FPGA Design Framework for Dynamic Partial Reconfiguration

Chris Conger, Ross Hymel, Mike Rewak, Alan D. George, and Herman Lam
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida
Gainesville, Florida 32611
Email: {conger, hymel, rewak, george, lam}@chrec.org

Abstract

Recent advances in Xilinx's FPGA devices and design tools significantly improve the practicality of incorporating dynamic partial reconfiguration into high-performance embedded computing systems. By taking advantage of internal configuration access ports, Xilinx FPGAs are capable of in-situ partial reconfiguration without the need for external components, a technique defined as self-reconfiguration. However, proper planning and a significant amount of manual floorplanning are required to effectively leverage partial reconfiguration and truly enhance the capabilities of a system and/or achieve cost savings. This paper proposes and analyzes partial reconfiguration design methodologies to enable self-reconfiguration using Virtex-4 and Virtex-5 FPGAs.

1. Introduction

SRAM-based FPGAs are reprogrammable devices, with the ability to modify their hardware architecture easily and at any time. Partial reconfiguration (PR) enhances this paradigm by reconfiguring only a portion of the chip's configuration memory, allowing the user to load and unload user-defined hardware modules without interrupting or resetting the rest of the device. Reconfiguration time is decreased using PR, and bitstream communication and/or storage costs can be reduced. Configuration memory scrubbing (i.e. checking for errors and correcting them, if possible) is made possible by PR. Despite these advantages, commercial interest in PR has not significantly materialized due mainly to a lack of supporting software tools and merciless design flows. Nevertheless, the powerful potential of this level of reconfigurability has been recognized by the research community, and different conceptual approaches have been developed to incorporate PR into embedded systems using Xilinx FPGAs [1-9].

Earlier Virtex devices, up to and including the Virtex-II Pro family, as well as their PR design tools, are constrained by critical restrictions on PR designs. For example, entire columns of the FPGAs have to be reconfigured. Also, under older tool flows, static signals cannot easily pass through a partially reconfigurable region. Ultimately these constraints make PR design with these older FPGA devices and tools too impractical to be useful in a wide variety of applications. However, with the release of the Virtex-4 and Virtex-5 series of FPGAs and their tile-based frame architectures, PR design constraints are greatly relaxed. With the emergence of the lucrative software-defined radio (SDR) market, Xilinx took the initiative to engineer a simplified PR design flow [10] within their standard FPGA development environment. This new design flow eliminates or automates many of the burdensome requirements put in place by the previous method.

Unfortunately, due to the relatively recent unveiling of Xilinx's new PR design flow, it is still in an early-release phase and unavailable to the general public. As a result, there exists a vacuum in academic research and results exploring partially reconfigurable systems using the latest Virtex FPGAs. By combining our PR design experience with strategies and key issues identified in literature, we seek to define design methodologies and "best practices" for incorporating dynamic self-reconfiguration into a variety of reconfigurable computing (RC) systems based upon Virtex-4 and Virtex-5 devices.

2. Related Research

In their early PR design approach, Ullmann et al. proposed a partially reconfigurable architecture for automotive systems, based on Virtex-II FPGAs [1]. Their FPGA architecture includes a MicroBlaze soft-core processor, a static internal configuration access port (ICAP) controller, and four user-definable, partially reconfigurable regions (PRRs) that span the entire height of the device. In addition to describing the architecture of the PR device, they discuss run-time resource management, such as keeping track of PR module locations in memory and preservation of state information when swapping reconfigurable modules, both of which are important facets of dynamic self-reconfiguration. A follow-up paper [2] explores the same PR architecture in more detail, focusing on a novel slice-based "bus macro" component. Bus macros provide anchor points that allow signals to enter and leave PR regions at known locations. Original bus macros were based on tri-state elements (TBUFs), which, if malfunctioning or used incorrectly, could cause a short circuit within the FPGA and potentially destroy it. Xilinx has since eliminated TBUFs in Virtex-4 devices and beyond and instead has incorporated slice-based bus macros into the PR design flow.

Hubner et al. extended the previous work with the MicroBlaze-based PR architecture by proposing a technique to avoid the requirement of Virtex-II FPGAs to reconfigure entire columns [3]. Their technique was very involved, however, requiring the user to perform a read-modify-write process on configuration memory using Xilinx's JBits tool (as opposed to simply loading a new partial bitstream). While improving the flexibility of PR on the earlier chips, the main drawback to this approach is large reconfiguration times, on the order of seconds, which may be an intolerable reconfiguration delay in tomorrow's high-performance systems.

Wichman et al. proposed their own self-reconfiguration methodology [4] using Virtex-II Pro FPGAs. Their approach inspired our initial approach to PR design with Virtex-4 FPGAs [5], which is to statically floorplan the entire FPGA at design time, breaking the overall FPGA up into pre-defined PRRs. Wichman suggests that each module that is to be executed on the FPGA should be placed and routed for each region of the chip, so that multiple versions of each module exist (one for each PRR on the FPGA). They indicate that using external components to perform reconfiguration of an FPGA can create a single point of failure, and introduces the risk of losing the ability to reprogram the FPGA in the event of a failure of the reconfiguration controller. Because of this risk, a recommendation is made to use the ICAP as described in [1] for self-reconfiguration.

Research by Sedcole et al. also proposed a method to circumvent the full-column reconfiguration requirement of the earlier Virtex FPGAs [6]. Their technique addresses the difficulty in routing static lines through partially reconfigurable columns, however their solution requires an additional step beyond place-and-route (realized through the implementation of a custom tool) since, at the time of their work, there was no mechanism to specify the required constraints in Xilinx's design flow. They also used a read-modify-write approach to performing the actual reconfiguration, similar to that used in [3].

Partial reconfiguration can also be leveraged to improve the reliability and lifetime of FPGA devices, as described by Emmert et al. [7]. They proposed a mechanism called Self-Testing AReas (STARs), a process that runs in the background and scans the FPGA in small square areas looking for permanent faults. Upon discovery of a faulty area of the chip, their mechanism reconfigures the design to simply "avoid" the bad section. They extended this research in [8] by considering enhanced capabilities such as partial use of a faulty region, as well as optimizing spare region allocation so as to minimize wasted resources and impact on system performance during normal operation. Other research discusses the use of PR for system-level fault tolerance on a more conceptual level, where the PR design flow is analyzed in detail and key challenges and design tradeoffs are highlighted [9]. An important observation is the need for and value of automated design tools, which help insulate the designer from the low-level details of a PR design and optimize the performance and efficiency of partially reconfigurable FPGA architectures.

3. Understanding the PR Design Space

Generally speaking, FPGAs are used to implement a wide variety of hardware architectures. Before proposing or suggesting any particular PR design strategy, it is important to consider the *type* of system or platform that is being designed. Both the device-level and system-level architecture of a reconfigurable computing (RC) platform will directly impact the connectivity of PR-specific mechanisms with other modules or system components. Another issue to consider is whether or not every potential PR module is defined prior to system implementation. If every application, or version of a specific application, that will be executed on a particular system is known at design time, then engineers can take advantage of this information to help optimize interfaces as well as determine resource-efficient FPGA floorplans. However, it is often conceivable that an FPGA system or board may be designed with the intention to re-use it for multiple missions in the future, or to upgrade components as improved or updated versions emerge. To clarify these concepts, we have defined three system classes as well as two different system design scenarios, which describe and refer to common use-cases of FPGAs in contemporary systems.

The three defined system classes include: (a) System-on-Chip (SoC), (b) coprocessor, and (c) stand-alone, as illustrated in Figure 1. FPGAs-based designs for SoC are characterized by the presence and use of

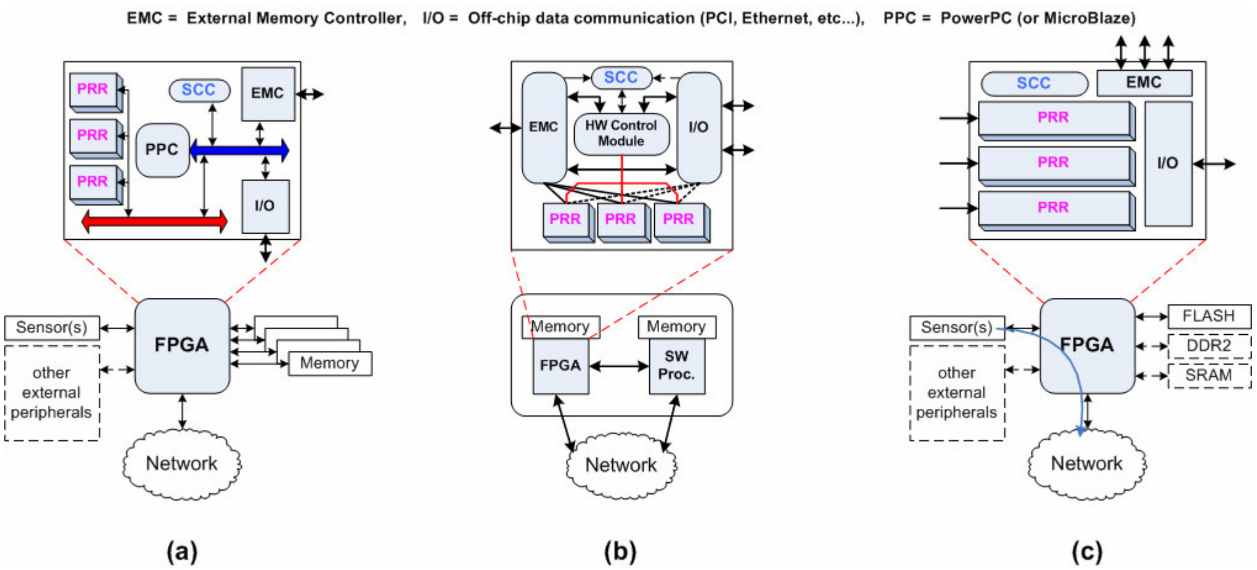
EMC = External Memory Controller, I/O = Off-chip data communication (PCI, Ethernet, etc...), PPC = PowerPC (or MicroBlaze)
PRR
SCC
EMC
PPC
I/O
FPGA
Sensor(s)
other external peripherals
Memory
Network
(a)
HW Control Module
SW Proc.
(b)
FLASH
DDR2
SRAM
(c)

Figure 1. Example illustrations of common FPGA use-cases: (a) System-on-Chip, (b) coprocessor, and (c) stand-alone.

on-chip, hard-core PowerPC processors or soft-core MicroBlaze processors, as well as an on-chip bus interconnect such as IBM's CoreConnect for interfacing various on- and off-chip peripherals with each other and the software processor(s).

FPGAs used as coprocessors are characterized by an association or pairing with some external software processor, where the software processor serves as a master and the FPGA coprocessor serves as a worker, typically a high-performance computational offload engine (or alternately, perhaps as an intelligent network controller). The internal architecture of a coprocessor-class FPGA can be more customized than the SoC-class FPGA, and typically the internal control logic is relatively simple. Good examples of coprocessor-class FPGA systems are PCI-based FPGA cards that fit into slots on a typical workstation motherboard, or high-performance computing (HPC) systems such as XtremeData's XD1000 platform, which pairs an AMD Opteron with an FPGA by housing the FPGA in one of the two HyperTransport-connected Opteron sockets of a dual-processor motherboard.

SoC- and coprocessor-class systems both have the advantage of being able to rely upon software to help manage and control the FPGA. The third system class, stand-alone FPGAs, is characterized by not being directly associated with or controlled by a software processor. An example of such a system might be front-end preprocessors in a streaming, real-time system. One or more FPGAs may be used to interface directly with sensors, where they perform line-rate pre-processing and/or framing of raw data before sending it over the network to the main processing system. These stand-alone FPGAs have the most customized internal architecture of all three system classes, and thus are the most difficult to generalize for specifying standardized components and design flows.

An omnipresent component in any FPGA design that intends to support self-reconfiguration is the static configuration controller (SCC). The SCC is mainly responsible for performing the reconfiguration of PRRs; however its responsibilities may include much more, depending upon the system class in which it is used. In SoC or coprocessor systems, the SCC design can be minimal, with only a basic state machine to control the ICAP and interface with data FIFOs for buffering configuration bitstreams, as well as an optional decompression and/or decryption unit [1]. However, in a stand-alone FPGA, the SCC may be required to serve additional roles, such as tracking of currently loaded modules or interfacing directly with other PRRs to enable dynamic, data-driven self-reconfiguration, thus increasing the resource overhead of the SCC component for these systems.

One can divide system-design scenarios into two categories: (1) special-purpose system design and (2) multipurpose system design. Any of the previously defined system classes could be designed for either of these two scenarios. Engineers designing special-purpose systems have the advantage of knowing at design time every module that will be executed on the FPGAs, and thus special approaches can be employed in designing the partially reconfigurable system in order to optimize the physical mapping of static and reconfigurable modules according to user-defined rules, such as minimizing wasted resources, minimizing partial bitstream sizes, or maximizing clock frequency. Multipurpose systems, in contrast, must allocate PRRs with fixed physical dimensions and locations, without knowing in advance how some application designers in the future might want to use the system. Interfaces between these PRRs and the static region of the design must also be fixed at design-time, further constraining the capability of the system to support arbitrary applications in the future. Because of this uncertainty, the PR design strategy for multipurpose systems should be focused on maximizing flexibility and promoting design reuse.

In addition to defining system classes and recognizing different design scenarios, it is important to define the basic metrics that we will use to measure and compare the quality of PR designs. Maximum achievable clock frequency is an important design metric to many engineers, and the manual floorplanning and other PR-specific overheads can affect the clock frequency relative to what is achievable by following a standard (non-PR) FPGA design flow. Another important metric is bitstream size. One established benefit of partial reconfiguration is the reduction of bitstream sizes, which decreases reconfiguration latency at run-time as well as reduces the amount of data that must be communicated over a network or read from memory. Bitstream compression can be used as well to further reduce the size of partial bitstreams. An analysis presented in Section 4.1 suggests that the physical geometry of a given module can significantly affect the corresponding partial bitstream size, so it may be possible to optimize a design so as to minimize total storage requirements. Finally, resource efficiency can become an issue when the designer is heavily involved in floorplanning of an FPGA (such as with PR design). Allocating PRRs of fixed sizes, which fit within the configuration frame granularity of the target device and provide sufficient resources to each of the modules that will share a given PRR, can be challenging to achieve while minimizing the amount of wasted resources.

4. PR Design Framework

The class of system being designed has more of an impact on the modules and interfaces present in a PR design, as well as the responsibilities of the SCC. However, the actual PR design flow should be tailored to the design scenario, regardless of the targeted system class, as described in this section.

4.1 Special-purpose Systems

From a designer's point of view, special-purpose FPGA systems have the distinct advantage of containing all the information that is needed to create a tailored, highly optimized design implementation before system deployment. For special-purpose systems, all PR modules that will exist throughout the life of the system are known at design-time, as are the transitions that define which PR modules exist for each context. As such, the design methodology for special-purpose systems can be divided into three main stages: region partitioning, overlay generation, and implementation with timing verification. An illustration of this process is shown in Figure 2.

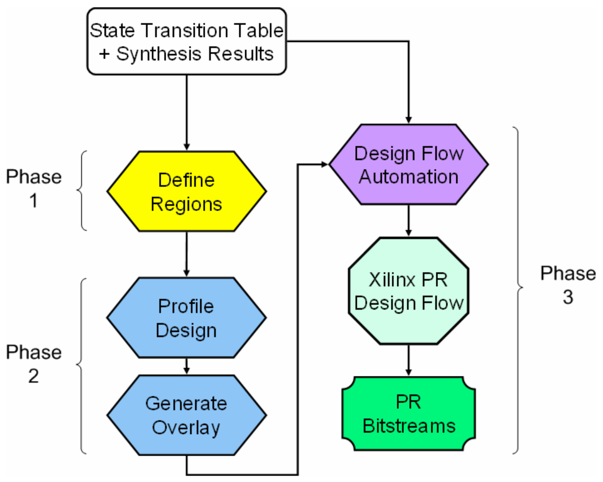

Figure 2. PR flow for special-purpose systems.

After the system as a whole has been architected, the first stage in special-purpose designs that differentiate them from normal, non-PR designs is region partitioning. One of the critical advantages of PR is that time-independent tasks can multiplex the hardware resources of a single device. For non-PR designs, all these time-independent tasks would coexist on the FPGA fabric, with idle tasks simply wasting resources. PR prevents those resources from being wasted by loading and unloading the hardware modules from shared sections of the FPGA fabric. Doing so can result in a reduced device count and/or a smaller FPGA. However, in order for this hardware-multiplexing to take place, some form of mapping must exist that specifies which device resources are to be shared by which PR modules.

In general, a special-purpose system will be characterized by a state-transition table. The entries within this table define the macro states that the system transitions through during all phases of operation as well as the modules that exist within each specific state. By locating sets of modules that do not exist at the same time over all states, the designer can allocate those modules to specific shared PR regions. Whether automated or performed by hand, this region partitioning requires heuristic solutions with many optimization goals available. These goals can include minimizing the number of PR regions, which would minimize the run-time of the Xilinx implementation tools, minimizing the total size of the regions to save resources, minimizing the number of I/O nets per region to prevent bus macro overhead, and many more.

After region partitioning, the designer will have obtained a set of PR regions, each of which containing a set of PR modules. In the next stage, overlay generation, these PR regions are mapped to specific locations within the FPGA fabric, in effect creating an overlay with holes or sockets for each PR region. To ensure that each module within a specific PR region can "plug" into its socket, wrapper files must be generated that ensure the top-level declaration of each module has the same port-level interface as every other module in that same region. Next, by synthesizing each individual module and comparing the resource requirements for each module within a specific region, the designer can determine the overall resource needs of each PR region. This resource knowledge is a critical advantage in special-purpose systems, as each PR socket in the overlay can be sized, shaped, and positioned to meet known needs.

At this point, the designer is once again faced with a problem requiring a heuristic solution: how to best map these PR regions to physical locations within the FPGA. One option is to use generic, pre-existing overlays such as those mentioned in the related research section, but doing so defeats the principle of having a special-purpose system. A better solution would be to algorithmically generate an overlay to match the needs of the specific design and then estimate the quality of this overlay by using a weighted sum of costs. Cost functions could include aspect ratio, amount of wasted resources, position relative to needed IOBs, routability, and others. The overlay could then be modified and rescored for a number of iterations until an acceptable solution is found. Alternatively, the designer could craft a custom overlay by hand, but we have found this method to be a tedious and error-prone operation with very inconsistent results [5].

We are currently trying to quantitatively analyze these cost functions so that an automated solution to this stage can be developed in the future. Initially, we have analyzed the performance effects (e.g. clock frequency, partial bitstream size) of varying the size and shape of PR regions containing different classes of PR modules, such as slice-intensive, memory-intensive, DSP-intensive and other hybrid types. Table 1 shows the different cores used as test PR modules, as well as their resource requirements when targeting an XC4VSX55 device. These cores include constant false-alarm rate detection (CFAR), beamforming (BEAM), an ARM7 soft-core processor (ARM7), advanced encryption standard (AES), and a simple sine/cosine look-up table (LUT).

Figure 3 plots the performance results of varying the aspect ratio, defined as the height in slices divided by the width in slices, of a PR region that contains an individual test core (two selected charts are offered of the five total cores that we have studied). These results are also compared to an unconstrained baseline and summarized in Tables 2 and 3. The baseline is defined as the performance of the core when it is not forced into the fixed geometry of a PR region but, instead, is allowed to be placed and routed freely.

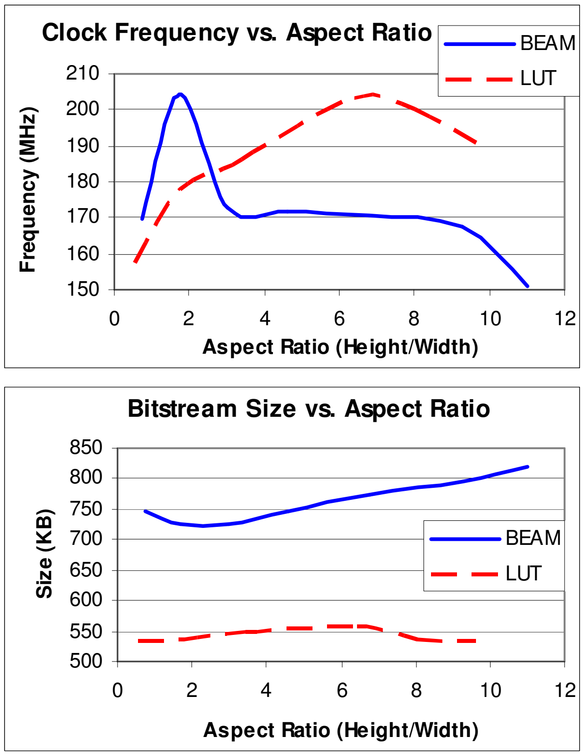

Figure 3. PR metrics vs. aspect ratio.

Table 1. *Resource requirements on XC4VSX55 device.*

	LUTs	FFs	BRAMs	DSPs
CFAR	4130	3625	34	2
BEAM	6484	5732	17	16
ARM7	3348	942	16	10
AES	3855	3943	4	0
LUT	199	60	27	0

Table 2. *Clock frequency variation (MHz).*

	Baseline frequency	Frequency range
CFAR	103.6	104.7 – 118.8
BEAM	127.8	151.2 – 204.2
ARM7	40.9	38.9 – 41.7
AES	80.5	75.9 – 86.7
LUT	204.9	157.2 – 203.1

Table 3. *Compressed bitstream size variation (KB).*

	Baseline size	Size range
CFAR	1,001	690 – 773
BEAM	1,614	726 – 819
ARM7	872	484 – 515
AES	1,393	639 – 677
LUT	571	726 – 819

Our results reveal that slice-intensive designs show best results for both clock frequency and bitstream size with an aspect ratio between about 2 and 4. One explanation for this result is that these aspect ratios are in the same ballpark as the aspect ratio of the device as a whole (2.66). Similarly, non-slice-intensive designs show best performance with aspect ratios >> 4. Resource wastage is minimized at these aspect ratios as well. This result is almost certainly due to the columnar distribution of RAMB16/DSP48 resources within the device layout. Another important result that we have observed is that the effect of aspect ratio on performance is more pronounced for cores with higher maximum operating clock frequencies.

Based upon these results, we are presently designing overlay optimization algorithms that can profile the individual PR regions in a special-purpose design and then help determine an efficient overlay that is tailored to the specific resource needs of the target system. This heuristic approach should eliminate the guesswork that occurs during this phase of a special-purpose design and provide improved design performance when compared to manual overlay generation, in addition to decreasing the amount of time spent in this stage of the special-purpose design flow.

After completing the overlay generation stage, the designer will have a useable overlay to accompany the module mappings. The last step in a special-purpose design is to generate the partial bitstreams that represent each of the PR modules as well as full bitstreams to represent each of the states within the state-transition table. The full bitstreams are necessary to ensure that each possible combination of PR modules satisfies the timing constraints of the system. A top-level wrapper is first generated to tie the design together, with black-box instantiations of each individual PR region as well as the static section of the design and bus macros for communication. Also, the SCC is updated to reflect the module mappings and the state-transition table and then inserted into the static portion of the design. At this point, the special-purpose design is completed from the user's perspective and the Xilinx software implementation tools are invoked.

4.2 Multipurpose Systems

Engineers developing a new multipurpose FPGA platform do not have the advantage of knowing in advance all of the modules that will be configured on that architecture. Possibly because the platform being designed is intended to be a flexible product, or perhaps it is impossible to fully anticipate future upgrades that will be required to avoid obsolescence or support new missions. We propose a design flow for multipurpose FPGA systems, illustrated in Figure 4, which promotes design reusability as well as the potential elimination of repeated qualification efforts. An important feature of multipurpose system design is the decomposition of the overall flow into two phases, one for the base design (i.e. static portion), and one for each partially reconfigurable module that is to be used within that base design.

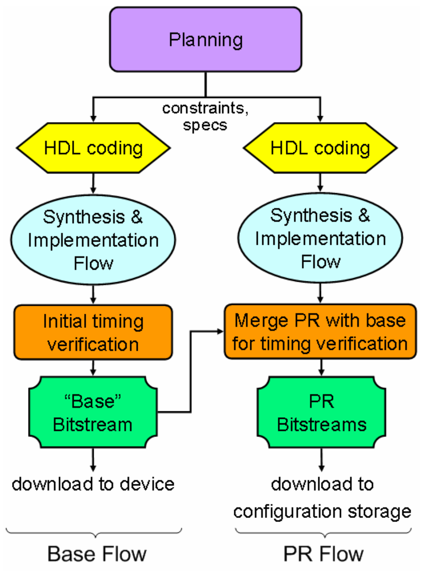
Planning
constraints, specs
HDL coding
HDL coding
Synthesis & Implementation Flow
Synthesis & Implementation Flow
Initial timing verification
Merge PR with base for timing verification
"Base" Bitstream
PR Bitstreams
download to device
download to configuration storage
Base Flow
PR Flow

Figure 4. PR flow for multipurpose systems.

The flexibility, efficiency, and reusability of a multipurpose PR system hinges largely on the degree of planning performed, and the level of anticipation of future designs that is possible at the beginning of the design cycle. The ultimate goal of the planning step is to decide on an overlay specification, which describes the shape and location of each PRR in the design, as well as an interface specification or template for each PRR. Two key issues that have to be considered during this "planning" step are: (1) PRR shaping and placement, and (2) interface definition for connecting each PRR to the remainder of the design. By defining a certain shape and size for a given PRR, a system designer is effectively fixing the number of resources available to the users for designing their own PR modules. Also, since signals that will cross between regions must be fixed in the base design and connected to bus macros, the interfaces on the PRRs cannot change. However, there are instances depending upon the system class where these issues may not pose a significant challenge. For example, in an SoC architecture, the interface issue might not be a concern since the PRRs will likely be attached to the CoreConnect bus of the PowerPC or MicroBlaze subsystem. The CoreConnect bus interfaces are already standardized, so fixing the interfaces of the PRRs at design-time should not pose any problem.

Once an overlay has been determined, the base flow only needs to be performed once to generate the base bitstream. The only difference between the base flow and a non-PR design flow is the specific way of partitioning the top-level design [10], and the presence of bus macros and the SCC. The constraints specified in the planning stage are used to lock down the placement of PRRs and bus macros in the .ucf file, and the design is then run through the normal Xilinx software tool flow to generate the base bitstream. The base bitstream can be downloaded to the device at any time, but it must also be archived for use when designing the PR modules for this system.

When an application designer wants to implement a new application on this platform (or upgrade an existing one), they only need to go through the second phase of the design flow and are completely insulated from the full, top-level design. An interface specification or template should be available for each PRR in the design, which can be used as the "top level" of the partially reconfigurable component. This isolation from the remainder of the design reduces the execution times in the synthesis and place & route tools, which otherwise can be exceedingly high during the iterative development and testing stage of large chip designs. However, it should be noted that after a new PR module has been designed, it still must be combined with the base bitstream by the design tools for timing verification of the entire design.

To help illustrate the challenge of specifying efficient and flexible PR designs, we have proposed three generic example FPGA floorplans for a Virtex-4 LX25 device, which decompose the FPGA into one, two, or four PRRs. Figure 5 shows each of the three floorplans (note: the yellow regions represent bus macro placement), and Table 4 quantifies the resource overhead of the PR-specific components of each architecture.

In these example architectures, only the SCC and bus macros exist in the static portion of the design. It should be noted that many systems may have a significant amount of user-defined logic located in the static portion, such as memory or network interface logic. The configuration frame resolution of Virtex-4 devices is 16 configurable logic blocks (CLB) of a single column, where each CLB consists of four slices arranged in a 2×2 array. Given the 96-CLB height of the XC4VLX25 device, the frame resolution puts a significant restriction on the minimum height of any region of the device. In each of the three example layouts, the SCC region represents the smallest possible static region (one configuration frame in height). If no logic beyond the requisite SCC and bus macros is present in the static region of the design, then the SCC resource utilization numbers represent the worst-case SCC overhead. However, in designs where user-defined logic shares the static region with the SCC, more efficient use is made of the resources in the static region, thereby reducing overhead. The actual overhead in that case would be the difference between the total static resources and the resources used by non-SCC logic. The number of bus macros used to connect the various regions was also selected to try and mimic a worst-case overhead, by providing as many signal paths between modules as possible.

Table 4. *PR-specific overheads of floorplans.*

Arch. (# PRRs)	Bus Macros (in slices)	SCC (in slices)	Total % of V4-LX25
One	522	1910	22.7 %
Two	792	1664	22.9 %
Four	1508	1784	30.6 %

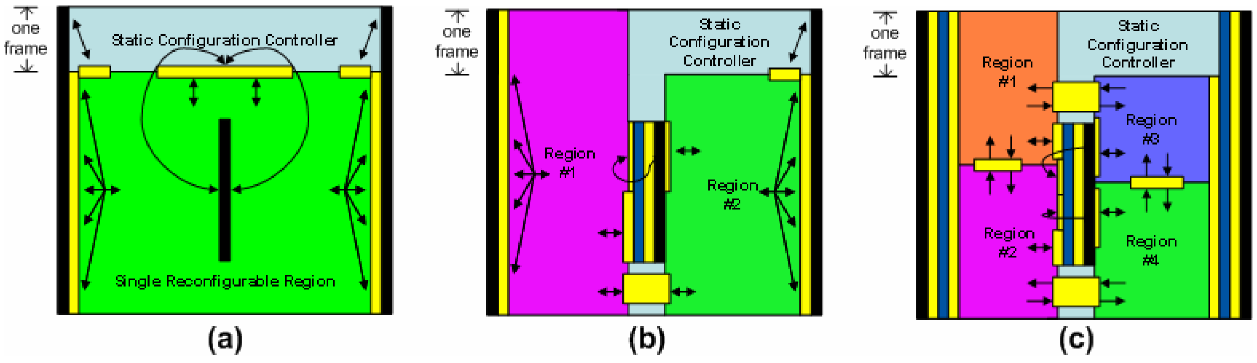
one frame
Static Configuration Controller
Single Reconfigurable Region
(a)
one frame
Static Configuration Controller
Region #1
Region #2
(b)
one frame
Static Configuration Controller
Region #1
Region #3
Region #2
Region #4
(c)

Figure 5. Example floorplans on XC4VLX25 device, with (a) one, (b) two, or (c) four PRRs.

5 Conclusions

Partial, self-reconfiguration is a powerful and increasingly practical capability of Xilinx FPGAs. However, the full potential of this level of reconfigurability has yet to be harnessed due to the relative youth of the technology and supporting design tools and a host of challenges. PR can enable the time-multiplexing of hardware resources on an FPGA and increase the adaptability of the system, allowing smaller devices to provide the same functionality as larger, statically configured devices. PR can also be leveraged to enhance the fault-tolerance capabilities of a system through configuration scrubbing, fault-isolation, and adaptive protection. The newest Xilinx devices, in both the Virtex-4 and Virtex-5 families, have important architectural differences from previous Virtex device families that make them much more amenable to partial reconfiguration. However, along with these enhanced capabilities comes a whole new design optimization problem, a problem with solutions that differ depending on the applicable system design scenario.

We have presented an organized view of the common use-cases of FPGAs in today's computing platforms, as well as proposed optimized PR design methodologies that are tailored to the purpose of the system being designed. Our work contributes much-needed research exploration of PR design with the latest FPGA devices and software tools, and provides insight into the new capabilities and key challenges of these new technologies. We have also extended our investigation of PR to include a wider range of system classes and applications.

However, there is still much work to be done. Based upon the performance impact that we have observed of varying PRR geometries, we are currently working on designing and analyzing virtual overlay optimization algorithms. There is a strong need to ease the PR design process with automated tools, and this overlay generation and optimization algorithm could be leveraged to create new methods for a CAD tool to assist in the design of special-purpose systems. Multipurpose systems present an interesting challenge to maximize the ability of a fixed architecture to support arbitrary modules, due to the need for interface standardization and fixed allocation of resources at design-time. Additional research and collaboration is needed to propose more effective methods of optimizing these multipurpose system designs. Lastly, PR could be leveraged to provide reconfigurable fault tolerance, allowing a common architecture to support multiple levels of fault tolerance, depending upon mission requirements or environmental conditions.

6 Acknowledgements

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge the software tools donated by Xilinx that helped make this work possible.

7 References

[1] M. Ullmann, M. Huebner, B. Grimm, and J. Becker, "An FPGA Run-Time System for Dynamical On-Demand Reconfiguration," *Proc. of the 18th International Parallel and Distributed Processing Symposium*, Apr. 26-30, 2004.

[2] M. Huebner, T. Becker, and J. Becker, "Real-Time LUT-Based Network Topologies for Dynamic and Partial FPGA Self-Reconfiguration," *Proc. of 17th Symposium on Integrated Circuit and Systems Design*, Sep. 7-11, 2004, pp. 28-32.

[3] M. Huebner, C. Schuck, M. Kuhnle, and J. Becker, "New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-time Adaptive Microelectronic Circuits," *Proc. of 2006 Emerging VLSI Technologies and Architectures*, Mar. 2-3, 2006.

[4] S. Wichman, S. Adyha, S. Ahrens, R. Ambli, B. Alcorn, D. Connors, and D. Fay, "Partial Reconfiguration Across FPGAs," *Proc. of Military and Aerospace Applications of Programmable Logic Devices and Technologies Conference*, Sep. 26-28, 2006.

[5] R. Hymel, A. George, and H. Lam, "Evaluating Partial Reconfiguration for Embedded FPGA Applications," *Proc. of 11th Annual High-Performance Embedded Computing Workshop*, Lexington, MA, Sep. 19-21, 2007.

[6] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, and T. Becker, "Modular Partial Reconfiguration in Virtex FPGAs," *Proc. of 2005 International Conference on Field Programmable Logic and Applications*, Aug. 24-26, 2005, pp. 211-216.

[7] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic Fault Tolerance in FPGAs via Partial Reconfiguration," *Proc. of IEEE Symposium on Field Programmable Custom Computing Machines*, Apr. 17-19, 2000, pp. 165-174.

[8] J. Emmert, C. Stroud, and M. Abramovici, "Online Fault Tolerance for FPGA Logic Blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 15, No. 2, Feb., 2007, pp. 216-226.

[9] W. Zheng, N. Marzwell, and S. Chau, "In-System Partial Run-Time Reconfiguration for Fault Recovery Applications on Spacecrafts," *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4, Oct. 10-12, 2005, pp. 3952-3957.

[10] Xilinx Inc., "Early Access Partial Reconfiguration User Guide," UG208 (v1.1), Mar. 6, 2006.