Module Relocation in Heterogeneous Reconfigurable Systems-on-Chip using the Xilinx Isolation Design Flow

Laurent Gantel, Mohamed El Amine Benkhelifa, Fabrice Lemonnier, François Verdier

Module Relocation in Heterogeneous Reconfigurable Systems-on-Chip using the Xilinx Isolation Design Flow

L. Gantel*† and M.E.A Benkhelifa* and F. Lemonnier† and F. Verdier‡

* *ETIS Laboratory*
CNRS UMR 8051 / UCP / ENSEA
6, avenue du Ponceau
95014 Cergy-Pontoise, FRANCE
{*name*}*@ensea.fr*

† *Embedded System Lab*
Thales Research and Technology
1, avenue Augustin Fresnel
91767 Palaiseau, FRANCE
Fabrice.Lemonnier@thalesgroup.com

‡ *LEAT Laboratory*
CNRS UMR 7248 / UNSA
930 Route des Colles
06903 Sophia Antipolis, FRANCE
Francois.Verdier@unice.fr

***Abstract*—Heterogeneous Reconfigurable Systems-on-Chip *(HRSoC)* contain as their name suggests, heterogeneous processing elements in a single chip. Namely, several processors, hardware accelerators as well as communication networks between all these components. In order to leverage the programming complexity of this kind of platform, applications are described with software threads, running on processors, and hardware threads, running on FPGA partitions. Combining techniques such as dynamic and partial reconfiguration and partial readback with the knowledge of the bitstream structure offer the ability to target several partitions using a unique configuration file. Such a feature permits to save critical memory resources. In this article, we propose to tackle the issue of designing fully independent partitions, and especially to avoid the routing conflicts which can occur when using the standard Xilinx FPGA design flow. To achieve the relocation process successfully, we propose a new design flow dedicated to the module relocation, using the standard tools and based on the Isolation Design Flow *(IDF)*, a special flow provided by Xilinx for secure FPGA applications.**

***Keywords*-FPGA; hardware thread; dynamic reconfiguration; module relocation, Isolation Design Flow;**

I. Introduction

With the flexibility brought by the Dynamic and Partial Reconfiguration *(DPR)*, the adaptation of the thread model on the hardware accelerator became interesting. Hybrid Thread [1] is a hardware thread model based on the POSIX[1] thread model. It contains a memory stack used to stock data parameters. It permits to process system calls understandable by a software thread, running on a CPU. This approach is pursued in ReconOS [2], where the hardware threads and a system call API, both described in VHDL, are proposed to make the programming process easier. In all these cases, as well as in SHUMDR [3], the thread is composed at least by a finite state machine to sequentially control the accelerator execution.

Some works such as [4] or [5] already used the DPR feature provided with Xilinx FPGAs to dynamically load new features into a running design. The thread was relocated from the genuine bitstream. Such an approach allows data-flow treatment but is not flexible enough for more complex accelerators. Context management, as in [6], combined with DPR permits to stop and restart a hardware thread at any time, offering hardware thread preemption capabilities.

All these processes must be executed by an operating system. To reduce timing overhead caused by dynamic reconfiguration, this operating system would have access to services implemented in hardware such as ICAP driver [7] or relocation modules [8].

At present, when implementing dynamic modules in a reconfigurable system, the static routing of this system can cross over the dynamic areas. Consequently, relocating a bitstream without corrupting the static routing is not possible. To deal with this issue, new tools have been developed by the community. RapidSmith [9] offers the ability to implement its own placer-router, whereas OpenPR [10] is an open-source based on the same routing engine which permits to create independent partition using blocker macros [11]. These macros prevent the static routing from crossing inside the reconfigurable partitions. However this tool is not integrated into the standard flow and the number of supported devices is limited. In order to keep using the Xilinx tools, we rely on the new Isolation Design Flow [12] and some additional constraints to design homogeneous and relocatable partitions.

The paper is organized as follow. A brief definition of a HRSoC is given in Section II. Then a hardware thread model is introduced in Section II-A. Section III deals with the management of this kind of thread by an operating system and especially how we perform thread relocation. Finally, a new design flow to create identical dynamic partitions relying on the Isolation Design Flow is detailed in Section IV. To conclude, implementation results as well as conclusion and future work are presented respectively in Section V and VI.

II. Context and Motivations

Reconfigurable platforms based on different processing elements are called Heterogeneous Reconfigurable Systems-

[1] *Portable Operating System Interface*

978-1-4673-2921-7/12/$31.00

on-Chip *(HRSoC)*. In such a platform, the application is divided into tasks. Some tasks are implemented as hardware accelerators and allocated into a partition of the chip, while others run as software tasks on computing processor elements. A hardware accelerator is defined as a hard-wired function developed to accelerate the processing of a task. A computing processor unit could be a General Purpose Processor, a specialized one like a Digital Signal Processor, a Graphics Processing Unit or a simple Micro-Controller Unit. Each one of these processing elements is more or less suited to certain types of tasks.

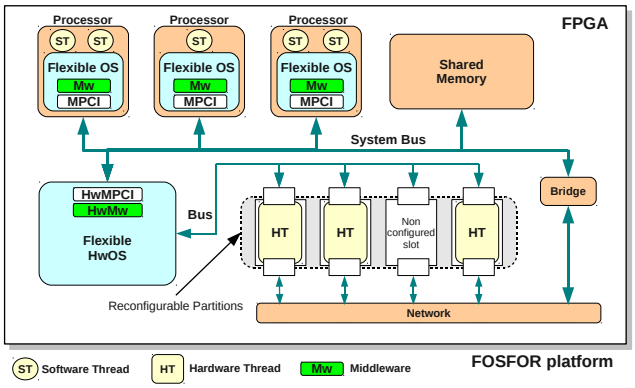

Figure 1. FOSFOR Architecture

We specified and implemented a HRSoC platform in the FOSFOR project [13] *(Flexible Operating System FOr Reconfigurable platform)*. This project aimed to define a reconfigurable multicore heterogeneous platform *(Fig.1)*. In the case of this project, hardware threads are managed by a hardware operating system *(Flexible HwOS)* implementing similar services in hardware to those offered by the operating system, such as thread management, semaphore counters or mailboxes.

A. Hardware Thread Model

The hardware thread *(HT)* we implemented in the FOSFOR project consists in two independent zones *(Fig.2)*. A static zone, corresponding to the interface between the thread and the rest of the system. Through this interface, the thread can communicate with the HwOS and so with the other threads. The OS interface is a memory accessible at the same time by the thread and by the HwOS. A last interface allows the thread to take advantage of multicore platforms allowing it to access to a dedicated network. For this project, a custom network-on-chip has been developed to enhance memory accesses [14].

The dynamic zone consists of a logical user function *(ie. the accelerator)*, and a finite state machine *(FSM)* to sequence system calls. This user FSM is controlled by a static system FSM, able to react to HwOS commands *(start, suspend or stop commands)*. Separation between system

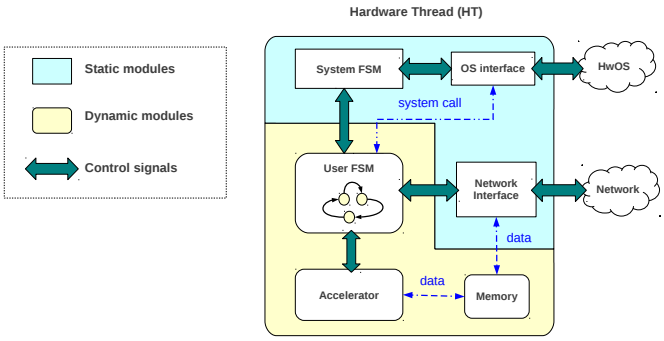

Figure 2. Hardware Thread (HT) Architecture

and user zones is done in order to plan hardware thread preemption management. Namely, our objective is to allow the operating system to relocate a hardware thread in any available slot of the platform.

III. Hardware Thread Management

An operating system managing hardware threads would rely on three services which are, a context management service, a reconfiguration service and a relocation service.

A. Context Management Service

There are two ways to save the context of a hardware thread. Either using check-pointing mechanisms [15] or processing partial readback [16]. The first one is intrusive and implies that the developer inserts checkpoints in his source code. Checkpoints are the only moments where the preemption is enabled. To preempt a thread, the scheduler has to wait that the thread reaches a checkpoint and so saves its context. Consequences are a time overhead at each checkpoints and latency in preemption decision. The advantage is that the context size could be dramatically reduced, and ideally to zero.

The second way is technology dependent but it avoids real time failure since preemption could be done immediately without risk to lose information. Readback consists in reading the contents of the partial zone where the module is located. Segregation between static part and dynamic part inside hardware thread permits task context reduction and offers a common interface in order to integrate different accelerators in the same partition.

B. Reconfiguration Service

A design using partial dynamic reconfiguration, as the one provided by Xilinx FPGAs [17], is composed of a static part and defined reconfigurable zones in which reconfigurable modules can be loaded. Using this technology, the operating system is able to schedule hardware threads [18], without resetting the rest of the system. For real-time applications, both readback and reconfiguration overheads must be minimized using a dedicated hardware reconfiguration controller,

such as FaRM [19], Uparc [20] or the solution offered by Koch et al. [21]. For instance, FaRM which is used in the design test detailed later allow to process configuration with a throughput of 400 MB/s.

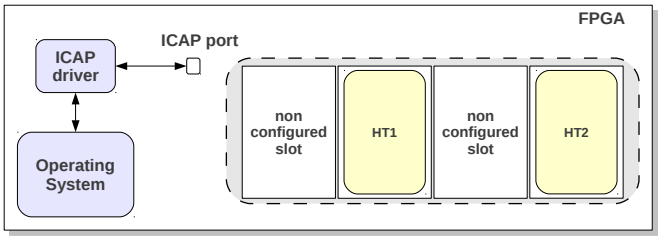

Figure 3. ICAP for Partial Reconfiguration

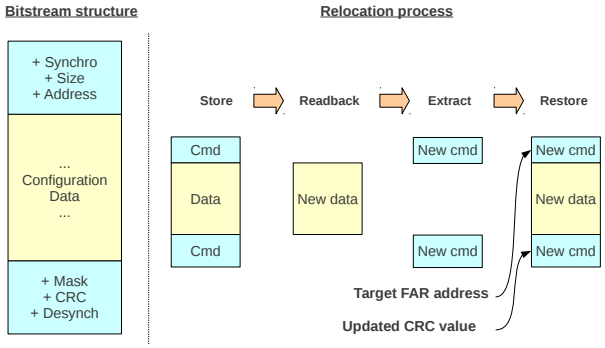

Figure 4. Partial bitstream relocation process

C. Relocation Service

As logic resources are critical in FPGAs, we would want to be able to run several threads in the same reconfigurable slot. One of the issue encountered in the classical flow is that a partial bitstream for a given module is generated for one slot and only one. To load a module on another slot, we need either another bitstream, which is memory consuming, or a relocated bitstream, whose creation is time consuming. In embedded system, with the increase of the FPGAs size, and so of the bitstream size, the amount of memory needed to store one specific partial bitstream for each targeted partition is becoming more and more prohibitive. This is why a relocation service seems to be the best choice. To relocate a partial bitstream, we implemented two services: a bitstream parser and a bitstream relocater.

A bitstream parser is needed to find the right information in the bitstream. Xilinx FPGAs are organized in rows and columns. Each column is composed of several frames, which is the smallest reconfigurable entity. To reconfigure a FPGA, the ICAP reads a bitstream, writes address information in the Frame Address Register *(FAR)* of the ICAP and writes frames contents into FPGA memory. Information which interests us in the bitstream is the FAR values and the CRC value. The process to relocate a partial bitstream is detailed in *Fig. 4*.

This process needs two bitstreams, one for the source partition and the other implemented for the target partition. A readback is done on the first partition. The resulting context is then saved in a new bitstream. The headers and footers of the second bitstream are then modified to target the wanted partition modifying the FAR and adding a newly computed CRC values. Finally, merging the headers and the saved context, we get a new relocated bitstream.

D. Interconnect implementation issues

In order to be able to safely relocate a dynamic module, the two dynamic reconfigurable partitions have both to be connected to the static partition in the same relative way. A signal going from the static partition to the reconfigurable module has to pass through a predefined route and to follow the same route in each reconfigurable partition. If it is not the case, hardware failures could occur and permanently damage the chip while relocating a bitstream. In order to do so, we want to have the ability to add some constraints on the inputs and the outputs of the LUTs used to interconnect the static and dynamic partitions, forming what is commonly called a bus macro. To achieve this goal, we designed hard macros using the XDL language [22]. This language allows to textually describe the route used by each nets in a component or even in a full design.

Another issue in thread relocation management is the routing conflicts between the static and the dynamic part. In the current provider tools, the static routing cross over each dynamic partition in different ways. To prevent the overlapping of this static routing, the routing and logic resources contained in these partitions have to be prohibited to the static partition. To achieve it, it is necessary to insert additional constraints. These constraints can be added using the Isolation Design Flow.

IV. Design Tools Flow

The Isolation Design Flow *(IDF)*, alternatively called Secure Chip Crypto *(SCC)* design flow, has been created to target fault-tolerant systems, especially in the critical applications in which safety and fault containment is a primary objective. This flow allows a designer to isolate the different modules of his system against each other. This is done regarding both the logic and the routing resources.

In this flow, each module to isolate is defined and synthesized separately. A top-level module groups all these modules as black boxes. To ensure a correct isolation, the implementation of these modules is done under some constraints. Namely, every connections between two isolated partitions have to pass through trusted routes *(Fig. 5)*. A trusted route specifies that an output of a partition has to pass through a direct route to reach another partition. If the output is used as a load for two different inputs, this signal

have to be split into two different signals passing through a LUT resource, and so forms what is called a trusted route. These constraints have to be applied to every inter-partitions signals when it is necessary except for the global signals such as the clock or the reset.

Moreover, in each isolated module, inputs and outputs which are not directly connected to an input or an output pad has to be defined as a non-buffered port as follow:

attribute buffer_type: string;
attribute buffer_type of <port_name> : signal is "none";

In the case of the relocation where routes between the static partition and the dynamic ones have to be relatively identical, we instantiated hard macros to connect these two types of partition *(Fig. 6)*.

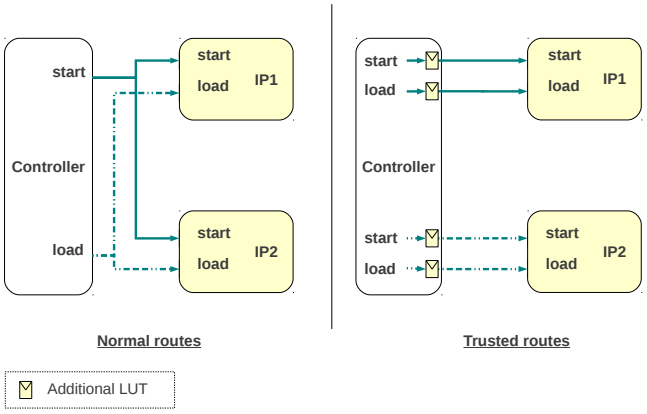

Figure 5. Trusted routes

Once all modules are synthesized, the main part of the flow is done using the PlanAhead tool. Modules netlists are imported in the design and these which need to be isolated are converted into partitions. Each partition is configured with the SCC_ISOLATION attribute, which notifies that the partitions have to be designed using the Isolation Design Flow. Then the physical block of each module is placed inside the FPGA matrix. Another constraint imposed by the Isolation Design Flow is that the input and output pads used by a partition have to be included inside the region covered by its corresponding physical block. In addition, the boundary between two isolated partition have to be of at least one CLB-wide, horizontally or vertically. This boundary is called a *Fence* and is an area in which neither the logic resources nor the routing switch matrices will be used. The following location constraints applied to the hard macros have to be inserted in an external constraint file and passed to XST using the *-uc* flag:

INST <hard_macro_name> LOC = SLICE_X#Y#;

where "#" represents valid Slice X and Y coordinates. This flag ensures that the synthesizer will respect the location constraints and that the hard macro will be placed at the correct position, over the static and the dynamic boundary. Finally, once the design is placed and routed, the correct isolation of each partition can be checked with the help of the Isolation Verification Tool *(IVT)* [12].

V. Implementation

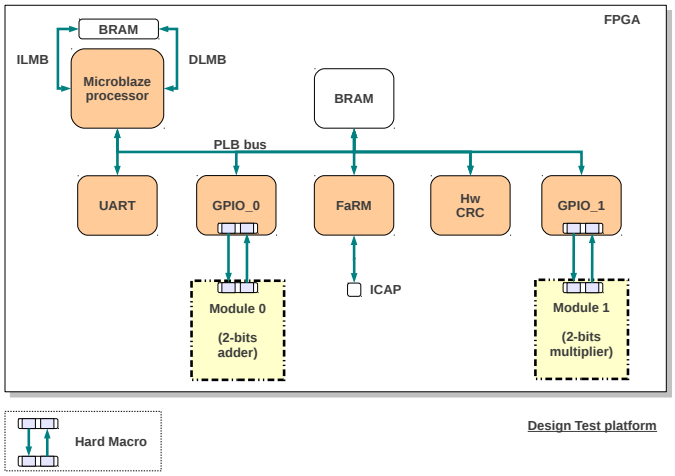

Figure 6. Test design

In a first step, we experienced the implementation of relocatable hardware module using the Isolation Design Flow on the simple design illustrated in *Fig. 6*, and implemented on a Virtex 5 SX50T FPGA using the version 13.1 of IDS[2]. It is a Microblaze-based platform composed of the FaRM IP used to reconfigure the dynamic partitions, a hardware CRC module used to compute the new CRC of the relocated module as well as two dynamic modules. There is no external memory. The only off-chip connections are the FPGA clock and the reset button. The two reconfigurable modules implement respectively a two-bits adder and a two-bits multiplier. Each one of these modules is controlled by the processor through a dedicated GPIO peripheral.

As trusted routes are direct routes from one partition to another, the number of available paths to route a signal is limited. To simplify the routing of the hard macros, we first implement "soft macros" instantiating LUTs directly in the top level module for one of the partitions. After the place and route phase, hard macros are extracted from the design netlist using using the RapidSmith framework [9]. Finally, the extracted hard macro is applied on each partition and the design is implemented once again.

After implementation *(Fig. 7)*, the two modules are well isolated in terms of logic and routing resources, and the one CLB-wide boundary between the dynamic modules and

[2]ISE Design Suite

the static partition is respected. This result permitted us to perform a safe relocation of these two modules in the a

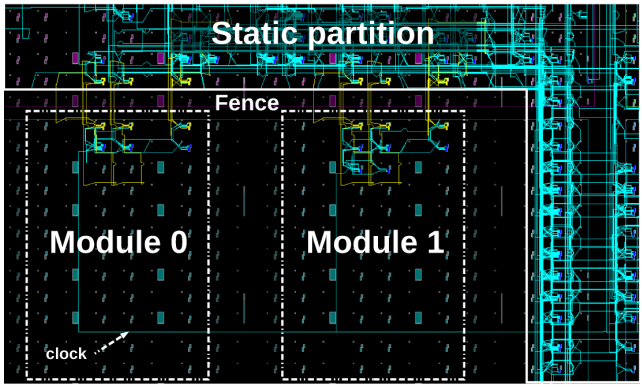

Figure 7. Design test - Partition isolation

VI. Conclusions and Future Work

We implemented a design in which two identical partitions have been created. The design flow based on the Isolation Design Flow allowed us to ensure that the relocation process can be executed without damaging the FPGA device. Moreover, this design flow relies on additional tools and routing techniques provided by Xilinx which is non-intrusive regarding to custom tools. For the moment, we target only partitions which offers the same set of resources but as FPGAs size is increasing continuously, as shown by the presence of super logic region *(SRL)* in the latest Virtex 7 FPGAs, finding identical areas in a FPGA will become less and less difficult. The simple example design implemented in this article has been used as a proof of concept. We are currently applying this design flow on the FOSFOR platform in order to relocate hardware threads in a real and more complex application. In future work, as introduced in [23], implementing and comparing a set of bus macros respecting in our case the IDF constraints, will help us to evaluate more precisely the usability and the interest of this design flow. In addition, we aim to integrate this work with previous work about on-line scheduling and placement algorithms in order to provide an efficient thread management service to heterogeneous reconfigurable systems-on-chips.

Acknowledgment

We would like to acknowledge Xilinx for its support and the access to the Isolation Design Flow.

References

[1] E. Anderson, W. Peck, J. Stevens, J. Agron, F. Baijot, S. Warn, and D. Andrews, "Supporting high level language semantics within hardware resident threads," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, 27-29 2007, pp. 98 –103.

[2] E. Lubbers and M. Platzner, "A portable abstraction layer for hardware threads," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, 8-10 2008, pp. 17 –22.

[3] Y. Wang, W.-N. Chen, X.-W. Wang, H.-J. You, and C.-L. Peng, "The hardware thread interface design and adaptation on dynamically reconfigurable soc," in *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, 25-27 2009, pp. 173 –178.

[4] M. Hübner, C. Schuck, M. Kühnle, and J. Becker, "New 2-dimensional partial dynamic reconfiguration techniques for real-time adaptive microelectronic circuits," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, vol. 00, 2-3 2006, p. 6 pp.

[5] T. Becker, W. Luk, and P. Cheung, "Enhancing relocatability of partial bitstreams for run-time reconfiguration," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, 23-25 2007, pp. 35 –44.

[6] J. Carver, R. Pittman, and A. Forin, "Relocation and automatic floor-planning of fpga partial reconfiguration bitstreams," *Microsoft Research, WA, Tech. Rep. MSR-TR-2008-111*, aug. 2008.

[7] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, aug. 2009, pp. 498 –502.

[8] C. Rossmeissl, A. Sreeramareddy, and A. Akoglu, "Partial bitstream 2-d core relocation for reconfigurable architectures," in *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, july 2009, pp. 98 –105.

[9] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *Proceedings of the 21th International Workshop on Field-Programmable Logic and Applications (FPL'11)*, September 2011.

[10] A. Sohanghpurwala, P. Athanas, T. Frangieh, and A. Wood, "OpenPR: An Open-Source Partial-Reconfiguration Toolkit for Xilinx FPGAs," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 228 –235.

[11] D. Koch, C. Beckhoff, and J. Torrison, "Fine-Grained Partial Runtime Reconfiguration on Virtex-5 FPGAs," in *Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM 2010)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 69–72. [Online]. Available: http://dx.doi.org/10.1109/FCCM.2010.19

[12] J. D. Corbett, "Xilinx White Paper 412: The Xilinx Isolation Design Flow for Fault-Tolerant Systems," Jan. 2012.

[13] L. Gantel, A. Khiar, B. Miramond, A. Benkhelifa, F. Lemonnier, and L. Kessal, "Dataflow programming model for reconfigurable computing," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, june 2011, pp. 1 –8.

[14] L. Devaux, D. Chillet, S. Pillement, and D. Demigny, "Flexible communication support for dynamically reconfigurable fpgas," in *Programmable Logic, 2009. SPL. 5th Southern Conference on*, 1-3 2009, pp. 65 –70.

[15] C.-H. Huang and P.-A. Hsiung, "Software-controlled dynamically swappable hardware design in partially reconfigurable systems," *EURASIP J. Embedded Syst.*, vol. 2008, pp. 4:1–4:11, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1155/2008/231940

[16] T.-Y. Lee, C.-C. Hu, L.-W. Lai, and C.-C. Tsai, "Hardware Context-Switch Methodology for Dynamically Partially Reconfigurable Systems," *J. Inf. Sci. Eng.*, vol. 26, no. 4, pp. 1289–1305, 2010.

[17] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Invited paper: Enhanced architectures, design methodologies and cad tools for dynamic reconfiguration of xilinx fpgas," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, 28-30 2006, pp. 1 –6.

[18] I. Belaid, F. Muller, and M. Benjemaa, "Off-line placement of hardware tasks on fpga," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, aug. 2009, pp. 591 –595.

[19] F. Duhem, F. Muller, and P. Lorenzini, "FaRM: fast reconfiguration manager for reducing reconfiguration time overhead on FPGA," in *Proceedings of the 7th international conference on Reconfigurable computing: architectures, tools and applications*, ser. ARC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 253–260. [Online]. Available: http://dl.acm.org/citation.cfm?id=1987535.1987569

[20] R. Bonamy, H.-M. Pham, S. Pillement, and D. Chillet, "UPaRC, Ultra-fast power-aware reconfiguration controller," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 1373 –1378.

[21] S. Hansen, D. Koch, and J. Torresen, "High Speed Partial Run-Time Reconfiguration Using Enhanced ICAP Hard Macro," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, may 2011, pp. 174 –180.

[22] C. Beckhoff, D. Koch, and J. Torresen, "The Xilinx Design Language (XDL): Tutorial and use cases," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, june 2011, pp. 1 –8.

[23] M. Koester, W. Luk, J. Hagemeyer, M. Porrmann, and U. Ruckert, "Design Optimizations for Tiled Partially Reconfigurable Systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 6, pp. 1048 – 1061, june 2011.