

# Leveraging Firmware in Multichip Systems to Maximize FPGA Resources: An Application of Self-Partial Reconfiguration

Juan Galindo and Eric Peskin  
Rochester Institute of Technology  
79 Lomb Memorial Drive  
Rochester, NY, 14623, USA  
{jmg1931,erpeee}@rit.edu

Brad Larson and Gene Roylance  
Hewlett-Packard  
11311 Chinden Boulevard  
Boise, ID, 83714, USA

## Abstract

*A number of SRAM-based field programmable gate arrays (FPGAs) allow for partial reconfiguration (PR). Partial reconfiguration can be used to maximize the resource utilization in these FPGAs. Current methodologies use both external and self partial reconfiguration for this purpose. On mature multichip (MC) systems that have not made use of the PR features of their SRAM-based FPGA(s), however, these methodologies would require changes in the existing FPGA configuration protocol and/or associated hardware outside the array. This paper presents a novel methodology that makes PR features available to these systems for the purpose of maximizing their FPGA resources without the modifications required by the current methodologies. The proposed methodology reuses an existing data interface to send the PR data to the array and directs this data to the FPGA's internal configuration port. A prototype of this methodology is demonstrated on a commercial color space conversion (CSC) engine design using two Xilinx Virtex-II Pro FPGAs.*

## 1. Introduction

*Partial reconfiguration (PR) [16] is a unique feature of SRAM-based FPGAs that allows the reconfiguration of a part of the array while the rest of the FPGA continues operating. There are two types of partial reconfiguration: self and external. In self PR, the configuration data may be generated by the FPGA itself or provided by an external source. In external PR, the configuration data is stored externally and external logic that sends this data to the array is required.*

Provided that there are some mutually exclusive functional blocks within an FPGA design, external or self-PR can be used to make room for the additional hardware fea-

tures required. Delahaye et al. have presented a platform for performing external PR [6]. This platform uses a DSP and a dedicated interface to load the initial configuration in the FPGA and to partially reconfigure it as needed.

Hubner et al. describe a self-PR platform for Network-On-Chip applications [10]. In this platform, the FPGA requires access to the flash memory where the configuration data is stored. Bomel et al. have proposed a self-PR architecture in which the FPGA obtains the configuration data via an Ethernet connection to an external server [3]. Note that this connection could potentially be used to send processing data to the FPGA. However, existing self-PR methods [1, 2, 4, 8] including those described above assume that the FPGA can obtain the configuration data via one of the following interfaces: RS232, IrDA, 10/100 Ethernet, or flash memory. Self-PR methods that generate the configuration data inside the FPGA are not considered here because they are mainly used for fault tolerant applications.

In addition, PR platforms have traditionally been first devised and then applied to designs matching the requirements of these architectures. The lack of commercial products that use partial reconfiguration, however, shows that this approach ought to change. Active research in the area of partial reconfiguration is addressing this issue by delivering PR methods tailored to specific applications rather than generic PR architectures [14, 15].

In Section 2, a module-based PR methodology for mature MC designs is described. It makes use of the Xilinx PR flow [7] and of the well-defined communication channels for both data and control between the FPGA and the integrated circuits outside the array. A demonstration of the module-based methodology is presented in Section 3, where a prototype is designed using the methodology discussed. This section also presents background information pertaining to the prototype application as well as the design, implementation, and testing of the prototype using two Xilinx Virtex-II Pro FPGAs. Section 4 presents the results of



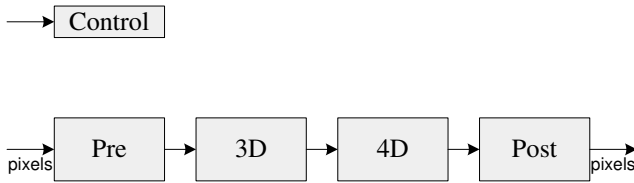


Figure 2. Core of the CSC engine.

stage pipeline and SRAM memory that function primarily as interpolation blocks and CLUTs respectively. To define these and other functional blocks within this engine, pipeline stages and SRAM memory are grouped into phases. Each phase contains up to three pipeline stages. These phases, however, are never all active simultaneously. In particular, one of the two largest phases is always bypassed. These phases are called the 3D and the 4D. The 3D phase is used when the input space has three channels. The 4D phase is used when the input space has four channels. Each of these two phases consists of three pipeline stages and over 40KB of SRAM. Figure 2 illustrates the core of the CSC engine.

In addition, the top-level IO of the CSC engine includes a pixel and a register interface through which pixel color data formats and CLUT values are sent to this engine. Color space conversion along with pixel scaling and halftoning are the main elements of any printer's image processing pipeline. Currently, HP implements this pipeline on a single *application specific integrated circuit* (ASIC). However, an HP printer is comprised of at least this ASIC, a memory subsystem, and an embedded processor. The latter components are employed to send data to the CSC engine and to implement the printer's user interface logic. This fact makes HP printers MC systems.

### 3.2. Prototype System

The CSC engine is implemented as part of an ASIC in commercial HP printers. This engine is synthesized for an FPGA in the prototype system to use it as an application example for the proposed methodology. Due to this difference in target devices, a new PCB board that accommodates the FPGA package implementing the ASIC functionality would be required to use the actual components that drive the CSC engine. Instead, these components are replaced by very similar off-the-shelf ones. This is a key aspect of the prototype system. Admittedly, the successful modification of the firmware run by the actual components of a printer to send the PR data to the FPGA-based CSC engine prototype would have truly demonstrated the interface timesharing technique. Nonetheless, the successful modification of the firmware run by similar hardware is also a valid demonstration.

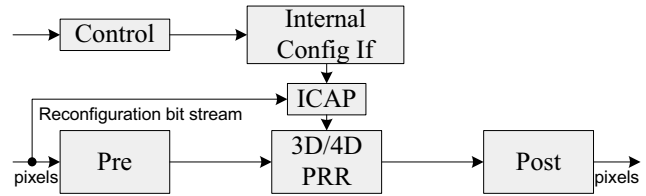


Figure 3. Modified CSC engine using PR.

The interconnects attached to the CSC engine's register and pixel interface are used by the modified firmware above to initiate the PR process and to transfer the PR configuration data respectively. The augmented control logic designed for this prototype consists of a new register that manages the PR process inside the FPGA and that siphons PR data off of the pixel interface as required. The internal configuration interface IP, which is connected to this register and to the pixel interface, drives the *internal configuration access port* (ICAP) of the Xilinx FPGA used in the prototype. Finally, additional logic is employed to stall the CSC engine pipeline while partial reconfiguration is being performed by the ICI block.

### 3.3. Prototype Implementation

Two development boards from the Xilinx University Program (XUP) have been employed to implement the prototype system described above. One of these boards, called *the CSC board* from now on, implements a modified version of the original HP CSC engine. The other board, called *the TestRig board*, emulates the components that drive the CSC engine in an HP printer. A large number of FPGA *general-purpose input and output* (GPIO) lines are available in these boards through standard IDE headers. These lines are used to transfer data from the TestRig board to the CSC board. Note that the Virtex II-Pro FPGAs on these boards and both a *compact flash* (CF) card and *double data rate* (DDR) memory on the TestRig board are the only XUP board components used in the prototype system.

In the prototype system, the TestRig board's embedded processor generates the test vectors required to stimulate all the inputs to the CSC engine. This 32-bit processor limits the width of the digital connection between the TestRig and the CSC board to 32 bits. Since the CSC engine has more than 32 inputs, *CSC packets* that stimulate all the CSC inputs in a cycle-by-cycle basis are created in the TestRig board. In the CSC board, the data in these packets needs to be depacketized and sent to the CSC engine. Therefore, an additional module that wraps the original CSC engine interface and that depacketizes the incoming data from the TestRig board is required in the FPGA-based CSC engine implementation prototype. A commercial version of such an implementation would not require this wrapper, since the

inputs to the CSC engine would be driven by the other hardware components of a commercial printer. The remaining implementation steps are valid for both the prototype system and a commercial version of such a system.

The original HP CSC engine which includes both the 3D and the 4D phase in addition to the other phases of the CSC pipeline does not fit on the FPGA of the CSC board. The prototype system leverages the fact that the 3D and the 4D phases are never active at the same time to fit the functionality of the CSC engine in this FPGA. It implements a modified version of this engine which uses partial reconfiguration to swap in and out the 3D or the 4D phase from the FPGA as required. Figure 3 illustrates the design. Partial reconfiguration, and in particular the proposed methodology, can also be used to incorporate new features into existing designs, provided that these features are mutually exclusive with at least a module of the original design.

Two module-based PR projects implement only one of the mutually exclusive CSC engine phases, namely the 3D or the 4D, as their PRMs and both the rest of the CSC pipeline phases and the logic required by the proposed methodology in their static regions. In the prototype system, the depacketization logic is implemented in the static region also. Bus macros need to be used to connect the PRR to the rest of the design in the top level [7, 16]. For behavioral simulation, we replace each bus macro by a simple module consisting of pass-through wires. This allows the behavioral models of both PR projects to be verified using a simulator such as ModelSim. However, since behavioral models cannot change at run-time, the logic associated with the proposed PR methodology cannot be verified in behavioral simulation. This logic can only be verified through a hardware test bench like the TestRig board described above.

Finally, firmware for the TestRig board's embedded processor needs to be developed for this board's hardware to be able to initially just send CLUT values and conversion data to the CSC board. Had the commercial embedded processor and DMA controller been used, this effort would not have been necessary. The final modification in this application of the proposed methodology is for the TestRig board's firmware to be enhanced so that its hardware is also capable of initiating the PR process and sending the PR data through the interconnects connected to the CSC engine's register and pixel interface respectively. These firmware enhancements are also possible in commercial printers, since here an embedded processor also drives the interconnects above either directly or indirectly. An example of the indirect control of the commercial processor over these interconnects is the fact that its firmware programs the commercial DMA controller to transfer pixel data to the CSC engine. In conclusion, the interface timesharing technique remains where firmware modifications are the only requirement for sending PR related information to the CSC engine through either the

TestRig or the commercial components that drive the CSC engine.

### 3.4. Prototype Testing

In addition to the CSC design, HP has also provided us with the CSC application. This application is a software program that emulates the hardware-based CSC engine. It can perform any conversion that is supported by the hardware-based CSC engine. One input to the CSC application is a file that contains the CLUT values required for the desired conversion. The other input to this program is the image file to be converted. The output is the same image converted to the target color space.

In commercial HP printers, these inputs are sent to the CSC engine as follows. First, the CLUT values are sent through the register interface if a new conversion type is to be performed. Then, the pixel data is sent through the pixel interface. Due to the packetization required in the prototype system, a *CSC frame* is defined here as a group of CSC packets that instruct the CSC engine to load its CLUTs and process an image. To build this frame in the TestRig board, a new *test bench* (TB) file that combines both of the CSC application input files as well as the input control signals of the CSC engine in a cycle-by-cycle basis is required.

As an initial test case, the firmware on the TestRig board first reads this new TB file from a CF card, parses it, and stores it in DDR memory as a CSC frame. Then, a user request on the TestRig board initiates the transfer of this frame to the CSC board. The output data bus of the CSC design is routed to the CSC board's FPGA GPIO lines. This allows a *logic analyzer* (LA) to collect the output data of the CSC engine while a CSC frame is being sent from the TestRig board. Finally, the output collected in the LA is brought into a MATLAB environment along with the output produced by the CSC application of the same conversion. A MATLAB program simply performs a bit-wise comparison between these two output files to verify the correct functionality of the modified FPGA-based CSC design.

In an additional test case, the TB file is expanded to include PR data within it. To send this additional data to the CSC engine, the register interface is used to inform the CSC engine that the following data in the pixel interface is PR data. Then the PR data is sent through the pixel interface. This reconfigures the PRR with either the 3D or the 4D phase as required. Finally, the CLUT values and the pixel data are sent through the register and pixel interface respectively. Note that the structure of the CSC packet does not change to accommodate the PR data. In other words, the existing CSC engine interface does not need to be modified for the PR data to be sent to this engine, which is indeed one of the main contributions of the proposed methodology.

For testing, we use two color-space conversions. One re-

**Table 1. CSC engine implementation results (XC2VP30).**

Feature	PRR	Static Region and PRR	Available Resources
Slices	4,407	10,035	13,696
BRAMs	60	92	136
Slice Flip Flops	1,399	4,313	27,392
4 input LUTs	7,214	16,732	27,392
Max. clock rate	50MHz		

quires the 3D phase and the other requires the 4D phase. Back-to-back 3D and 4D conversions are performed in the prototype system using the above approach. Finally, the output of these conversions is compared against the output of the CSC application using the MATLAB program above to verify the functionality of the PR logic of the FPGA-based CSC design.

#### 4. Results

The results of the prototype are presented in this section. First, a quantitative analysis regarding the FPGA resource utilization in the prototype is presented. Then, the performance of the prototype is discussed to determine if the processing speed requirements of the application are met.

The FPGA resource usage of the PR CSC engine prototype is provided in Table 1. The PRR of this design uses one third of the slices available in the array. In addition, the logic required to perform partial reconfiguration in this prototype employs less than 0.5 percent of the slices in the FPGA. Therefore, the FPGA slices required to implement the CSC engine would increase approximately by 32.5 percent if the implementation of the CSC engine did not use some sort of run-time reconfiguration. In fact, this implementation would require an FPGA denser than the one of the prototype, since 73 percent of the FPGA slices are required to just implement a single PRM and the static modules. The proposed methodology provides one way to use run-time reconfiguration without having to modify the external interface of the design.

On the other hand, implementing a design originally developed for an ASIC in an FPGA always has some consequences. Significant performance degradation in the maximum frequency of the CSC engine has been the most important hit encountered in the prototype implementation (from 166 MHz to 50 MHz). The inherent degradation of porting a design from an ASIC to an FPGA implementation can be minimized by modifying the modules in this design to leverage the FPGA architecture (*e.g.*, deeper pipelining). Nonetheless, in spite of the performance degradation when

**Table 2. Reconfiguration performance.**

User Logic	Bit-stream Size	Reconfiguration Time	
		w/TestRig (measured)	raw design (calculated)
PRR	465 KB	550.4 ms	9.5 ms
Full FPGA	1414 KB	n/a	29.0 ms

**Table 3. CSC engine performance.**

Action	Number of CSC Cycles	Processing Time	
		w/TestRig (measured)	raw design (calculated)
Load CLUTs	6.8 K	7.9 ms	0.14 ms
160x120 image	19.2 K	22.2 ms	0.38 ms
8.5"x11" image	33.7 M	n/a	0.67 sec

compared to the original ASIC implementation, the prototype system could still be used in some of the commercial printers, namely low-end printers, in which the CSC engine is not driven at full speed. In this context, a realistic goal for the prototype system is to reconfigure the PRR, load the CLUT values and process an 8.5 inch by 11 inch image within one second.

Table 2 shows both the measured and the calculated reconfiguration times of the entire FPGA design and that of the PRR. Table 3 reports the measured and the calculated processing times for loading CLUT values and converting different image sizes in the prototype system. The calculated reconfiguration times have been obtained based on the maximum speed of the ICAP of the Virtex II-Pro FPGA, which is 50MHz [19]. The calculated processing times have been obtained based on Xilinx's post-place-and-route static timing analysis of the PR CSC design, which reports a maximum speed of 50MHz also. This match in maximum frequency of operations is optimal for reusing an existing interconnect to send the PR data to the design. According to the calculated values, reconfiguring the PRR, loading the CLUT, and processing a full page takes 683ms. This is well within the target of one second.

Note that the reconfiguration and processing times measured are about 60 times slower than that of the calculated speed of the prototype. This is due to the hardware used to emulate the actual components that drive the CSC engine in a printer. Nevertheless, the times measured are faster than many of the results obtained by other methodologies proposed in the literature [8, 12].

#### 5. Conclusions

We have presented a methodology that enables self-partial reconfiguration in mature MC systems through addi-

tional yet negligible logic and firmware modifications inside and outside the FPGA of these systems respectively. The main contribution of the proposed methodology is that the connections amongst the *integrated circuits* (ICs) of these systems do not need to be modified to enable partial reconfiguration, and therefore the optimal use of FPGA resources in these systems. This allows new hardware-based algorithms to be devised in MC systems where the array resources are scarce and BOM modifications not possible.

Furthermore, this paper presents a prototype that has two main purposes. First, it demonstrates the successful application of the proposed methodology where the interface of the FPGA-based design does not change for partial reconfiguration features to be incorporated in the MC system. Second, this prototype has also shown that it is feasible to replace ASIC designs with FPGA-based implementations provided that there is some tolerance in terms of circuit performance and that the design has some mutually exclusive modules. The flexibility of programmable logic like FPGAs opens the possibility of rapid time-to-market development projects in these applications.

Future work will explore FPGA-oriented optimizations such as deeper pipelining and *partial evaluation* [5, 17], within the context of the CSC application. Such optimizations may narrow the gap in throughput between the FPGA-based implementation and the original ASIC.

## References

- [1] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan. A self-reconfiguring platform. In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, volume 2778 of *Lecture Notes in Computer Science*, pages 565–574. Springer, 2003.
- [2] B. Blodget, S. McMillan, and P. Lysaght. A lightweight approach for embedded reconfiguration of fpgas. In *Proc. Design, Automation and Test in Europe (DATE)*, page 10399. IEEE Computer Society, 2003.
- [3] P. Bomel, G. Gogniat, and J.-P. Diguët. A networked, lightweight and partially reconfigurable platform. In *Proc. Int. Workshop Applied Reconfigurable Computing (ARC)*, pages 318–323. Imperial College London, Mar. 2008.
- [4] J. Castillo, P. Huerta, V. Lopez, and J. I. Martinez. A secure self-reconfiguring architecture based on open-source hardware. In *Proc. Int. Conf. Reconfigurable Computing and FPGAs (ReConFig)*, page 10. IEEE Computer Society, 2005.
- [5] A. DeHon, J. Adams, M. DeLorimier, N. Kapre, Y. Matsuda, H. Naeimi, M. Vanier, and M. Wrighton. Design patterns for reconfigurable computing. In *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 13–23. IEEE Computer Society, 2004.
- [6] J. Delahaye, G. Gogniat, C. Roland, and P. Bomel. Software radio and dynamic reconfiguration on a dsp/fpga platform. *Frequenz*, 58(8):152–159, 2004.
- [7] N. Dorairaj. Planahead software as a platform for partial reconfiguration. Xcell Journal [Online]. Available: <http://www.xilinx.com/publications/xcellonline/xcell155/xcpdf/xcpmethod55.pdf>, Dec. 2005.
- [8] R. J. Fong, S. J. Harper, and P. M. Athanas. A versatile framework for fpga field updates: An application of partial self-reconfiguration. In *Proc. IEEE Int. Workshop Rapid System Prototyping (RSP)*, page 117. IEEE Computer Society, 2003.
- [9] P. Green and L. MacDonald, editors. *Colour Engineering: Achieving Device Independent Colour*. John Wiley and Sons Ltd., 2002.
- [10] M. Hubner, L. Braun, D. Gohringer, and J. Becker. Runtime reconfigurable adaptive multilayer network-on-chip for fpga-based systems. In *Proc. IEEE Int. Symp. Parallel and Distributed Processing (IPDPS)*, pages 1–6, Apr. 2008.
- [11] J. M. Kasson, S. I. Nin, W. Plouffe, and J. L. Hafner. Performing color space conversions with three-dimensional linear interpolation. *Journal of Electronic Imaging*, 4(3):226–250, 1995.
- [12] Y. E. Krasteva, A. B. Jimeno, E. de la Torre, and T. Riesgo. Straight method for reallocation of complex cores by dynamic reconfiguration in virtex ii fpgas. In *Proc. IEEE Int. Workshop Rapid System Prototyping (RSP)*, pages 77–83. IEEE Computer Society, 2005.
- [13] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Interconnection networks enable fine-grain dynamic multi-tasking on fpgas. In *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, pages 795–805. Springer-Verlag, 2002.
- [14] J. Noguera and R. Esser. Application-driven research in partial reconfiguration. [Online]. Available: <http://ce.et.tudelft.nl/cecoll/slides/07/0524noguera.pdf>, Mar. 2007.
- [15] T. Pionteck, R. Koch, and C. Albrecht. Applying partial reconfiguration to networks-on-chips. In *Proc. Int. Conf. Field Programmable Logic and Applications (FPL)*, pages 1–6. IEEE, 2006.
- [16] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght. Modular dynamic reconfiguration in Virtex FPGAs. *IEE Proceedings - Computers and Digital Techniques*, 153(3):157–164, 2006.
- [17] S. Singh, J. Hogg, and D. McAuley. Expressing dynamic reconfiguration by partial evaluation. In J. Arnold and K. L. Pocek, editors, *Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 188–194. Napa, CA, Apr. 1996.
- [18] J. Thorvinger. Dynamic partial reconfiguration of an fpga for computational hardware support. Master's thesis, Dept. Electroscience., Lund Institute of Technology, June 2004.
- [19] Xilinx. Virtex ii-pro fpga user guide. Application Note [Online]. Available: <http://www.xilinx.com/support/documentation/userguides/ug012.pdf>, Nov. 2007.