DCM) must be in the top-level module contrary to EDK's default assignment. The Bus Macros (BMs) that are required to provide communication between the PR Regions and the static module are also placed at the top level. There can be multiple components within the static-design module. The PR Regions are instantiated as *black-boxes* in the top level.

2.4. PlanAhead

Xilinx introduced the PlanAhead software to facilitate complex designs that require multiple designers [8]. PlanAhead partitions the target device into multiple physical regions that can be implemented independently and merged in later steps to form a complex, multi-person design. PlanAhead can also be used to streamline the process of implementing a DPR system. It exists as an independent tool to complement the ISE CAD tools.

PlanAhead abstracts the intricate details of the implementation process and provides a Graphical User Interface (GUI) throughout the design flow. PlanAhead uses the concept of Physical Blocks (PBlocks) to constrain logic within the pre-defined boundaries on the target FPGA. Each PBlock is flexible in size. The static module and each PR Regions are constrained into separate PBlocks. After verifying the design using PlanAhead's Design Rule Checks (DRCs), the partial reconfiguration flow wizard performs the actual implementation of the design. The resulting PlanAhead project will contain the full bitstream of the design and partial bitstreams for each PR Module generated. To implement a processor-based DPR design using PlanAhead, the Block RAM (BRAM) Memory Map (BMM) file must be included during the synthesis of the static region of the design. Otherwise, the contents of the BRAMs will not be properly initialized after the FPGA is initialized with the full bitstream.

3. Overview of *EuTOPIA*

Xilinx has simplified the complex process of implementing an ISE-based DPR system through the use of PlanAhead. However, detailed knowledge of their CAD flow is still needed to generate a DPR system from a processor-based design. *EuTOPIA* abstracts these low level details, allowing designers to quickly generate their EDK-based DPR designs. The high-level overview of *EuTOPIA* is shown in Figure 4.

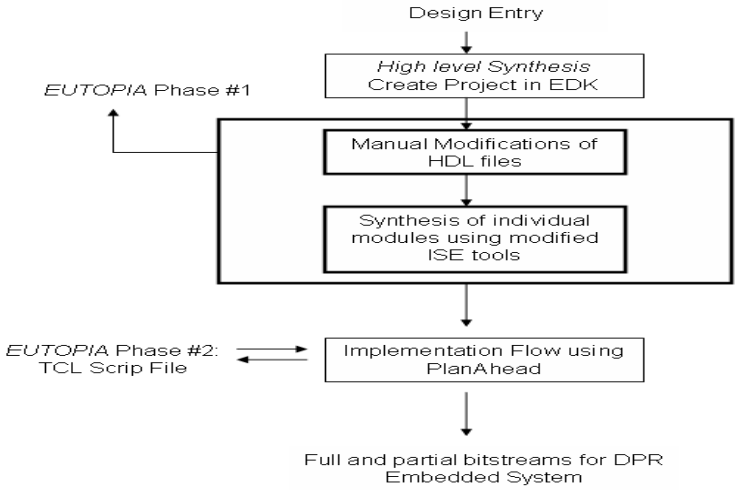

Figure 4. High-level overview of *EuTOPIA*

EuTOPIA automates the process of converting an EDK-based DPR design to a PlanAhead project. It uses a two-phase approach to generate the PlanAhead project and full and partial bitstreams. In Phase#1, *EuTOPIA* modifies HDL files in the EDK project to conform to the DPR design flow. The modified HDL files are then synthesized using ISE. A TCL script that contains detailed implementation instructions for PlanAhead is also generated. In Phase#2, PlanAhead invokes the TCL script to create the PlanAhead project. Currently *EuTOPIA* supports only the Peripheral Local Bus (PLB) and the On-chip Peripheral Bus (OPB). Future versions will incorporate additional communication infrastructures such as the Fast-Simplex Links (FSLs). *EuTOPIA* has been designed to require no updates for newer releases of EDK, provided that there are no major structural changes to the HDL files it generates. *EuTOPIA* is not designed as a replacement for either EDK or PlanAhead, but is meant to complement the functionality provided by both tools. It is designed to provide novice

users with only the core functionality and flexibility necessary to generate a DPR design. After the initial DPR system is established, the users can pursue more optimized designs by modifying the existing PlanAhead project.

4. *EuTOPIA* – phase #1

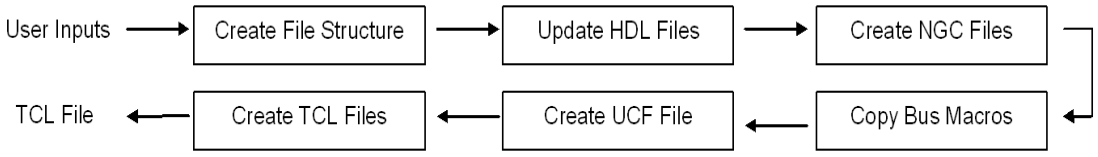

Figure 5. High-level overview of *EuTOPIA* phase#1

For a DPR design developed in EDK, Phase#1 modifies and synthesizes the HDL files in EDK to conform to the DPR design flow. The design constraints are specified in the UCFs and a TCL script is generated for Phase#2. The following subsections detail the actions taken in Figure 6.

4.1. User inputs

EuTOPIA currently accepts the following design parameters through a simple text based user interface:

- Development Board Type
  - Virtex-2 Pro
  - ML505
- Number of Partial Reconfigurable Regions (Up to 3)
- Desired location of the PR Regions (Top right, top left, bottom right or bottom left)

In the current version of *EuTOPIA*, all PR Regions are fixed in size and have to reside in the same vicinity of the FPGA (e.g. top right, bottom left). Future versions will allow users to implement an increased number of PR Regions, additional development boards, and specify the desired location and size of each PR Region individually.

4.2. Create file structure

The file structure of *EuTOPIA* is shown in Figure 6. Folders generated by *EuTOPIA* are shaded in grey.

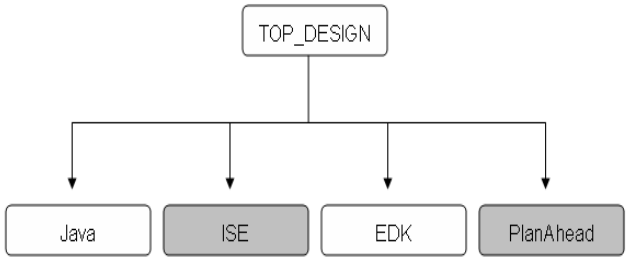

Figure 6. *EuTOPIA* file structure

To automate the DPR flow, it is required that the user place the EDK project and the Java folder inside the *TOP_DESIGN* folder, where the *TOP_DESIGN* is the name of the DPR design. The Java folder contains *EuTOPIA* and BMs for all supported development boards. All subsequent files and directories will be created automatically. The modified HDL files and resultant NGC files from Phase 1 synthesis are placed in the ISE folder. The PlanAhead folder will contain the DPR project implemented in PlanAhead. Designers are able to make further modifications to the PlanAhead project

as desired after *EuTOPIA* has completed execution. Currently, the Java folder is required for *each* DPR design. *EuTOPIA* is now being updated to require only one instance of the *EuTOPIA* executable and BMs for all DPR designs that reside on the local host.

4.3. Update HDL files

Hardware Description Language (HDL) files generated by EDK need to be modified to conform to the DPR design flow. The HDL file *system.vhd* is generated automatically by EDK when the EDK-based design is synthesized as a sub-module of a top-level design, *system_stub.vhd*, as shown in Figure 7. *Systemstub.vhd* contains high-level port mapping of the underlying modules. Without modifications, the only component included in this top level is *system.vhd*, which includes both the static and dynamic components of the complete DPR system. Changes are required for *system_stub.vhd* to comply with the DPR design flow. In particular, this top-level design must explicitly contain all:

- I/O instantiations
- DCMs and BUFGs
- Static module
- PR module instantiations
- Global signal declarations
- Bus macro instantiations

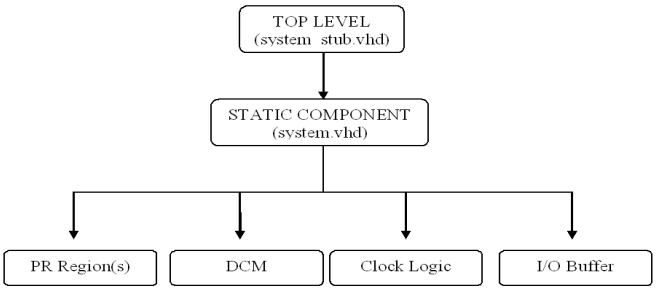

Figure 7. Design hierarchy of *systemstub.vhd* before modification

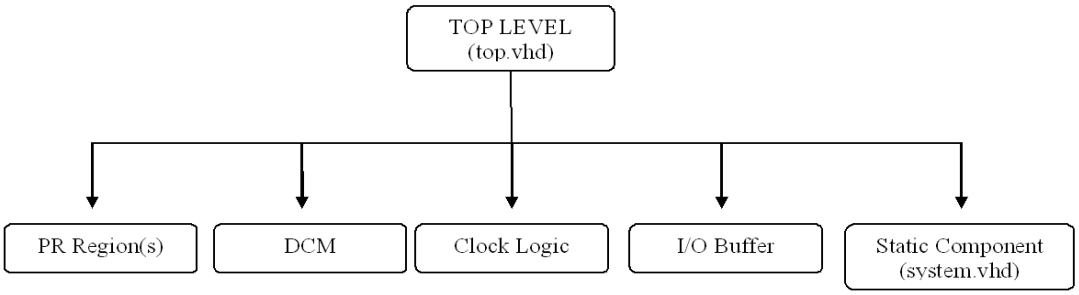

Figure 8. Design hierarchy of *systemstub.vhd* after modification

Figure 8 shows the logical hierarchy after modifications are made to *systemstub.vhd,* which is then renamed as top.vhd and placed in the ISE folder for later synthesis. First, *system.vhd* has been updated to just include the static component. The PR Regions and I/O ports are removed from *system.vhd* and elevated to the top level (*top.vhd*). Next, the BMs are placed with the correct nets between the PR Regions and the static module. Two unidirectional BMs are required for each PR Region to provide duplex communication. The DCM is also instantiated at the top level and internal signals are adjusted to ensure that connectivity between components has not been disrupted.

4.4. Create NGC files

This step utilizes the modified HDL files and ISE to produce the necessary NGC files for PlanAhead in Phase 2. If there are *p* PR Modules in the system, there will be a total of *p+2* NGC files created. The two additional NGC are files for the top-level HDL (*top.vhd*) and the static components *(system.vhd)*. *EuTOPIA* invokes command-line calls to the Xilinx Synthesis Tool (XST) to synthesize and create the NGC files. The script file (.scr) contains executables and options available within XST. Options include optimization mode, target device, IOBUF enable, speed setting...etc. Currently, *EuTOPIA* provides defaults for all the options. Future versions of *EuTOPIA* will allow users to specify parameters for each option. Note that only the *top level* (*top.vhd*) synthesis should have IOBUFs enabled.

4.5. Copy bus macro

Device specific BMs are used to provide unidirectional point-to-point communication between a PR Region and the static module. All connections between the PR Regions and static module must pass through a BM, with the exception of clock signals and power. There are three parameters for the BMs available for each of the supported FPGAs:

- Signal Direction -- Left to Right (L2R), Right to Left (R2L). Used as inputs or outputs to the PR Regions with respect to the position of the PR Regions and the static design.
- Physical width of the BM
  Wide – 4 Configurable Logic Blocks (CLBs) wide
  Narrow – 2 CLBs wide
- Synchronous versus asynchronous

In total, there are 2^3=8 variations of the BMs available for each supported FPGA. Currently, *EuTOPIA* automatically assigns narrow, synchronous BMs (both L2R and R2L for signal inputs and outputs) for each PR Region. The default BM parameters are chosen to minimize design footprint and to reduce system complexity that can arises from an asynchronous design. All supported FPGAs' BMs are contained in the Java directory. Future versions of *EuTOPIA* will allow designers to specify the types of BMs that designers consider best suited for their designs. The appropriate BMs are then copied into the ISE folder for synthesis.

4.6. Create user constraints files

PlanAhead requires the DPR design to be budgeted and partitioned before implementation. The UCF specifies the physical location on the FPGA and the constraints for the place and route tool. *EuTOPIA* automatically assigns the location of all PR Regions to one user-specified region of the FPGA (e.g. top left). Clock BUFG and BM locations and parameters are also automatically assigned with respect to the target FPGA and PR Regions' locations. Designers can choose new locations after the PlanAhead project has been created to further optimize their designs.

4.7. Create script file for PlanAhead

A script file (.tcl) detailing the steps to be performed by PlanAhead is created and placed in the PlanAhead folder. This file will be invoked by the PlanAhead software in the next phase.

5. *EuTOPIA* – phase #2

PlanAhead performs a fixed set of operations outlined in the TCL file. A bottom-up approach is used to implement the DPR design. After Phase#1 is completed, the user invokes the TCL file using PlanAhead to create the PlanAhead project. Users are able to validate their designs and make further modifications as necessary. Figure 10 shows the high-level overview of the major steps required in PlanAhead to generate the PlanAhead project that contains the full and partial bitstreams.

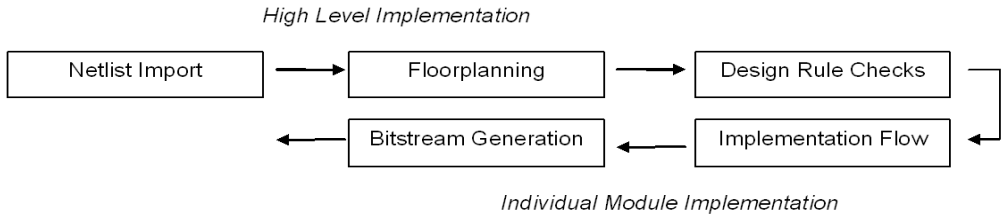

Figure 9. High-level overview of PlanAhead steps

An initial project is first created according to the development board type, number and location of the PR Regions and PR Modules. Netlists are imported and floorplanned from the constraints contained in the UCF. DRCs are used to ensure the entire design conforms to the DPR design flow. The static module and individual PR Modules are then implemented separately. Bitstreams are generated using PlanAhead-specific commands provided in the *EuTOPIA* generated TCL script.

5.1. High level implementation

The initial PlanAhead project combines the static module with one of the possible PR Modules for each corresponding PR Region to generate an initial full bitstream. Alternate PR Modules are imported from the ISE folder. All netlists are imported as either *EDIF* or *NGC* files. The design is then properly floorplanned and partitioned according to the UCF. PlanAhead provides internal DRCs to verify the correctness of the design. Rule checks include Bus Macro DRCs, Floorplanning DRCs, Glitching logic DRCs and Timing Advisor DRCs.

5.2. Individual Module Implementation

The static logic module is placed and routed first followed by each of the PR Modules. PlanAhead's *ExploreAhead* tool implements and displays the status of the progress in real-time. BMM files for the static module must be included into the static module's implementation to ensure the processor's on-chip executable is present when the FPGA s initially configured. After the static module and the PR Modules have been implemented, the full and partial bitstreams are generated by using PlanAhead's *PR_verify* and *PR_assemble* command. *PR_verifydesign* is run to generate partial bitstream for each of the PR Module. *PR_assemble* is run to generate the full bitstream containing the static module and the selected PR Module as the initial configuration of the target device.

5.2. PlanAhead file structure

PlanAhead automatically creates an array of files and folders to store the DPR system. Figure 10 shows the important folders within the PlanAhead project.

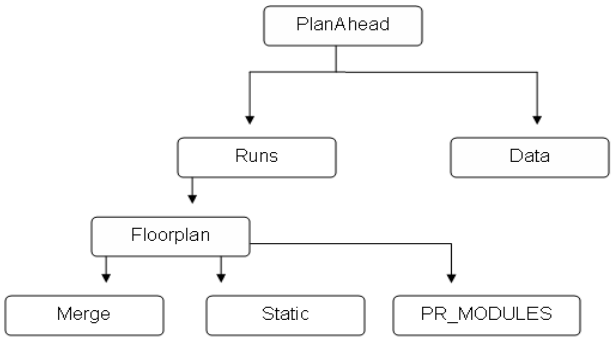

Figure 10. PlanAhead file structure

The *data* directory contains the implementation details of design (e.g. Netlists, *ExploreAhead* runs, floorplans). The *static* directory contains the static implementation and each PR Module is implemented in the *pr_modules* directory. All bitstreams are generated to the *merge* directory.

6. Implementation

This section discsses both *EuTOPIA*'s implementation and the expected structure of the EDK-based architecture.

6.1. Development language

EuTOPIA is written in Java to take advantage of its portability to multiple platforms. Users are able to download *EUTOPIA* and run it locally regardless of the operating platforms (e.g. Windows, Linux, Mac). The additional options outlined in Sections 4 & 5 are being incorporated at present. The current size of *EuTOPIA* is less than 100KB, and these additions will not substantially increase the code space requirements.

6.1. Input architecture requirement

To detect the PR Regions and automatically modify the HDL files match the DPR design flow requirements, there is one architectural requirement for *EuTOPIA* to be able to modify an EDK-based DPR design. In particular, the PR Regions need to be connected to the Peripheral Local Bus (PLB) via PLB_Bridges and the naming of the PR Regions must follow a set of convention.

Currently, up to three PR Regions can be accommodated by *EuTOPIA*. The naming convention requires each region to be named reconfig_region_X where X is the PR Region number, starting with 0 (zero). The PLB Bridge is an 8-bit unidirectional communication channel between the PLB and a custom IP. Two PLB Bridges are required for duplex communication with a PR Region. A simple system highlighting the inclusion of these bridges for an EDK-based DPR system is shown in Figure 11.

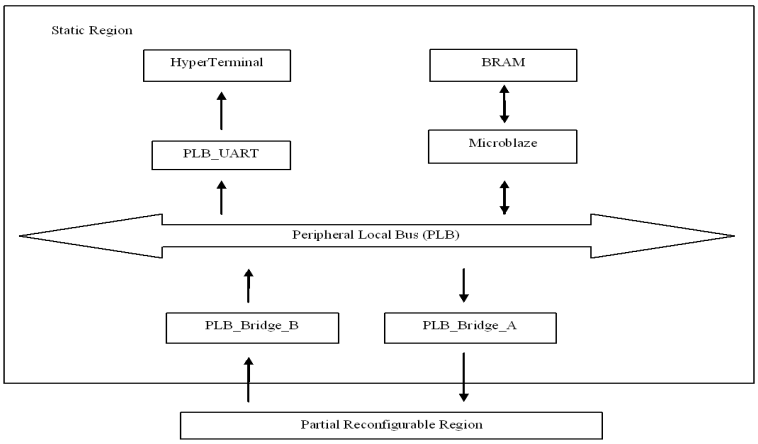

Figure 11. PLB reconfigurable module bridge in EDK

PLB_Bridge_A is responsible for communication directed from the PLB to the PR Region. PLB_Bridge_B is responsible for directed communication from the PR Region to the PLB. Currently all PR Regions must be connected to the PLB. Future versions of *EuTOPIA* will permit direct communication between the PR Regions and the soft-processor via Xilinx's Fast Simplex Link (FSL), a unidirectional First-In-First-Out (FIFO) communication bus. It should also be noted that *EuTOPIA* can generate a DPR system for designs that include embedded PowerPCs as opposed to MicroBlazes. However, the PR Regions still must be connected to the PLB. Investigations as to the possibility of connecting the PR Regions directly to the PowerPCs Local Bus (PLB) are currently underway.

7. Experimental procedures and results

To demonstrate the time savings that can be achieved by using *EuTOPIA*, a case study of an EDK-based DPR project is described in this section. The experimental procedures and results that are used to quantify the time-savings are presented.

7.1. Experimental system description

The same DPR system is implemented twice, once using *EuTOPIA* and the second time manually. The starting point for this experiment is to assume the existence of a non-DPR system implemented using EDK. As the purpose of *EuTOPIA* is to reduce the design time of DPR systems specifically, the time required to create the initial non-DPR system in EDK is not considered. Table 1 outlines the specifications for the implemented DPR system.

Table 1. Implemented DPR system specifications

Number of PR Regions	2
Processor	MicroBlaze
Number of PR Modules per PR Region	3 (Including a *blank* PR Module)
Number of PLB Bridges	(2 Bridges per PR Region) x (2 PR Regions) = 4
Number of Bits per Bridge	8 (One BM Width)
Location of both PR Regions	Bottom Left
Target Development Board	V2P Development Board
Processor Speed	100MHz

The data width of a single BM is 8 bits. Currently, one BM is used for the unidirectional communication between the PR Region and the PLB. Future versions of *EuTOPIA* will incorporate multiple BMs per directional connection to increase the data width between the PR Region and the PLB to at least 32 bits. The experimental system was created using EDK 9.2 SP2 and ISE 9.2 SP1 with the *Implementation Tool Add-on for Partial Reconfiguration* [12]. The output of *EuTOPIA* is generated for PlanAhead 9.2.2. The system is implemented using Windows XP Pro, running on a Pentium IV 2.4 GHz with 2GB of memory. The high level block diagram of the implemented system is shown in Figure 12.

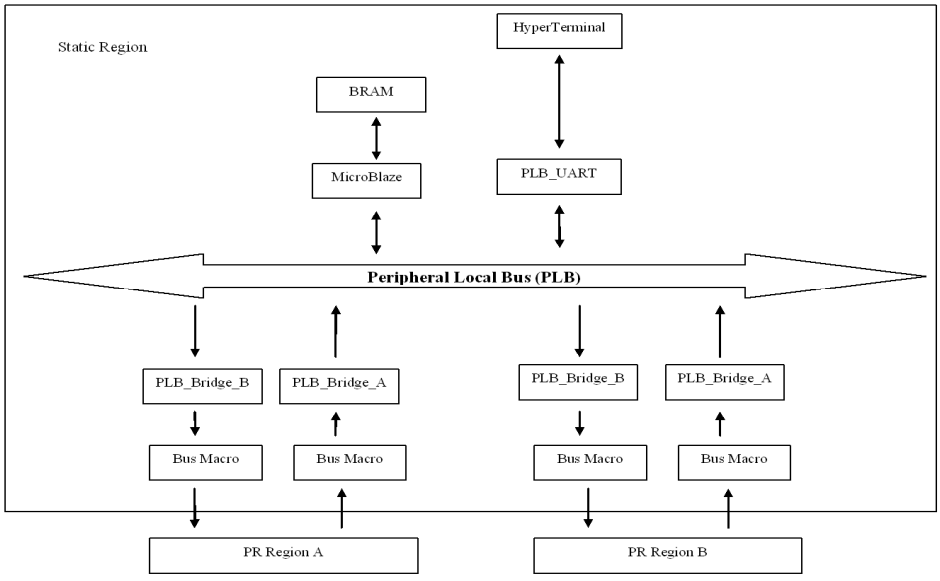

Figure 12. High level block diagram of the implemented system

Data and instructions are relayed to and from the PR Regions through the PLB_Bridges. The information is then relayed to the soft processor (MicroBlaze) through the PLB. Information received from the PR Regions is displayed serially through the UART onto the HyperTerminal. There are two active PR Modules for PR Region 1: PR Module 1 continuously outputs *"1234"* and PR Module 2 continuously outputs *"ABCD"*. Similarly, PR Region 2 has two PR Modules, however, their output strings are "*5678"* and *"EFGH"* respectively. Therefore, including the "blanking" PR modules for both PR Regions, there are a total of 2^3=8 possible combinations of PR Modules at runtime. Both the automated and manual implementation methods generated final bitstreams that produced the expected behaviour, dynamically switching the PR Modules running in the PR Regions in real time.

7.1. Experimental result

To implement the DPR system manually without the aid of *EuTOPIA* is a considerable challenge. Not only does the designer have to understand the intricate details of the DPR flow, but also the detailed usage of ISE and PlanAhead. In contrast, by using *EuTOPIA*, only nominal knowledge of PlanAhead and the DPR design flow are required to generate the initial DPR system. No prior knowledge of ISE is required. The *EuTOPIA*-generated PlanAhead project can also be further modified using all of PlanAhead's and capabilities as required. Table 3 summarizes the estimated initial learning time required to understand the DPR design flow and the usage of the ISE and PlanAhead tools.

Table 3. Approximate initial learning time to implement a DPR system

Task	Approximate Learning Time (hrs)
DPR Design Flow and Methodology	40
Understanding of the modifications using ISE	15
Understanding of PlanAhead for DPR systems	15

Table 4. Approximate implementation time for the experimental DPR system

Task	Manual Implementation (Experienced User) Approx Time (hrs)	*EuTOPIA*
Create Initial File Structure	0.25	<5 sec *EuTOPIA* Phase1
Modification of HDL files in EDK	2	<20 sec *EuTOPIA* Phase1
Create NGC Files using ISE according to DPR design flow	1	~5 minutes *EuTOPIA* Phase1
PlanAhead Project Implementation ● Netlists Import ● Floorplanning ● Design Rule Checks ● Implementation flow ● Bitstream generation	3	~15 minutes *EuTOPIA* Phase2

For designers utilizing *EuTOPIA*, only its input constraints and design structure need to be followed and understood to generate the initial DPR system. The test subjects were Master's students who had a basic knowledge of FPGAs and embedded systems design using EDK, but no previous experience with DPR. The test subjects needed 3 hours to understand the constraints and architectural changes required by *EuTOPIA* before generating their DPR systems. Compared to the 70 (40+15+15) hours needed to fully understand Xilinx's DPR design flow, this time saving is quite significant, even before the implementation of the DPR system. We then had an experienced user manually implement the necessary changes to generate this DPR system to minimize the design time. Table 4 summarizes both the experienced user's and *EuTOPIA*'s implementation times, demonstrating the reduced turnaround time using *EuTOPIA* even for experienced DPR designers.

By using *EuTOPIA*, the actual implementation time of this experimental DPR system has been reduced from 6 hours to 20 minutes for a factor of 18 times speed up. This reduction in design time can increase significantly for novices who lack DPR design experience. Most (>90%) of the

implementation time needed by *EuTOPIA* to generate this experimental system is for synthesis using Xilinx's CAD tools. This time is dictated by the design complexity and is identical for both implementation methods. If the synthesis time is excluded, the actual setup time needed by *EuTOPIA* (e.g. HDL modification, file structure creation) is less than 1 minute compared to the manual approach time of 2+ hours. For more complex DPR design, *EuTOPIA* will only experience a negligible increase in time as a result of having to parse larger HDL files. However, the setup time for manual implementations may increase dramatically due to this greater complexity in the HDL files, which impacts both design floorplanning and partitioning.

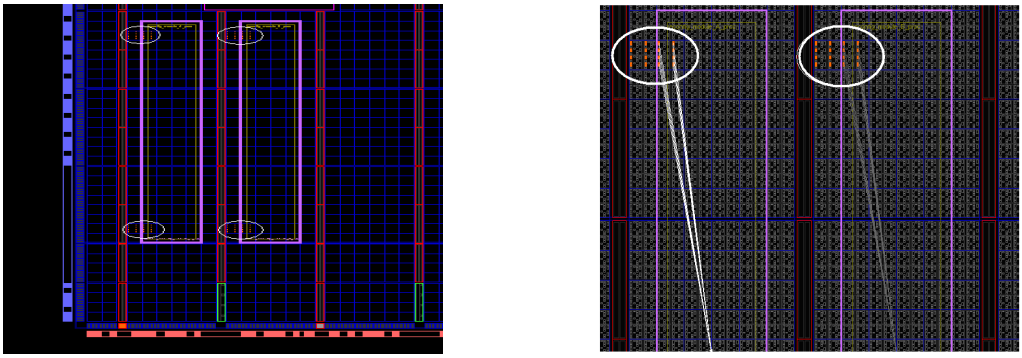

Figure 13. (a)PR regions implemented on the Virtex-2- pro board and (b) PR Regionscommunications using BMs

Figure 13(a) shows the two PR Regions (PRR_1 on the left, PRR_2 on the right) implemented in the bottom left position of the XC2VP30 FPGA on the Virtex-2 Pro development board. Duplex communication is achieved through the BMs (encircled in white) located at the top and bottom portion of each PR Region. The top BM for each PR Region is shown in Figure 13(b). As noted, two BMs are required to provide duplex communication between the PR Regions and static regions of the design. In this example, the top BM is the 8-bit input from the static region to the PR Region. The bottom BM (not shown) is the output from the PR Region to the static region. Future versions of *EuTOPIA* will allow for wider communication buses to and from the PR Regions.

9. Conclusion and future work

A Dynamically Partially Reconfigurable (DPR) system implementation allows designers to implement complete systems on devices with a smaller foot-print, that potentially improves power consumption, while enhancing FPGA fault tolerance. While it's many advantages, the implementation of a DPR system is very complex and time-consuming. Xilinx has introduced the PlanAhead tool that can be used to streamline the design flow for ISE-based DPR systems. However this is not sufficient for designers who have only nominal knowledge of the underlying DPR flow to quickly generate their processor-based DPR design in EDK.

This paper presented *EuTOPIA*, a software tool that facilitates the current design flow for DPR systems. Given a structured input architecture in EDK, this tool generates a PlanAhead project with full and partial bitstreams. Designers are then able to make further modifications and optimize their systems as desired. *EuTOPIA* provides designers a quick method to generate their systems with the core functionalities and flexibilities of a PlanAhead Project.

An identical experimental DPR system was implemented twice, once using *EuTOPIA* and the other manually. It was found that the implementation time for *EuTOPIA* was **18X** faster than manual implementation for experienced users. There is also an initial time saving of **70 hours** by using *EuTOPIA* from the initial learning stage of the DPR flow compared to three hours for *EuTOPIA*. It abstracts the low-level details of the DPR flow so that designers are not required to understand fully the intricate details to generate an initial system. They are only required to have nominal knowledge of the underlying process and the input constraints to *EuTOPIA*. Over the design-cycle of a product where

designers may generate tens or even hundreds of different implementations of their DPR design, these time savings would be significant.

Future versions of *EuTOPIA* will allow designers more flexibility in the options they can choose for their PDR system's design. Current upgrades being developed include: increasing the number of allowable PR Regions, user-specification of the desired location and size of each PR Region, increasing the data-width between the PLB and the PR Regions, supporting FSL and PLB connections to PR Regions via Bus Macros, user specification of the parameters of the Bus Macros, and the inclusion of additional development boards and FPGA device types.

10. References

[1] "Two Flows for Partial Reconfiguration: Module Based or Difference Based", Xilinx Application Note XAPP290 (V1.2), www.xilinx.com/bvdocs/appnotes/xapp290.pdf, Dec 8, 2007.

[2] P. Lysaght, B. Blodget, J. Mason, J. Young and B Bridgeford: Invited Paper: Enhanced Architecture, Design Methodologies and CAD Tools for Dynamic Reconfiguration for Xilinx FPGAs. FPL2006: 1-6

[3] S. Trimberger, D. Carberry, A. Johnson, and J. Wong, "A time-multiplexed FPGA. In Proc. 5th IEEE Symposium on Field-Programmable Custom Computing Machines 1997, p.22-28

[4] J. Becker, M. Huebner, and M. Ullmann, "Power estimation and power measurement of Xilinx Virtex FPGAs: trade-offs and limitations", 16th Symposium on Integrated circuits and Systems Design, SBCCI), 8-11 Sept 2003, pp.283-288

[5] K. Bondalapati, and V. K. Prasanna, "Reconfigurable computing systems, Proceedings of the IEEE Volume 90, Issue 7, July 2002, pp1201-1217.

[6] M. Hübner, J. Beaker : "Tutorial on Macro Design for Dynamic and Partially Reconfigurable Systems".

[7] N. Dorairaj, E. Shiflet, and M. Goosman. "PlanAhead Software as a Platform for partial Reconfiguration" Xilinx Xcell Journal Fourth Quarter 2005 pp68-71

[8] C. Kao. "Benefits of Partial Reconfiguration" Xilinx Xcell Journal Fourth Quarter 2005 pp65-68

[9] Xilinx XST User Guide 8.2i pp515-528

[10] Xilinx Early Access Partial Reconfiguration with PlanAhead 9.2 Users Guide

[11] A Zeineddini, "Secure Partial Reconfiguration of FPGAs " , M. Sc Thesis, George Mason University , Fairfax, VA, USA, 2005

[12] R.V. Kshirsagar, R.M. Patrikar : "Design of a Reconfigurable Multiprocessor Core for Higher Performance and Reliability of Embedded Systems", *IEEE proceedings of IFIP 14th International conference on Very Large Scale Integration*, VLSI-SoC 2006,pp 251-254, Oct. 16-18, 2006,Nice,France

[13] E. Chen, D. Sabaz, W. Gruver, and L. Shannon, "Facilitating Processor-based DPR Systems for non-DPR Experts", *IEEE Proceedings of the 16th International Symposium on Field-Programmable Custom Computing Machines,* April 14-15 2008, Palo Alto CA, pp 318-319