INVITED PAPER: ENHANCED ARCHITECTURES, DESIGN METHODOLOGIES AND CAD TOOLS FOR DYNAMIC RECONFIGURATION OF XILINX FPGAS

Patrick Lysaght, Brandon Blodget, Jeff Mason

Xilinx Research Labs
San Jose, Ca. 95124, USA
email: patrick.lysaght@xilinx.com,
brandon.blodget@xilinx.com,
jeff.mason@xilinx.com.

Jay Young, Brendan Bridgford

Xilinx
Longmont, Co. 80503, USA
email: jay.young @xilinx.com,
brendan.bridgford@xilinx.com.

ABSTRACT

We describe architectural enhancements to Xilinx FPGAs that provide better support for the creation of dynamically reconfigurable designs. These are augmented by a new design methodology that uses pre-routed IP cores for communication between static and dynamic modules and permits static designs to route through regions otherwise reserved for dynamic modules. A new CAD tool flow to automate the methodology is also presented. The new tools initially target the Virtex-II, Virtex-II Pro and Virtex-4 families and are derived from Xilinx's commercial CAD tools.

1. INTRODUCTION

An FPGA that is dynamically reconfigurable is capable of being partially reconfigured while operational without compromising the integrity of the applications running on those parts of the FPGA that are not being reconfigured. Researchers have reported many reasons why the ability to time multiplex hardware dynamically on a single FPGA is advantageous. These include applications as diverse as:

- Reducing the size of FPGA required for implementing a given function with consequent reductions in cost and/or power consumption [1, 2, 3]
- Improving FPGA fault tolerance [4, 5]
- Accelerating configurable computing [6, 7]
- Thermal monitoring of the FPGA die [8]

The research community has responded to the challenges of realizing these types of systems by proposing a variety of design flows and CAD tools for the creation of dynamically reconfigurable systems. Perhaps the most influential of these efforts is JBits [9] which is essentially a structural design flow based on the Java programming language. Most of the reported work describes experimental systems that have never been commercialized. A recent attempt at a design environment for commercial FPGAs and CAD tools was reported by Nasi et al [10] but the capabilities of the FPGAs targeted are very modest.

The contribution of this paper is to report new architectural enhancements to Xilinx FPGAs that better support the creation of dynamically reconfigurable designs. In addition, a new design methodology is presented that uses pre-routed IP cores for communication between static and dynamic modules. It introduces a new capability that permits nets associated with the static portion of a design to be routed through regions in which the logic is reserved exclusively for implementing dynamic modules. The CAD tool flow to automate this methodology is based on Xilinx's commercial tools for FPGA design. The combination of new architectural features, new IP cores, an improved design methodology and supporting CAD tools makes the process of designing dynamically reconfigurable systems with industrial quality devices and tools much more accessible.

1.1. Note on terminology

In this paper we use the terms *active partial reconfiguration* and *dynamic reconfiguration* synonymously. Other authors have preferred the term *run-time reconfiguration* (or permutations of the above) to describe essentially the same capability. We also use the terms *static design* and *base design* interchangeably so that during dynamic reconfiguration the static or base design is not reconfigured and hence its operation is not compromised in any way. For historical reasons the new CAD tool flow introduced here is referred to as the *early access partial reconfiguration flow* or PR flow for short. In this context, support for dynamic reconfiguration is implicit.

1.2. Organization

In the next section we review the most important new developments in support of dynamic reconfiguration from an architectural perspective. Three main topics are addressed. The first is the replacement of hard-wired tri-state buffers (TBUFs) with pre-routed *bus macros* as the preferred mechanism for implementing the communication ports to interface static and dynamic regions of a design. The second is the reduction in the granularity of the unit of reconfiguration from a full device column in a Virtex-II or Virtex-II Pro FPGA to a smaller unit of sixteen CLBs in the Virtex-4 architecture. Finally, we describe enhancements

to the internal configuration access port (ICAP) that improve the bandwidth of that interface. In section 3 we present the new design methodology for dynamically reconfigurable circuits on Virtex-II, Virtex-II Pro and Virtex-4 FPGAs. Section 4 concludes the paper and identifies some opportunities for future work.

2. NEW FPGA CAPABILITIES

2.1. Bus macros replace TBUFs

Earlier design flows had recommended the use of tri-state buffers that were hard-wired into the architecture of the Xilinx FPGA architectures [11]. However, this practice is discouraged with newer architectures for two reasons. The effectiveness of TBUFs was compromised by their pre-determined locations and the fact that they were dispersed across the architecture. Practical design experience highlighted the need for greater concentrations of connections between static and dynamic regions and more flexibility in where they could be located.

The removal of TBUFs from newer architectures also made the search for an alternative more urgent. The solution is a family of pre-routed macros generically referred to as *bus macros* [12]. These are implemented as relationally placed macros (RPMs) in the Xilinx design flow terminology. An example of a simple bus macro for a Virtex-II device is shown in Fig. 2. It consists of two configurable logic blocks (CLBs), one on either side of the module boundary between a static region and a reconfigurable region or vice versa. Within each Virtex-II CLB there are four logic slices and one slice can implement two unidirectional connections, from left-to right in the case of Fig. 1.

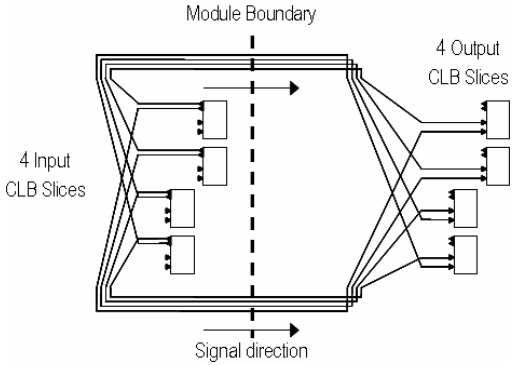

Fig.1 Basic 8-input, 8-output left-to-right bus macro

Each of the four slices on the left has two outputs (X, Y) and these are routed across the module boundary to two of the inputs on each of the corresponding slices on the right hand side. Thus two CLBs can efficiently implement 8 interface ports. Note that the connections to the nets of the static and dynamic regions are not shown in Fig. 2. The nets would be connected to the unused terminals shown on each of the slices. Since CLBs are the basic primitives of all Xilinx FPGA architectures, they are both abundant and ubiquitous, allowing us to create bus macros for any architecture and place them with maximum freedom. Separate bus macros are created for connections in the right-to-left direction. Many other permutations are possible, including the following, which are available with the early access tool release:

- Bus macros that are expanded "horizontally" so that the input and output CLBs are separated by an even number of columns. These are useful to permit "nesting" of macros of successive widths to form denser concentrations for high bandwidth interfaces
- Synchronous bus macros with registered inputs and outputs with optional clock enables
- Vertically oriented bus macros

2.2. Unit of reconfiguration granularity

FPGAs are configured by loading application specific data into their configuration memories. The configuration file consists of a number of packets that typically contain configuration control information as well as the configuration data itself. Virtex-II and Virtex-II Pro devices are dynamically reconfigurable and a single frame is the smallest unit of reconfiguration. The precise number of frames and the number of bits per frame are device-dependent for the different devices in the family. The number of bits per frame is proportional to the height of the device, measured in CLBs. For example the smallest family member, the XC2V40, consists of an array of 8 by 8 CLBs while the largest XC2V8000 has 112 by 104 CLBs. Consequently, the unit of reconfiguration, as measured in bits, scales adversely with increasing device size, from 832 bits in the best case to 9,152 bits in the worst case.

In Virtex-4 devices, the unit of reconfiguration granularity is a smaller, bit-wide column corresponding to 16 CLBs (or integer multiples thereof) and is independent of device size or family (LX, SX or FX). All Virtex-4 configuration frames consist of forty-one 32-bit words resulting in a total of 1,312 bits per frame.

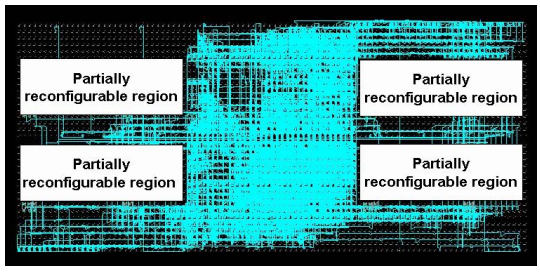

Fig. 2 V-4 LX25 with 4 partially reconfigurable regions

It is now feasible to dynamically reconfigure regions as small as 16 CLBs high and to have more than one of these regions arranged vertically for the first time as shown in Fig. 2. Note that the floorplan of the Virtex-4 LX25 is shown rotated through 90 degrees in the diagram.

2.3. Upgraded internal configuration access port

The Virtex-II and Virtex-II Pro were the first Xilinx architectures to have internal reconfiguration access ports (ICAP). These ports provide an 8-bit input data bus and an 8-bit output data bus which can be used by an internal controller to reconfigure and read back configuration memory. When combined with an internal soft or hard microprocessor the ICAP has been used to build self-controlling, dynamically reconfigurable systems [13].

SelectMap and ICAP are the external and internal reconfiguration ports for Xilinx FPGAs respectively. The three Virtex-4 families all feature an updated reconfiguration interface. The base functionality and the intended use models of the new ICAP are identical to its predecessor but the new port now has 32-bit wide input and output data buses and is specified to run at the same speed as the external SelectMap interface (100 MHz). When combined with the 16-CLB reconfiguration granularity, the speed of reconfiguration can be increased by as much as an order of magnitude for smaller modules.

Fig. 3 shows the reconfiguration latency in milliseconds (x-axis, top) and reconfiguration frequency in reconfigurations per second (x-axis, bottom) for partial reconfiguration of a range of Virtex-II Pro and Virtex-4 LX devices.

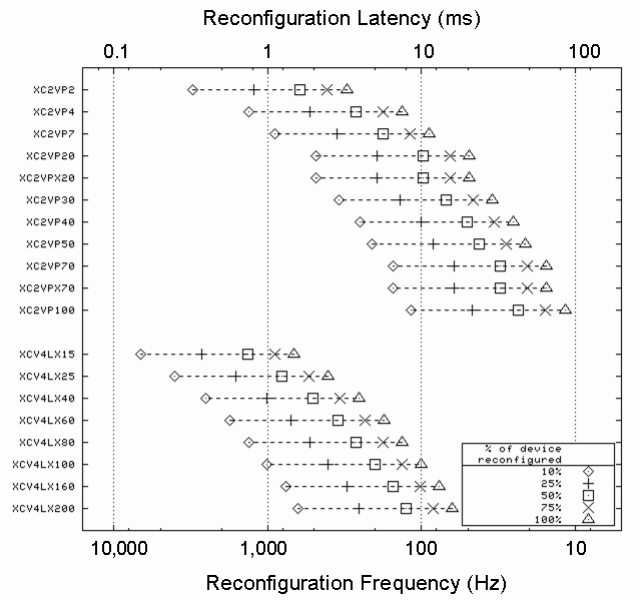

Fig. 3 Virtex-II Pro versus Virtex-4 reconfiguration speeds

The impact of the greater bandwidth of the ICAP interface in the Virtex-4 devices is clear. (Note that nothing about the use of the term *dynamic reconfiguration* implies a particular speed of reconfiguration, though faster reconfiguration is generally more desirable).

3. DESIGN METHODOLOGY

Implementing a dynamically reconfigurable system on an FPGA introduces several additional steps in the design process. The complete partial reconfiguration design flow is shown in Fig. 4. In this section we have decomposed the flow into seven key steps and we consider each in turn. The design process is enabled by two key enhancements to the mainstream design tools.

The first is the removal of the restriction that Virtex-II and Virtex-II Pro devices can only be partially reconfigured in whole columns. As a result, the region being reconfigured can now be of any rectangular size. There is one proviso but this is automatically taken care of by the new tools. For Virtex-II and Virtex-II Pro devices, look-up tables used as distributed RAMs (LUTRAMs) or shift registers (SRL16s) must not appear in the static regions directly above or below the rectangular region being dynamically reconfigured.

The second major change in the new PR software flow permits signals in the static (or base) design to cross through partially reconfigurable regions without the use of a bus macro. This enhancement dramatically improves timing performance and simplifies the process of building a PR design. Virtex devices have glitchless reconfiguration and it is this feature that enables static routes to cross PR regions. Glitchless reconfiguration guarantees that if a configuration bit has the same value before and after reconfiguration no glitch will occur on that bit during configuration. As long as the static routes are implemented identically in every PRM no glitches will occur on them.

3.1. Design phase 1: Planning & synthesis

The designer begins by identifying the opportunity to deploy dynamic reconfiguration. He partitions the design into a static subset and one or more dynamic subsets. The static subset is characterized by a one-to-one mapping between its logical functions and the resources used to implement that functionality. This part of the design maps most closely to the conventional FPGA design flow but as we shall see, some changes will be introduced here also.

The designer may identify one or more dynamically reconfigurable subsets. Each of these dynamic subsets is associated with a set of functions or tasks whose operation is mutually exclusive with respect to each other in time. If the cardinality of a particular mutually exclusive (mutex) set is *n*, then there will be an *n-to-1* mapping between the tasks in that mutex set and the dynamically reconfigurable subset of the FPGA that will be used to implement those

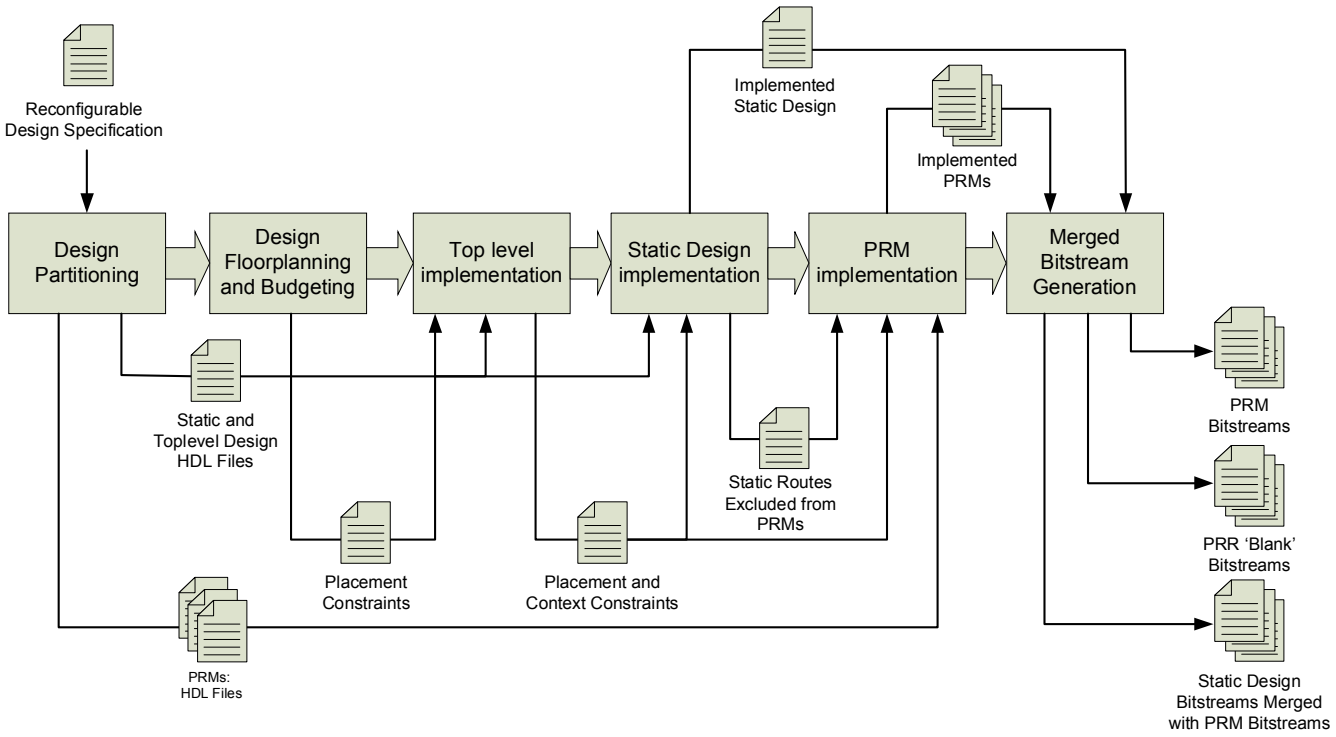

Fig. 4 Partial Reconfiguration Design Flow

functions. In the methodology described here, the term partially reconfigurable region (PRR) is used to describe the area of the FPGA that will be reserved for implementing all of the tasks in one dynamically reconfigurable subset. The term partially reconfigurable module (PRM) is used to describe the implementation of a single dynamic task that will be mapped into a PRR. Each task in the subset will be implemented as a partially reconfigurable module (PRM) that will be swapped in and out of its corresponding PRR at run-time.

The designer must establish precisely the nature of the communication between the modules in a given region and the static logic. For each input or output, an appropriate bus macro of the correct orientation must be allocated. Note that all communication from dynamic regions to or from device I/O must be routed through the static logic.

At the end of the first stage, the design must be organized into a series of HDL files with a pre-determined file structure. The first of these is the top-level file, which we will call *TOP* for simplicity. *TOP* contains all global logic such as clock primitives (e.g. digital clock managers (DCMs) and BUFG global clock buffers); IO port instantiations; bus macro instantiations; signal declarations; base design instantiations; and PR module instantiations. A user constraint file (.UCF) is also created for *TOP*.

Next come all the HDL files for the static portion of the design. Then, finally, the HDL files for each of the PRMs associated with each of the PRRs must be included. In synthesizing the design files, the synthesis parameter IOBUF (in XST, the Xilinx synthesis tool set) must be enabled for TOP and disabled for all other files.

3.2. Design phase 2: Budgeting with PlanAhead

The goal of the budgeting phase is to determine the size and location of the PRRs and to lock down the placement of the bus macros. The budgeting phase can be done manually. The process, however, is laborious and instead many of the steps have been automated with a tool called *PlanAhead*.

PlanAhead is a Xilinx software tool for the design and analysis of circuits on Xilinx FPGAs, and is based on the principles of hierarchical floorplanning. It is deployed between synthesis and place-and-route and enables the designer to more rapidly analyze, modify, constrain and implement their designs. PlanAhead is an optional tool that complements the mainstream ISE tool chain. It is typically deployed for designs requiring the highest performance and consistently leads to faster and more compact solutions. It also promotes design reuse through the creation of reusable intellectual property blocks.

PlanAhead allows us to import the logical design hierarchy in the form of EDIF files. We can then create a corresponding physical hierarchy and map it to an initial floorplan. This step is remarkably straightforward due to the tool's highly intuitive interface and allows us to begin the budgeting phase. PlanAhead uses the concept of a physical block (PBlock) as the basic unit of physical hierarchy. PBlocks partition a design physically and can be

composed hierarchically for block nesting. Once created, the PBlocks can be placed and sized on a floorplan of the target FPGA. Various estimates of performance are immediately available (prior to place and route) to guide the designer in permuting the floorplan and iteratively improving the design.

The partial flow is based on rectangular PRRs. In the budgeting phase we determine the size and location of the PRRs by allocating them to a PBlock. For any PBlock, PlanAhead can interactively provide estimates of the resources used by the PRR as it is moved or re-sized on the floorplan. This makes it very easy to test that each PRM associated with a given PRR will indeed fit in that rectangle and to determine the best shape and location for the PRR. Once the size and location of the PRRs are optimized, the bus macros must be placed in a CLB column inside the PBlock that is close to the PRR boundary so that it can physically straddle the dynamic and static regions.

PlanAhead generates a customized constraints file (.UCF) for each module. The static design components are mapped to a single PBlock and are required to have an AREA_GROUP constraint but do not need a RANGE constraint. All PRMs must have both AREA_GROUP and RANGE constraints. The RANGE constraints enumerate the resources (slices, BRAMs, etc) required by the PRMs.

3.3. Design phase 3: Top level context

After the top level HDL file and UCF have been created and synthesized it is necessary to translate this information into a native Xilinx format. This is done by running the tool NGDBuild with the '-modular initial' option on the synthesized netlist. This creates a file which contains the top level context. The top level context includes IO placement, clock resources, bus macros, static and PRM module placement. This file is used to pass context information to the later static implementation and PRM implementation phases.

3.4. Design phase 4: Static implementation

The purpose of the static implementation phase is to place and route the static subset of the design. This phase requires that the synthesized netlists that comprise the static subset of the design be copied into a reserved directory. The bus macros and user constraint file containing information about the context and static logic must also be copied to this directory. NGDBuild is then run in this directory referencing the top level context file that was generated in phase 3. Finally MAP and PAR are run resulting in a placed and routed file for the static part of the design.

The static design is prohibited from using logic resources in the PR regions. However, the static design is allowed to use routing resources contained in PR regions. Allowing the static design to route through PR regions improves the routability and performance of the static design. The routing resources in the PR regions that are used for static routes are stored in a file called "static.used". This file is used in the PRM implementation phase (described next) and the static routes are copied into every PRM and these routing resources cannot be used by the PRM logic.

3.5. Design phase 5: PRM implementation

The PRM implementation phase builds the PR modules that will go into the PR regions. Each PRM is implemented separately within its own directory. The PRM directories contain the synthesized netlist for the PRM and the user constraints file containing information about the particular PRM and associated context logic. The "static.used" file generated during the static implementation phase is copied into the PRM directories and renamed "arcs.exclude". NGDBuild, MAP and PAR are run in each PRM directory. NGDBuild is run with the '-modular module_name' option and references the top-level context file generated in phase 3. The MAP and PAR tools do not require any special options. PAR automatically looks for and uses any "arcs.exclude" file that is present in the PRM implementation directory.

3.6. Design phase 6: Merge

The final implementation phase is the *merge* phase. During the merge phase, a complete design is built from the static design and each PRM. As many complete designs will be built as there are PRMs. The merge function is performed by running the PR_verifydesign and PR_assemble scripts. These tools are unique to the EA PR software.

The PR_verifydesign script will generate a merged full bitstream and partial bitstream for each PRM. The PR_assemble script is required to generate full bitstreams for designs that have multiple PR regions. The PR_assemble script takes a list of PRMs, one for each PRR, and generates a full bitstream for the merged design. The resulting bitstream can be used as the initial full bitstream with the desired PRMs already in place. The PR_assemble script also generates "blanking" bitstreams for the PR regions. A "blanking" bitstream is a default configuration containing no PRM logic. The "blanking" bitstreams do contain the static route-throughs. Blanking bitstreams can be loaded when a PRM is not required which can reduce power consumption.

3.7. Design phase 7: Bitstream download and test

Once the full and partial bitstreams have been generated they need to be tested. The early access PR version of the

iMPACT (the Xilinx tool for downloading bitstreams to program devices) has been updated to support downloading partial bitstream via JTAG. The general procedure is to first download a full bitstream using iMPACT and then to download the various partial bitstreams to verify that they work as well. Once the full and partial bitstreams have been validated using iMPACT, they can be integrated into an embedded solution.

4. CONCLUSIONS

The current early access version of the PR tools are being offered to well-qualified partners who are willing to deploy them and contribute feedback for their improvement. A number of reference designs are also included. For interested readers, more details are available at:
www.xilinx.com/support/prealounge/protected/index.htm.
As one would expect with an early-access tool flow, there are opportunities for improvements and future work. One immediate opportunity would be to extend device support to include Xilinx's latest Virtex-5 FPGAs.

There are also opportunities to better automate the sizing of PRRs in the PlanAhead tool. At present, the designer must consider each PRM to identify the final requirements on PRR resources. Finally, we have not discussed automating support for design simulation or reconfiguration scheduling in this paper and these would be important candidates for future work.

5. ACKNOWLEDGEMENTS

Many colleagues at Xilinx have contributed both directly and indirectly to the development of the current flow for partial reconfiguration. These include Adam Donlin, Eric Shiflet, Jason Moore, Manuel Uhm and Steve Lass. We would like to record our thanks to all of them.

We would also like to acknowledge that credit for the original idea and deployment of the bus macros goes to the team of Prof. Juergen Becker (including Michael Huebner, Michael Ullmann and Tobias Becker) at the University of Karlsruhe, Germany. The macros described here were inspired by their earlier work, as described in [14, 15].

6. REFERENCES

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

[2] Y. Adachi, K. Ishikawa, S. Tsutsumi, and H. Amano, "An implementation of the Rijndael on Async-WASMII", International Conference on Field-Programmable Technology (FPT), 2003. 15-17 Dec. 2003, pp. 44 – 51.

[3] 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.

[4] J. Emmert, C. Stroud, B. Skaggs, and M. Abramovici, "Dynamic fault tolerance in FPGAs via partial reconfiguration", Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on, 17-19 April 2000, pp. 165 – 174.

[5] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs", Parallel and Distributed Processing Symposium, 2004. Proceedings, 18th International 26-30 April 2004.

[6] K. Bondalapati, and V. K. Prasanna,, "Reconfigurable computing systems", Proceedings of the IEEE Volume 90, Issue 7, July 2002, pp. 1201 – 1217.

[7] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, amd E. M. Panainte, "The MOLEN polymorphic processor", IEEE Transactions on Computers, Volume 53, Issue 11, Nov. 2004, pp.1363 – 1375.

[8] S. Lopez-Buedo, J. Garrido, E. Boemo, "Dynamically inserting, operating, and eliminating thermal sensors of FPGA-based systems", IEEE Transactions on Components and Packaging Technologies, Volume 25, Issue 4, Dec. 2002, pp. 561 – 566.

[9] S. Guccione, D. Levi, and P. Sundararajan, "JBits: Java-based interface for reconfigurable computing", Proc. Conf. on Military and Aerospace Application of Programmable Devices and Technology, 1999.

[10] K. Nasi, T. Karouhalis, M. Danek, and Z. Pohl, Z. "FIGARO - an automatic tool flow for designs with dynamic reconfiguration", Field Programmable Logic and Applications, 2005. International Conference on, 24-26 Aug. 2005, pp. 590 – 593.

[11] "Two Flows for Partial Reconfiguration: Module Based or Difference Based", Xilinx Application Note XAPP290, version 1.1, Xilinx, Inc. (2003).

[12] "Two Flows for Partial Reconfiguration: Module Based or Difference Based", Xilinx Application Note XAPP290 (V1.2), ww.xilinx.com/bvdocs/appnotes/xapp290.pdf, Sept. 9, 2004.

[13] B. Blodget, S. McMillan and P. Lysaght, "A lightweight approach for embedded reconfiguration of FPGAs", Design, Automation and Test in Europe Conference and Exhibition, 2003, pp. 399 – 400.

[14] J. Becker, M. Huebner, M. Ullmann, "Real-Time Dynamically Run-Time Reconfiguration for Power-/Cost optimized Virtex FPGA Realizations", Proceedings of the IFIP International Conference on Very Large Scale Integration (VLSI-SoC), Darmstadt, Germany, December 1 –3 2003, pp. 129 – 134.

[15] M. Hübner, M. Ullmann, L. Braun, A. Klausmann, J.Becker, "Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems", Proceedings of the 14th International Conference on Field Programmable Logic and Application (FPL'04), 2004, August, Springer, pp. 1037-1041.