MODULAR PARTIAL RECONFIGURATION IN VIRTEX FPGAS

Pete Sedcole *

Dept. of Electrical & Electronic Engineering, Imperial College London, UK

Brandon Blodget, James Anderson Patrick Lysaght

Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, USA

Tobias Becker

Universität Karlsruhe (TH), Kaiserstraße 12 - 76131 Karlsruhe, Germany

ABSTRACT

Modular systems implemented on Field-Programmable Gate Arrays can benefit from being able to load and unload modules at run-time, a concept that is of much interest in the research community. While dynamic partial reconfiguration is possible in Virtex series and Spartan series FPGAs, the configuration architecture of these devices is not amenable to modular reconfiguration, a limitation which has relegated research to theoretical or compromised resource allocation models. In this paper two methods for implementing modular dynamic reconfiguration in Virtex FPGAs are compared and contrasted. The first method offers simplicity and fast reconfiguration times, but limits the geometry and connectivity of the system. The second method, recently developed by the authors, enables modules to be allocated arbitrary areas of the FPGA, bridging the gap between theory and reality and unlocking the latent potential of partial reconfiguration. The later method has been demonstrated in three applications.

1. INTRODUCTION

The transistor density of Field Programmable Gate Arrays has reached a level where an entire system may be implemented within a single device. A complex system is generally composed from many functionally discrete modules, which are connected to form a coherent whole. In some cases where the requirements on the system are time-variant, not all modules need to operate concurrently. An unused module resident in the FPGA will waste power, area and cost, and therefore it would be advantageous if modules are able to be loaded only when an application is invoked and removed again once the application has terminated.

There has been a large amount of research in the area of dynamic modular systems in FPGAs [1, 2, 3, 4, 5]. These are predicated on the property of dynamic partial reconfiguration, however module-based reconfiguration has not been intrinsically supported in FPGAs since the demise of the Xilinx 6200 series. While the Virtex and Spartan series of FPGAs are partially reconfigurable, the essentially linear organisation of the configuration memory is not amenable to the implementation of module-based systems with two-dimensional floorplans. As a result research has tended to be either theoretical, or severely circumscribed, typically by reducing the resource model to one dimension.

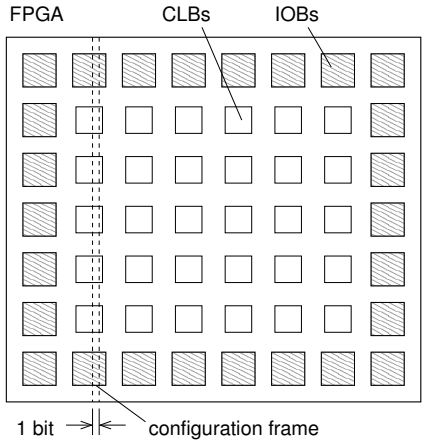
In this paper two methods for implementing dynamic partial reconfiguration on Virtex FPGAs are compared. In the first method, modules must occupy the full height of the device and the topology and connectivity are limited to 1-D. This *direct partial reconfiguration* is fast and simple, and has been previously documented [6]. The second method, recently developed by the authors, demonstrates how 2-D modular systems can be made tractable through the use of an innovative bitstream merging process and reserved routing. This enables modules to be assigned arbitrary rectangular regions of the FPGA and relocated at run-time, bridging the gap between theory and reality. Moreover, it is possible to achieve much greater flexibility in the connectivity of the system. The costs of these advancements are increased complexity and reconfiguration time.

The novel second method, termed *merge partial reconfiguration*, has similarities to the PARBIT tool and design methodology developed by Horta *et al.* for the FPX platform [7]. PARBIT operates on bitstreams, inserting modules into a target area of a device, and even re-targeting the bitstream for a different sized device [8]. The work presented in this paper differs most significantly in the following ways: (a) static routing is possible in reconfigurable regions, (b) bitstreams are integrated at run-time, (c) the target bitstream is read from configuration memory before the integration operation, which enables (d) more sophisticated integration operations to be used.

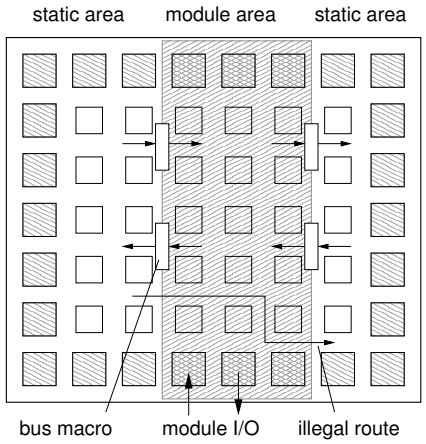
2. VIRTEX CONFIGURATION ARCHITECTURE

The configuration architecture of the Virtex family of FPGAs is described in a Xilinx Application Note [9], and is essentially the same for Virtex II / Pro devices. The configuration is stored in SRAM memory which can be read from

*The authors wish to thank Jean Belzile, Normand Leclerc, Pierre-André Meunier and David Roberge from ISR Technologies for their invaluable contributions, and also Peter Cheung for his critique of this paper.



(a) Configuration architecture



(b) Direct reconfiguration

Fig. 1. Virtex configuration architecture and the direct reconfiguration method.

or written to without halting the device. The smallest unit of configuration memory that can be read or written is a *frame*, which spans the entire height of the device (including I/O blocks) and a fraction of one column (see Fig. 1(a)).

It should be noted that Virtex II / Pro FPGAs have the characteristic of 'glitchless partial reconfiguration': if a configuration bit holds the same value before and after configuration, the resource controlled by that bit will not experience any discontinuity in operation [10], with the exception of LUT RAM and SRL16 primitives. It is therefore possible for a reconfigurable module to occupy an arbitrary area, provided that (a) the areas above and below the module area do not contain LUT RAM or SRL16 logic, and (b) the configuration data written to these areas when the module is replaced overwrites the existing configuration with exactly the same values. Similarly, static, system-level routing may pass through a reconfigurable region if its configuration data are persistent when the module is reconfigured.

3. DIRECT PARTIAL RECONFIGURATION

In the direct partial reconfiguration process, reconfigurable modules are composed from complete frames of configuration memory. This implies that a module occupies the full height of the device, including the I/O at the top and bottom of the reconfiguration region (see Fig. 1(b)). The module may be a variable number of CLB columns in width, and all logic and routing within the reconfiguration region are dedicated to the module. Using this scheme, a module may be replaced very simply, by writing over the existing configuration for the frames that coincide with the module area, using a partial bitstream. *Bus macros* are predefined units of logic and wiring that ensure the locations at which signals pass between the module and the rest of the system are preserved from module to module. More information on this method can be found in [6].

There are a number of limitations with this approach. The design flow for creating the bitstreams for the static portion of the design and the modules leverages Xilinx's Modular Design^{TM} methodology, which restricts the position of modules in the logical hierarchy of the design to the top level. Driver contentions can occur if one module configuration is written over immediately with another, which must be avoided by replacing the module with the default empty configuration before loading in the next module. For large devices a full height module is not the most efficient use of resources, and timing closure can become problematic for modules with large aspect ratios. Finally, while it is possible to pass signals across the reconfigurable area via bus macros on either side, it is highly probable that the signal path will be re-routed during reconfiguration, making the signal non-valid at this time. This means that while a module is undergoing reconfiguration the parts of the system on either side of the module are isolated from each other.

4. MERGE PARTIAL RECONFIGURATION

The merge partial reconfiguration method was created in order to circumvent the limitations of direct reconfiguration, and exploits the glitchless reconfiguration property of Virtex FPGAs. As noted in Section 2, a statically routed signal can pass through a reconfigurated region unperturbed provided the configuration bits associated with the route persist in the new configuration. However, since the module designs are placed and routed independently from the static part of the design, the resources allocated to a static route could also be used in one or more module implementations. This is avoided through the use of *reserved routing*: within a module region, certain routing resources are always reserved for static routing, and modules must avoid using any of these resources, even if unused by the static design. For example, in the Virtex routing architecture each horizontal and vertical channel has 24 long lines and 120 hex lines as well as other

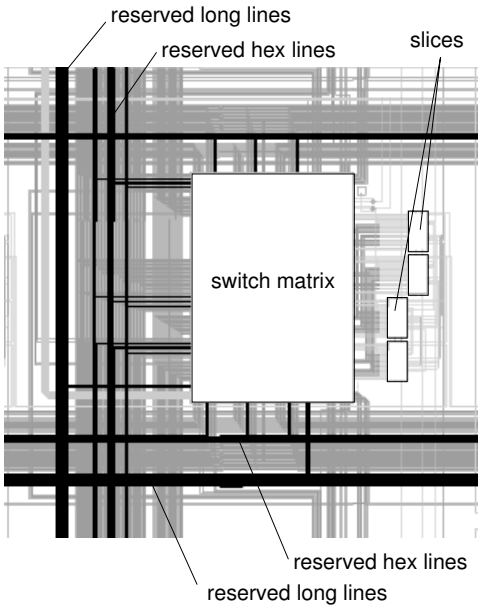


Fig. 2. One CLB, with reserved routing resources highlighted.

more local routing resources; we chose to allocate 20% of the hex lines and 100% of the long lines within module regions to statically routed signals (see Fig. 2). The production router in the ISE `par` tool has the ability to follow very specific constraints, but unfortunately there is no way to provide `par` with these constraints in the current tool flow. Therefore, a custom tool was used to generate the constraints and perform rerouting in a post-`par` step.

The second major innovation in merge reconfiguration is in the way the partial bitstream is loaded. Rather than writing the bitstream directly to the configuration memory, the current configuration is read back from the device and modified with information from the partial bitstream before being written back. This is performed on a frame-by-frame basis, which minimises the amount of memory required to store the bitstreams. As a result, it is possible to have two or more module regions vertically aligned within the device, by masking off parts of the configuration outside the region of interest during the modify step.

Within the module region, static routing is preserved by using an exclusive OR (XOR) operator to merge the partial bitstream with the current configuration. If information is present in both the original configuration and the partial bitstream it will be removed by the XOR operation; in order to prevent this it is necessary to remove all redundant static information from the partial bitstream, which can be done in a straightforward post-processing step at design time. The XOR merge technique has a number of advantages:

- While it is still necessary to remove the module before loading a new one, the same operation and bitstream can be used for loading and unloading, since repeating the XOR operation returns the value to the original state ($a \oplus b \oplus b = a$). This reduces the amount of storage required, as a default empty bitstream is no longer needed.

- Since the module includes no information on static routing, it is position-independent. This is significant, since it means modules can be relocatable. This feature has been demonstrated (see Section 5.2).

- A module can be loaded in several stages, by separating information into several bitstreams which are then effectively overlayed on one another. An example where this ability is useful given in Section 5.3.

As Fig. 3 shows, the static and module bitstreams are created in separate parallel design flows, and a simple post-processing step is performed on the module bitstreams to remove redundancy.

5. EXPERIMENTS AND APPLICATIONS

The novel merge partial reconfiguration method has been applied in three applications; these are described in this section. All three scenarios employ the self-reconfiguring platform reported by Blodget *et al.* [11]. It should be noted that while this was a convenient framework for development, particularly as bitstream manipulation functions can be easily added by extension of the existing driver, self-reconfiguration is not a necessity and the functionality could be provided by an external embedded processor or even by a PC.

5.1. Software Defined Radio

In a collaboration between Xilinx, Inc. and ISR Technologies, a demonstrator of a Software Defined Radio (SDR) has been developed. While part of the radio is software-based, the modulation and demodulation is performed by hardware modules (PLB peripherals) which are loaded using partial reconfiguration. Intended as a proof-of-concept, the demonstrator was developed with a predecessor of the merge reconfiguration method which allows for a single module design only per reconfiguration region. In addition, static configuration information is incorporated into the module bitstreams at design-time, rather than at run-time. Nevertheless, the SDR demonstrates the advantages obtained by exploiting the property of glitchless reconfiguration: the modules are less than the full height of the device (see Fig. 4) and there are hundreds of statically routed signals that pass through the reconfigurable regions. Note also that new, slice-based macros were developed to enable greater signal densities at module interface points.

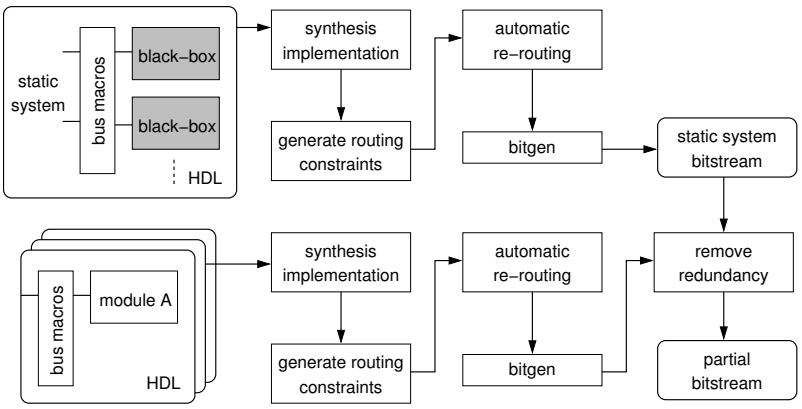


Fig. 3. The design flow for merge partial reconfiguration.

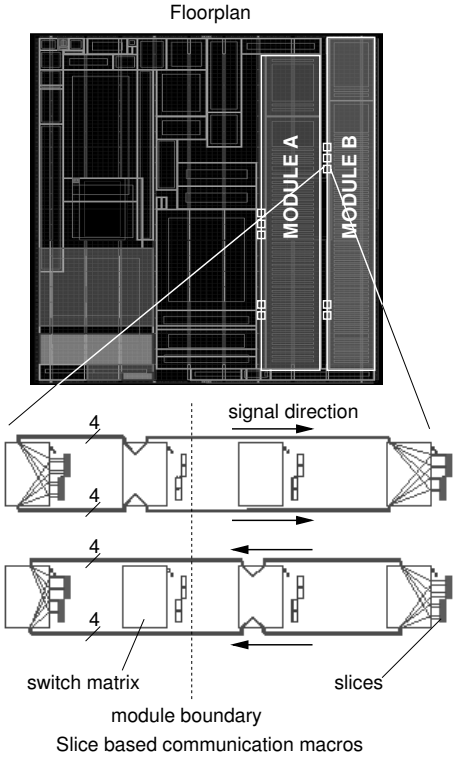


Fig. 4. The floorplan for the XC2VP40 used in the SDR demonstrator, showing the locations of the two modem modules, and the new slice-based communication macros.

5.2. Microprocessor Peripheral

The second experiment was a test framework created to assist the development of the merge partial reconfiguration method, and was used in particular to demonstrate the retargeting of a module bitstream. The test setup used the Xilinx ML300 development board, based on a XC2VP7 part. The FPGA has 34 CLB columns, with one PowerPC processor embedded towards the right hand side of the device in columns 20 to 27. Due to the layout of the board the external DDR-RAM memory is connected to I/O blocks on the left hand side of the FPGA. Space was allocated in columns 3 to 10 for two reconfigurable modules, placed one above the other. The signals from the processor subsystem to the DDR-RAM are necessarily routed through the area for the reconfigurable modules, and must persist during reconfiguration since the program code and module partial bitstreams are stored in the external memory. Two very simple bus peripheral modules (a single register, and a one's complimenting register) were designed, which attached directly to the on-chip Processor Local Bus. The slice based bus macros from the SDR demonstrator were reused for this design.

Following the procedure for merge partial reconfiguration, bitstreams for the static microprocessor subsystem and the two modules were created in three separate implementation phases. The two module designs both targeted the lower of the two reconfiguration regions. The modules were successfully loaded into and unloaded from the lower target location. In addition, by using a simple shift in the XOR merge operation, the modules were re-targeted to the upper location at run-time. An illustration of this is shown in Fig. 5.

5.3. Sonic-on-a-Chip

In the previous two examples modules are connected directly to the PLB bus; this is not the only, nor necessarily the most effective, connectivity model. The final study involved the application of merge reconfiguration to a Sonic-on-a-Chip prototype [12], also using the ML300 development board. Sonic-on-a-Chip, an architecture for video image processing systems, uses a custom bus structure and protocol, designed to be a lightweight solution specifically for dataflow applications. Bus routing in this case is locked to specific routes using hard macros.

The module interfaces to the bus through tristate drivers, which created an interesting problem. In order for the bus

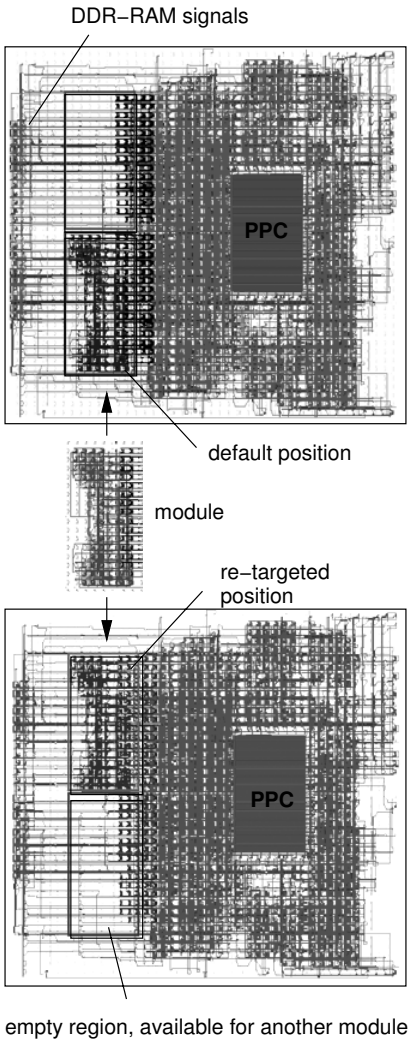


Fig. 5. Modules loaded in default and re-targeted positions. Note that the images are composites, and are an illustration of what is happening inside the FPGA.

to continue uninterrupted during reconfiguration, the tristate drivers need to be disabled. However, these drivers belong to the module, and the disabling signal would need to be routed from within the module – routing which would be in flux during the reconfiguration process.

The chosen solution to this involved a two-phase module reconfiguration process. The static and module designs were implemented as per Fig. 3. For each module implementation, a second design was created by copying the original implementation and disconnecting the tristate buffers from the bus lines. Two partial bitstream were generated, the first containing the configuration for the disconnected module, and the second comprising the difference between the two designs (generated using the `bitgen -r` option). By removing all redundancy from the second bitstream, it carries

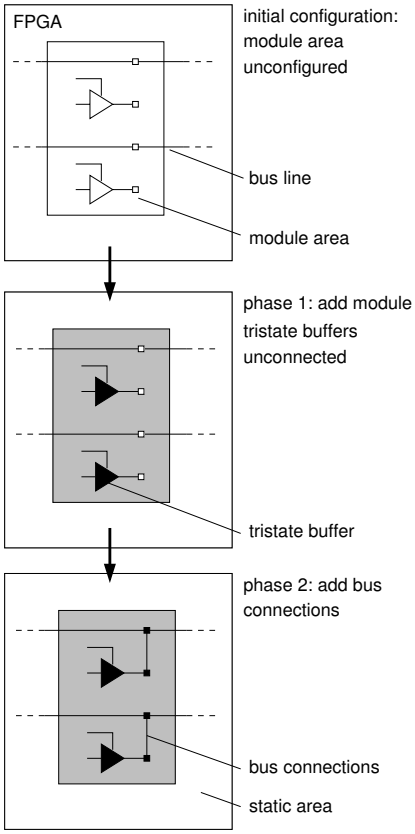


Fig. 6. A module loaded in two phases.

just the information required to connect the module to the bus lines. The module could thus be loaded with two successive merge operations, as depicted in Fig. 6. Removing the module is done by repeating these operations in reverse.

5.4. Configuration Overhead

The use of a read-modify-write operation to configure partial bitstream comes at a cost of increased configuration time. Using empirical measurements, it was ascertained that the configuration time for a direct partial reconfiguration operation can be approximated by:

$$T_d \approx f \times \left(\frac{1}{t} + \frac{1}{w_1} \right)$$

Where f is the equivalent number of frames in the bitstream, t is the rate at which a frame is transferred from memory to the OPB peripheral, and w_1 is the rate at which a frame is written into the ICAP.

Using the read-modify-write operation requires the partial bitstream to be processed in software to identify which frames need to be read from the device; the frames must be read; parts of the frame modified; and the frames written

back. The configuration time is approximately:

$$T_m \approx f \times \left(\frac{1}{p} + \frac{1}{r} + \frac{c}{m} + \frac{1}{w_2} \right)$$

Where p is the rate at which the bitstream is processed, r is the rate at which frames are read from the device, m is the modification rate per row of CLBs, c is the number of CLB rows the module occupies, and w_2 is the rate at which frames are written back to the ICAP. It should be noted that when reading configuration information, the data are prepended by a 'pad' frame. Similarly, when writing a configuration, an extra pad frame is required after the final frame of real data. This means operating on a single frame at a time is much slower than operating on several contiguous frames in one go, and is one reason $w_2 \ll w_1$.

Using the Sonic-on-a-Chip platform (system CPU/bus clock speed 100MHz, 16MB data cache + 16MB instruction cache), we obtained the following values for the parameters (all in frames/ms unless noted): $t = 7.86$, $w_1 = 117$, $p = 4.15$, $r = 30.8$, $m = 185$ CLB rows/ms, $w_2 = 19.3$. From these values, it can be calculated that time for the read-modify-write configuration in this case is between 2.4x and 4.0x slower than that for the write-only configuration, depending on the height of the module. For example, for a module 21 CLB rows high and 15 CLB columns wide $T_d = 47.5$ms, and $T_m = 151.4$ms.

However, in both situations a large percentage of the configuration time is due to inefficiencies in the driver software, particularly where data transfer is involved. Ignoring these inefficiencies one can derive a lower bound to the inherent overhead. By measurement and inspection we estimated the intrinsic values of $r = 54.9$, $w_2 = 46.1$, and $t = 60.9$. Therefore, the relative increase in configuration time of the merge method is at least:

$$\left(\frac{1}{r} + \frac{1}{w_2} \right) / \left(\frac{1}{t} + \frac{1}{w_1} \right) = 1.58$$

6. CONCLUSION

This paper presented two partial reconfiguration methods for modular systems in Virtex FPGAs. The first method uses partial bitstreams directly to reconfigure the FPGA. However due to the organisation of the configuration memory in Virtex devices, modules exclusively occupy complete vertical sections of the device, which severely restricts resource allocation and connectivity. These restrictions are avoided with the second, novel, merge partial reconfiguration method. In this method, information in the partial bitstream is merged with the current configuration as read back from the device. By using an exclusive-OR function to combine the two bitstreams, existing configuration information is preserved, and the module can be removed by repeating the XOR operation. Modules can be allocated any rectangular region in the device, and static routes can pass through reconfiguration regions. To avoid conflicts, some of the routing resources are reserved for the static routes.

The merge partial reconfiguration has been employed in three applications, which have successfully demonstrated run-time re-targeting of module bitstreams and multi-phase reconfiguration. From one of these applications, speed measurements have revealed an increase in configuration times of between 2.4x to 4.0x, with a baseline overhead of at least 1.58x.

7. REFERENCES

[1] G. Brebner and O. Diessel, "Chip-based reconfigurable task management," in *Proc. Field–Programmable Logic and Applications*, 2001.

[2] J. Burns, A. Donlin, J. Hogg, S. Singh, and M. de Wit, "A dynamic reconfiguration run-time system," in *Proc. IEEE Symposium on FPGAs for Custom Computing Machines*, 1997.

[3] J.-Y. Mignolet, V. Nollet, P. Coene, D.Verkest, S. Vernalde, and R. Lauwereins, "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip," in *Design, Automation and Test in Europe*, 2003.

[4] C. Steiger, H. Walder, and M. Platzner, "Heuristics for online scheduling real-time tasks to partially reconfigurable devices," in *Proc. Field–Programmable Logic and Applications*, 2003.

[5] G. B. Wigley, D. A. Kearny, and D. Warren, "Introducing ReConfigMe: An operating system for reconfigurable computing," in *Proc. Field–Programmable Logic and Applications*, 2002.

[6] D. Lim and M. Peattie, "Two flows for partial reconfiguration: module based or small bit manipulation," Xilinx, Application Note 290, 2002.

[7] E. L. Horta, J. W. Lockwood, and S. Kofuji, "Using PARBIT to implement partial run-time reconfigurable systems," in *Proc. Field–Programmable Logic and Applications*, 2002.

[8] E. L. Horta and J. W. Lockwood, "Automated method to generate bitstream intellectual property cores for Virtex FPGAs," in *Proc. Field–Programmable Logic and Applications*, 2004.

[9] "Virtex series configuration architecture user guide," Xilinx, Application Note 151, 2004.

[10] B. Blodget, C. Bobda, M. Huebner, and A. Niyonkuru, "Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs," in *Proc. Field–Programmable Logic and Applications*, 2004.

[11] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan, "A self-reconfiguring platform," in *Proc. Field–Programmable Logic and Applications*, 2003.

[12] N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, "A reconfigurable platform for real-time embedded video image processing," in *Proc. Field–Programmable Logic and Applications*, 2003.