

MULTICONTEXT DYNAMIC RECONFIGURATION AND REAL TIME PROBING ON A NOVEL MIXED SIGNAL PROGRAMMABLE DEVICE WITH ON- CHIP MICROPROCESSOR

Julio Faura¹, Juan Manuel Moreno², Miguel Angel Aguirre³, Phuoc van Duong⁴, and
Josep Maria Insenser¹

¹ SIDSA, PTM, C/ Isaac Newton 1, Tres Cantos 28760, Spain (faura@sidsa.es)

² Universitat Politècnica de Catalunya, c/ Gran Capità, s/n, edif C4, 08034 Barcelona, Spain

³ Universidad de Sevilla, GTE, Avda Reina Mercedes s/n, 41012 Sevilla, Spain

⁴ MIKRON GmbH, Am Söldermoos 17, 85399 Hallbergmoos, Germany

Abstract. In this paper we present a novel RAM-based field programmable mixed-signal integrated device consisting of a large granularity Field Programmable Gate Array (FPGA), a set of programmable and interconnectable analog cells, and a microprocessor core. Two configuration contexts are available, at least one of them mapped on the microprocessor memory space. The microprocessor can be used for partial (or complete) fast dynamic reconfiguration, to run general purpose user programs, and to probe in real time internal points of the analog and digital programmable hardware. The device can be partially or totally reconfigured *while* it is working by loading the new configuration (including initial states for FFs) in the non-active context without stopping operation, then transferring it to the active one in just one microprocessor write cycle.

1 Introduction

In the past years, the microelectronics industry has shown an increasing interest in flexibility and in-house programmability. FPGAs and field programmable microcontrollers have thrived due to this interest, and recently also analog field programmable arrays are also appearing [1], [2], [3]. Mixed signal field programmable arrays [4] and flexible microcontrollers with A/D, DAC and PWM interfaces try to take a step beyond: That of a field programmable *system*.

Within this framework we introduce the FIPSOC (Field Programmable System On Chip) prototyping and integration system. The goal is to bring together the benefits reported by field programmable gate arrays to a system integration level. A novel field programmable integrated circuit has been developed including a FPGA, a set of programmable analog cells, and a standard microcontroller. The power of this approach relies on the optimized communication between the three subsystems: The microprocessor can read the outputs of the programmable blocks of the FPGA as memory locations, the microprocessor bus can be directly interfaced to the FPGA routing channels, the digital side of the analog cells (DAC, ADC, comparators) can be directly connected to the digital blocks and to the microprocessor, and the

microprocessor can dynamically change the configuration of the digital and analog programmable hardware in terms of memory accesses.

The ability of the microprocessor to overwrite the configuration of the digital and analog cells makes it specially suitable to be used as a dynamic reconfiguration engine. To further enhance this behavior, the configuration cells are duplicated, and at least one of these cells are mapped on the microprocessor memory space. This way, the microprocessor can interact with the configuration memory while the device is working, without having to stop operation while the new configuration is being loaded.

The other great advantage of having an on-chip microcontroller is the possibility to use it as a development system. For this reason, the (digital) outputs of the programmable blocks have been mapped on the microprocessor memory space, so any output of any programmable block can be read in real time. Furthermore, any point of the analog subsystem can be dynamically routed to the internal ADC block, which is directly interfaced to the microcontroller. Therefore, the microcontroller can be used to probe in real time any point of the analog architecture, thus acting as an emulated digital oscilloscope, and any output of the logic programmable blocks, thus emulating a logic analyzer.

This paper is mainly focused on the enhancements that an on-chip microprocessor can bring when included in a field programmable device, especially for the multicontext dynamic reconfiguration mechanism and the real time probing capability. The presented material comes from worst case simulation results of a full custom implementation of the programmable cells. Area results are valid since they come from actual full custom implementations included in the first fabricated chip prototype.

2 System Description

Fig. 1 shows a block diagram of the FIPSOC device.

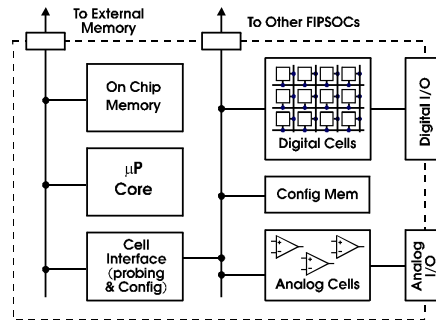


Fig.1. Block diagram of the FIPSOC chip

The chip is a mixed signal field programmable device with an on-chip microcontroller. It includes a Field Programmable Gate Array (FPGA), a set of fixed-functionality yet configurable analog cells, and a microprocessor core with RAM memory and some peripherals. The programmable digital and analog blocks

are well defined and are separated from one another due to noise immunity considerations. Nevertheless, the different interfaces between these blocks themselves and to the microprocessor provide a very powerful interaction between software, digital hardware and analog hardware. A whole family of FIPSOC devices has been planned, with different number of programmable cells in each family member.

The FIPSOC chip includes a two-dimensional array of programmable DMCs (Digital Macro Cell). The DMC is a large granularity, Look Up Table (LUT) based, synthesis targeted 4-bit wide programmable cell, whose structure is depicted in fig. 2. The combinational and sequential outputs of the DMCs are mapped as memory locations in the microprocessor address space. The microprocessor can then *probe* in real time actual digital signals just by issuing memory read commands to these memory locations. Figure 3 outlines the programmable analog tile, a four analog channel section made out of fully balanced differential amplifiers, filters, comparators, and a flexible programmable ADC/DAC structure. More details about the programmable blocks included in the FIPSOC chip can be found in [5].

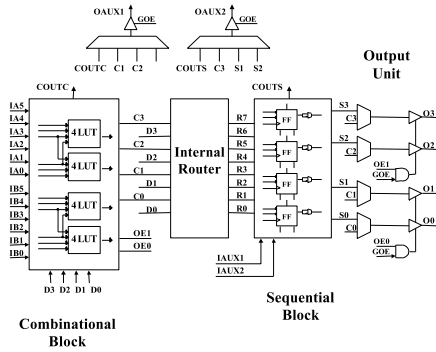


Fig.2. Simplified DMC block diagram

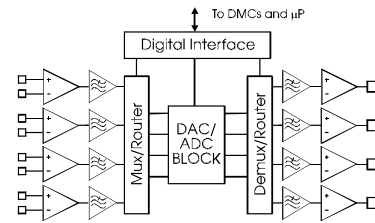


Fig.3. Analog block

In particular, it is worth noting that nearly any internal point of the analog block can be routed to the ADC. Then, the microprocessor can use the ADC to probe in real time nearly any internal signal of the analog structure by dynamically reconfiguring these analog routing resources. The ADC/DAC block is especially suitable for reconfigurable applications: it can be configured as one 10-bit DAC or ADC, two 9-bit DAC/ADCs, four 8-bit DAC/ADCs, or even one 9-bit DAC/ADC and two 8-bit DAC/ADCs at the same time. In the latter, two 8-bit DACs can be used to dynamically set the references for the 9-bit DAC/ADC, easily adjusting ranges and offsets on the fly.

As it has been said, the chip contains a standard 8051 microcontroller, which can be used either for general purpose user applications or for configuration management tasks. Apart from FPGA-microprocessor interface issues [6], most of the efforts done for on-chip integration of a microprocessor and programmable hardware have been targeted to enhance the processing power of the microprocessor [7] ,[8], by providing

reconfigurable coprocessors or customizable instruction sets, while we have focused a bit more on prototyping needs.

As the commercial device, the microcontroller contains not only a microprocessor core but a serial port (RS232), timers, parallel ports, etc. In particular the serial port is used to allow the chip to communicate to a PC, so the user can download the configuration and debug his applications using a simple monitor program running in the microprocessor and controlling it from the PC. The real time probing of the digital and analog programmable hardware is also done in terms of read data transferred to the PC via the serial port, though it could be also done in a parallel fashion.

3 Configuration Bits and Multicontext Dynamic Reconfiguration

In this section we describe how the configuration memory has been implemented and how the dynamic reconfiguration can be done in the FIPSOC chip. The study that follows is based on a full-custom implementation on real silicon of the. Data about memory stability, noise immunity, etc. has been extracted from SPICE simulations and validated afterwards in real silicon. Area results are valid since they come from actual full-custom implementations: a small array of DMCs have been included in the fabricated chip.

3.1 Microprocessor Access to the Configuration Memory.

As it has been already mentioned, the chip configuration is managed by the internal microprocessor, which can in turn be used as a dynamic reconfiguration engine. To do so, the configuration memory is organized in words which could be mapped onto the microprocessor address space. Such a system may have two important shortcomings: First, a weak memory cell used for hardware configuration would be prone to suffer from noise immunity problems, so the access and the use of these cells would be tricky; second, a large part of the memory space would be devoted to configuration, which makes inefficient the use of the microprocessor for general purpose user applications: Any user not willing to use the dynamic reconfiguration feature would have to cope with a large and critical part of the memory map devoted to configuration instead of user data or programs.

Mainly due to the first, it was clearly advisable to isolate the *actual* configuration cell from the interfaced memory. Fig. 4.A shows this concept: The actual configuration bit is separated from the mapped memory through a NMOS switch. This switch can be used to load the information coming from the memory bus onto the configuration cell. The microprocessor can only read and write the *mapped* memory, and it could only transfer the information in one way.

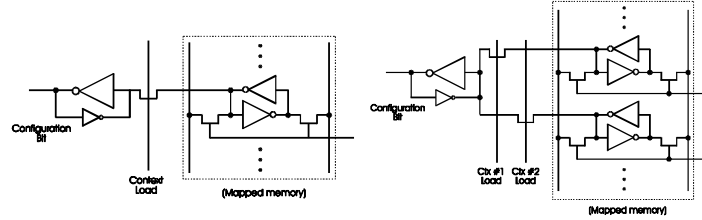


Fig.4.A (left) and 4.B (right). One (left) and two (right) *mapped* context and a *buffered* one.

This technique entails several benefits:

- The mapped memory can be used to run programs and to store general purpose RAM data. After configuration, which would be triggered by a *context load* command after downloading the configuration to the mapped memory, this memory would be free and could be used for something else.
- The configuration bit is isolated from the mapped memory cell. Therefore the memory cell is safer as its output does not have to directly drive the other side of the configuration bit. In our case, the routing architecture, which uses most of the configuration data, is made out of wide multiplexers, so the configuration signals cross with the routing channels and switches and therefore would be prone to be overwritten if they came from relatively weak memory cells.
- As long as these wide multiplexers have been implemented using NMOS switches, it becomes especially interesting to be able to drive them with a voltage level above the power supply. This implementation is especially suitable for this, because the power supply of the *actual* configuration bits can be easily separated from the rest of the circuit.

This implementation is said to have one *mapped* context (the one mapped on the microprocessor memory space) and one *buffered* context (the *actual* configuration memory which directly drives the configuration signals). There was also the possibility of having more than one mapped contexts to be transferred to the buffered context, as depicted in Fig. 4.B. However, as the number of mapped contexts increase, the efficiency of the proposed solution could decrease due to the bigger decoders needed to drive so much memory. The size of the DMC would of course increase, but more user memory would be available to the user.

A final possibility to increase the number of contexts would be to stack them, as depicted in Fig. 5. This technique consists of providing a greater number of contexts, maybe four or eight, but only having access to one of them. Data can be transferred both ways to effectively *push* and *pop* configurations onto the *configuration stack*. This way, a given application would have different smaller subsystems which could be treated as *hardware subroutines* [9]. These subroutines could be hierarchical and call more sub-subroutines, and so on. Everytime a subroutine is released, the actual DMC space upon which it was working would recover the task it was doing at a higher hierarchical level. Of course, the efficiency of this technique would greatly depend on the application and the detailed scheduling of hardware tasks. A further detailed study of this solution would be necessary to assure its feasibility.

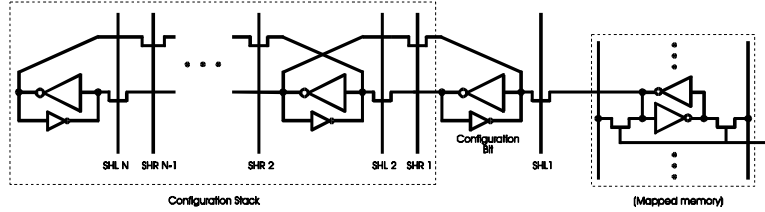


Fig. 5. A configuration stack structure

3.2 What has been implemented

As it has been said, the goal is to provide a family of devices. Two full custom implementations have been done, one as depicted in Fig. 4.A and another one as in Fig. 4.B. The DMCs included in the first silicon implementation have two mapped contexts and a buffered one as depicted in Fig. 4.B. Area results given in next section account for this implementation.

3.3 Economics

Table 1 shows an area breakdown of the DMC implemented in silicon:

Table 1. Area breakdown of a multicontext DMC

Function	Area percentage
Routing switches (NMOS multiplexers)	34
Buffered context	9
Mapped context	14
Extra mapped context	9
Four 4-LUTs	20
FFs and their control	4
Output buffers, microprocessor interface logic, etc	3
Internal interconnections and stray area	7

As it can be seen from the table, an extra context is not prohibitive. The area difference between the mapped and the extra mapped context comes from the decoder and its interface to the microprocessor bus - the same decoder is used for both contexts, while it is counted only once in the breakdown. The buffered context is bigger than the mapped one because it uses a larger gate width to be 5V compatible (this is interesting since the routing resources extensively use NMOS multiplexers), and because of noise immunity considerations. This buffered context should not be completely regarded as an “extra” option to configuration: It provides two polarity signals to the NMOS multiplexers, which would be complicated to route from the mapped memory blocks (which is evenly organized).

It is also important to note that the mapped context, after the configuration load, can be freely used for general purpose use. The memory is distributed along the DMCs, but the microprocessor can use it normally to run programs and to store user data. For a 16x16 DMC array, around 16 Kbytes of RAM memory is provided to the user.

The FFs are also internally duplicated to support the dynamic reconfiguration feature. The area percentage shown in the table accounts for two-context FFs.

An important point is the LUT configuration data. The area required by the LUTs includes the memory cells (which store the configuration), their connections and their interface. As far as this amount of area is somewhat large, the multicontext implementation of the LUTs must be studied separately.

3.4 The LUTs

Four LUTs have been included in the DMC. They have been implemented as 16x1 dual port memory blocks. One of these two ports is connected to the microprocessor interface, while the other one can be explicitly connected to the DMC inputs and outputs - to use the LUT as an actual user memory block with read and write access -, or implicitly connected to perform custom boolean functions - no user write access provided here.

Every LUT has been designed to be used as a single 16-bit LUT or as a two-context 8-bit LUT. A (multicontext) configuration bit decides the LUT mode (static or dynamically reconfigurable) and the context being used in each moment. As far as every two static 4-LUTs share two inputs (that makes a total of six inputs per pair of LUTs), two dynamically reconfigurable 3-LUTs are available in this mode, sharing no inputs. This way, no input pins are wasted, and while dynamic LUTs are smaller, thus supporting smaller functions, they share no inputs, increasing flexibility.

Exactly as the static 4-LUTs in a DMC can be combined to form two 5-input LUTs or one 6-input LUT, in the dynamically reconfigurable mode one can have four 3-LUTs, two 4-LUTs or one 5-input LUT. The rest of the functionality is the same. In particular, the 4-bit adder macro mode fits in the dynamically reconfigured mode, i.e., it can be performed with 3-LUTs.

3.5 Dynamic reconfiguration

An optimized interface between the microprocessor and the configuration memory, suitable for partial dynamic reconfiguration, is provided. Fig. 6 schematically shows how this interface works, not very different from existing ones [6].

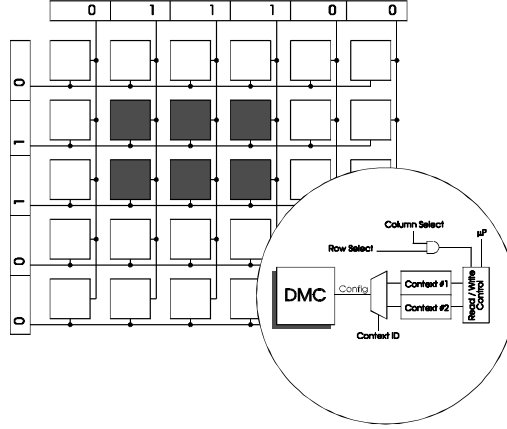


Fig. 6. Microprocessor interface for dynamic reconfiguration

A mask of columns and rows can be selected for any memory write operation, including a context load command. Therefore, any *logical rectangle* of DMCs can be selected for reconfiguration, so any write command from the microprocessor would apply to all members of the selected set at the same time. We define a logical rectangle as a set of DMCs (X,Y) such as if (X1,Y1) and (X2,Y2) are in the set, then (X1,Y2) and (X2,Y1) are also in. This structure is especially efficient for array based applications such as systolic multipliers, array-based FIR filters, etc.

For a DMC structure based on the configuration cell depicted in Fig. 4.A, dynamic reconfiguration would work as follows: Upon reset, the microprocessor would load the initial configuration on the mapped context, and the circuit would start working. Then, while the circuit is working, the microprocessor would fill the mapped context of the set of cells to be reconfigured, which does not interact with the active context (the buffered one) of these or any other cells. The initial states of the FFs in the new context could also be loaded without affecting the actual values stored in the active FFs. When the new configuration is loaded on the mapped context, a mask would be set to design the set of cells going to be reconfigured. Hitherto the circuit is still working with the old configuration. Then, a single write command is performed (the *context load* signal in Fig. 4.A is activated) and the selected part of the circuit starts working with the new configuration, including the desired initial states of the FFs.

Furthermore, the FFs can be used in single context or double context mode. In the single context mode the data is kept upon reconfiguration, just as global variables or pointed parameters in software programs and subroutines. In the double context mode data would be swapped away with the context change, and it could be swapped back upon a new reconfiguration. This second mode is especially useful for storing initial values before the reconfiguration takes place.

A DMC structure with two mapped contexts, as depicted in Fig. 4.B, goes one step forward in reconfiguration: The microprocessor can load two different configuration contexts upon reset, then dynamically change between them on the fly

with no extra reconfiguration time (just that of a microprocessor write cycle). If only two contexts are necessary, this approach would be enough for a number of applications.

A related work which should be mentioned here is the DPGA chip [10], a real multi-context device for programmable combinational functions based on dynamic RAM. Finally, it is worth noting how in [11] an interesting study on dynamic reconfiguration and its applications can be found. It can be seen how the improvements suggested there have been already implemented in this new system.

4 The FIPSOC as an Integrated Prototyping Workbench with Internal Probing Capability

Another interesting possibility comes from the optimized interface between the microprocessor and the programmable hardware itself, not its configuration memory as we have already studied. It is the possibility of probing in real time nearly any point of the analog or digital user application mapped onto the programmable hardware. In fact, it is possible to emulate a whole laboratory benchmark using just a FIPSOC Chip and a PC. The communication between them would normally be done through the on-chip RS232 serial port (an external driver is needed for RS232 voltage levels).

Logic data acquisition for this real time probing can be done in terms of memory read operations from the outputs of the DMCs, which are mapped as memory locations on the microprocessor memory space. LUTs configured as memories and dedicated DMCs configured as counters could also be used for fast logic data acquisition like in logic analyzers. Note that the microprocessor can look up data from the LUTs while the LUT is in operation.

As it has been said, the internal ADC block can be dynamically rewired to probe nearly any internal point of the analog architecture. An analog data acquisition system, emulating a digital oscilloscope, could then be done using the microcontroller or some dedicated DMCs, as far as the digital side of the ADC can be connected to the digital routing channels and can be directly interfaced to the microprocessor. Even the internal DAC could also be used as a function generator, accepting data from the microprocessor core or from some DMCs configured as counters and memories.

The rest of the laboratory workbench is a matter of software: A digital oscilloscope could be emulated to present the acquired analog data (from the internal ADC) on the PC screen; a logic state analyzer could be provided printing out the digital data obtained from the DMC outputs in real time. The real advantage is the integrated approach that the user can follow to develop his system application: Everything is integrated, and the final hardware and software solution is exactly what the user is measuring when probing in real time. The proposed analog programmable hardware is especially suitable for this operation. In particular, the possibility of severing the ADC/DAC block into a 9-bit ADC and two 8-bit DACs, and then using these two DACs to provide the ADC references, greatly simplifies the emulated digital oscilloscope described: the “offset” and “amplification factor” knobs would directly apply over the reference DACs.

5 CAD Tools for FIPSOC

An integrated set of software CAD tools is being developed for design entry and optimization, technology mapping, placement and routing, device programming, mixed-signal simulation and real-time system probing from a WindowsTM-based PC station.

A dynamic reconfiguration management tool, able to handle the multicontext operation of the chip, would constitute a very interesting research area here. Such a tool could analyse HDL code to check the coincidence in time of the processes, their criticality and their system requirements. Up to now, some experiences on dynamic reconfiguration software have already been reported [12].

The FIPSOC chip is especially suitable for hardware-software co-design techniques due to the flexible interfaces between the programmable hardware areas and the microprocessor core, which results in a very powerful hardware-software interaction. This interaction is enhanced mainly due to: *a)* Internal signals from the programmable hardware can be probed and read as memory locations from the microprocessor core (analog signals have to be converted with the ADC). *b)* The microprocessor can dynamically reprogram a piece of hardware by overwriting the configuration memory. A co-design CAD tool could then be targeted to this device, putting in hardware those critical processes needing too high a computational speed, and performing with software those tasks which would be prohibitively area-consuming.

We think that FIPSOC would be a suitable benchmark platform for this kind of tools and methodologies.

6 Conclusions

In this paper we describe some of the benefits of including an on-chip microprocessor within a mixed-signal field programmable device, in terms of enhanced dynamic reconfigurability and integrated system prototyping. A new dynamic reconfiguration technique, based on buffered contexts, is introduced. An actual silicon implementation of the reconfiguration model has already been done and shows the feasibility and effectiveness of this technique: An extra mapped context is not prohibitively large; the reusability of the configuration memory is maximized because the mapped context can be used for general purpose programs and data if no dynamic reconfiguration is desired; the buffered context is isolated from the weak memory cells, prone to be affected by the circuit noise; and the buffered context can be independently powered, which is interesting if voltage boosting is necessary when NMOS routing switches are used.

The optimized interface between the microprocessor and the actual signals mapped on the programmable hardware makes possible to provide a whole development system, including an emulated digital oscilloscope and logic analyzer, with just a FIPSOC chip and a host computer, namely a PC (a RS232 driver is also needed to convert the 3V chip signals to RS232 levels). This arises from the fact that the outputs of the programmable logic blocks are mapped on the microprocessor

memory space, and the internal points of the analog hardware can also be probed with the internal ADC/DAC block.

7 Acknowledgements

This work is being carried out under the ESPRIT project 21625. The authors would like to thank the European Commission for the financial support.

References

- 1 D. Anderson, C. Marcjan *et al*, "*A Field Programmable Analog Array and Its Application*", 1997 IEEE Custom Integrated Circuits Conference, Santa Clara (CA, USA).
- 2 Hans W. Klein, "*The EPAC architecture: an expert cell approach to field programmable analogue devices*", FPGA'96, Monterrey CA.
- 3 "*Analog-Silicon-Breadboard - an analogue FPGA*", Fraunhofer-Institut für Mikroelektronische Schaltungen und Systeme, IMS2 Dresden (commercial communication).
- 4 Chengjin Zhang, Adrian Bratt and Ian Macbeth, "*A New Field Programmable Mixed-Signal Array And Its Application*", The 4th Canadian Workshop on Field-Programmable Devices, 1996 Toronto, Canada.
- 5 Julio Faura, Chris Horton, Phuoc van Duong, Jordi Madrenas, Miguel A. Aguirre, and Josep M. Insenser "*A Novel Mixed Signal Programmable Device with On-Chip Microprocessor*", 1997 IEEE Custom Integrated Circuits Conference, Santa Clara (CA, USA).
- 6 Stephen Churcher, Tom Kean and Bill Wilkie, "*The XC6200 fastMapTM processor interface*", European FPL '95, Oxford (UK).
- 7 H.F. Silverman *et al*, "*Processor Reconfiguration Through Instruction-Set Metamorphosis*", IEEE Computer, 23 (3) March 1993.
- 8 Neil Hastie and Richard Cliff, "*The implementation of hardware subroutines on field programmable gate arrays*", 1990 IEEE Custom Integrated Circuits Conference.
- 9 Neil Hastie and Richard Cliff, "*The implementation of hardware subroutines on field programmable gate arrays*", IEEE 1990 CICC.
- 10 E. Tau, D. Chen, I. Eslick, J. Brown and A. DeHon, "*A First Generation DPGA Implementation*", FPD '95 -- Third Canadian Workshop of Field-Programmable Devices, 1995 Montreal, Canada.
- 11 Patrick Lysaght and John Dunlop, "*Dynamic reconfiguration of FPGAs*", European FPL '93, Oxford (UK), pp 82 to 94.
- 12 Patrick Lysaght and Jon Stockwood, "*A Simulation Tool for Dynamically Reconfigurable Field Programmable Gate Arrays*", IEEE transactions on VLSI systems, vol. 4, nr. 3, September 1996.