Fault-Tolerance in FPGA's through CRC Voting

Helano Castro
LESC - UFC
Campus do Pici, SN B 723
Fortaleza CE Brazil
(+55) 85 3366 9608
helano@lesc.ufc.br

Alexandre Coelho
LESC - UFC
Campus do Pici, SN B 723
Fortaleza CE Brazil
(+55) 85 3366 9608
alexandre@lesc.ufc.br

Jardel Silveira
LESC - UFC
Campus do Pici, SN B 723
Fortaleza CE Brazil
(+55) 85 3366 9608
jardel@lesc.ufc.br

ABSTRACT

The use of FPGA's for implementing fault-tolerant systems (FTS) has been widely discussed. Many FTS's have been proposed in this context and TMR is by far the most used architecture. However, those implementations have to count on the memory configuration's integrity of those FPGA's, since all the TMR's circuitry is stored into it. In fact, radiation or even electromagnetic noise can disturb the content of the configuration memory, with disastrous results for the system. In this paper we propose a way of dealing with this problem by using the FPGA's CRC as its signature. In case of an error derived from a kind of fault mentioned above, that signature will change. By voting those signatures in TRM architectures we can not only detect the faults, but we can recover from them by copying the memory configuration of a faultless FPGA into a faulty one. We discuss the difficulties of implementing this technique and the workarounds used to get over those difficulties. Finally we implement an experiment to validate the idea.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance.

General Terms

Measurement, Reliability, Experimentation, Theory, Design, Verification.

Keywords

Cyclic redundancy check, fault tolerance, FPGA, partial reconfiguration.

1. INTRODUCTION

Critical applications need very reliable computer systems (CS) in order to perform the specified functions should a fault occurs. This problem is somewhat more important because many of those applications have very strict timing requirements and because of that they are known as Real-Time Applications. In order to meet these requirements, the CS's must have reliability's levels very high. One way of enhancing the CS's reliability is to provide fault-tolerance, which can be achieved by hardware and software replication (temporal replication is also used).

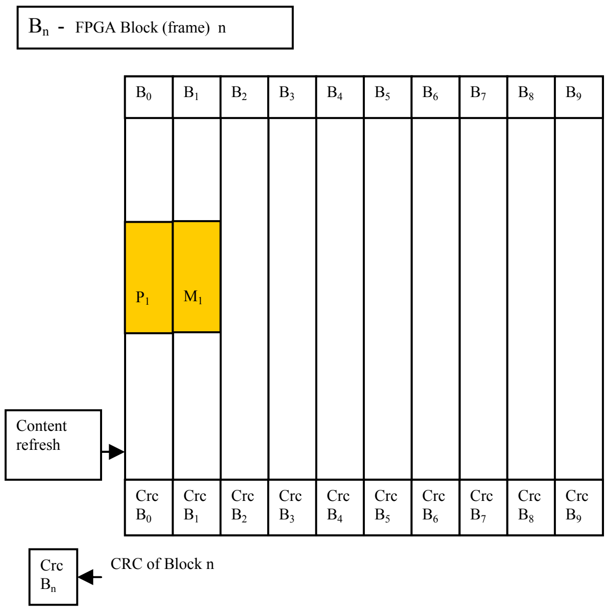
Hardware replication has become very attractive because its cost has decreased dramatically over the last decades, especially when compared to software's cost. However, this approach has its drawbacks. Nowadays, many applications are performed in a special class of systems known as Embedded Systems (ES). ES's have an embedded computer system that is used to help to carry out the actions needed by the application. As ES are getting smaller and smaller, the restrictions concerned with size, weight and power are getting very strict. As a result, redundancy has to be thought very carefully so that those restrictions are not violated.

On the other hand, the use of FPGA's in fault tolerant systems has been much appealing, particularly when use of reconfigurable FPGA's[1][2] is made. In fact, once is possible to replicate processing units inside a single FPGA, the restrictions aforementioned could have more easily met. However, the use of FPGA's brings another challenger to the designer. SRAM-based FPGA's[3] are very susceptible to radiation and electromagnetic noise, what can cause *single-event upsets* (SEU) in the *configuration RAM*[4]. The effects of such hazard on FPGS's are different from those occurring in RAM found in usual computer systems. On FPGA-based designs, such a fault can provoke a bit inversion on its configuration memory, with reflection on the circuit design implemented in it. Worse still, many modern FPGA's do not support random bitstreams, therefore a bit inversion could cause an internal contention, connecting directly different logic's levels (*0* and *1*). This problem has to be managed when implementing fault-tolerant architectures in a FPGA.

A much well-known fault-tolerant architecture is a TMR (Triple Modular Redundancy) setup. Takahara et al [5] have proposed a TRM that uses instances of Xilinx's *soft core* 32 bits RISC Microblaze, with the voter implemented in SDRAM. Carmichael at al [6][7] describes a technique that can be used to correctly implement a TMR in a single FPGA. Useful as those systems can be, the problem with faults aforementioned has still to be dealt. In this paper we propose a way of dealing with this problem by voting the CRC's of the redundant modules in the FPGA. We also describe the difficulties of implementing the idea on some FPGA's due to unavailable inside information from the manufacturers. Besides we propose and describe the implementation of a setup meant to replicate the problem and we validate the proposal through an experiment.

2. USING CRC FOR FAULT DETECTION IN FPGA

Figure 1 represents a 10-blocks (frames) FPGA where each block can be reprogrammed independently. Each block $\mathbf{B_n}$ corresponds to a FPGA`s programming unit. The block named *content refresh* rewrites periodically the content of each column $\mathbf{B_n}$, as well as it computes periodically its CRC (Cyclic Redundancy Code). The CRC value of block B_n is stored into a register, named ***CRC_B$_n$*** (where n = 0…9), associated with that block, as is shown in figure 1. Thus, each ***CRC_B$_n$*** contains the CRC of block B_n.



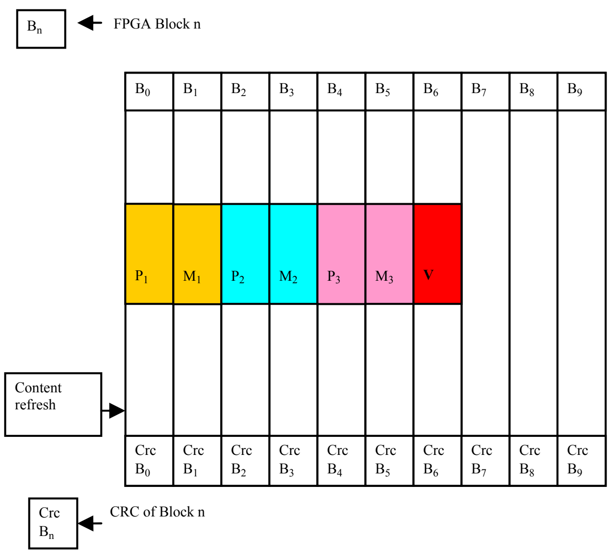
B_n - FPGA Block (frame) n
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
P_1
M_1
Content refresh
Crc B_0 Crc B_1 Crc B_2 Crc B_3 Crc B_4 Crc B_5 Crc B_6 Crc B_7 Crc B_8 Crc B_9
Crc B_n
CRC of Block n

Figure 1: A Simplex system implemented in a FPGA.

Now suppose we wish to implement a computer, hereafter named $\mathbf{C_1}$, on that FPGA, and that two blocks are used for that purpose: $\mathbf{P_1}$ is used to implement the processor, and $\mathbf{M_1}$ is used to implement the memory. As it can be seen in figure 1, $\mathbf{B_0}$ contains $\mathbf{P_1}$, and $\mathbf{B_1}$ contains $\mathbf{M_1}$. Correspondently, **CRC_B0** contains $\mathbf{P_1}$'s CRC and **CRC_B$_1$** contains $\mathbf{M_1}$'s CRC. Therefore $\mathbf{CRC_0}$ represents the CRC of the bits that make up **P1**, while $\mathbf{CRC_1}$ represents the CRC of bits that make up $\mathbf{M_1}$. As $\mathbf{C_1}$ has only one $\mathbf{P_1}$ and one $\mathbf{M_1}$, we call it a Simplex System**.**

Suppose now that one decides (e.g. in order to increase the fault tolerance of the simplex system $\mathbf{C_1}$) to implement a TRM (Triple Redundancy Module) by replicating $\mathbf{C_1}$ in the same FPGA shown in figure 1. Figure 2 shows that scenario, where $\mathbf{C_2}$ (composed by $\mathbf{P_2}$ and $\mathbf{M_2}$) and $\mathbf{C_3}$ (composed by $\mathbf{P_3}$ and $\mathbf{M_3}$) were added to the original project. Note that all units (**P**'s and **M**'s) were implemented in disjoint blocks (that restriction will be raised later). A Voter **V,** used to vote the results produced by the redundant modules, can also be seen in this figure. As the CRC of one $\mathbf{P_n}$ processor can change while the CRC of another one is being read, the voter should stop the clock of all processors $\mathbf{P_n}$, calculate the CRC of each $\mathbf{P_n}$, and carry out the voting process. In this paper, we propose to use a Xilinx Virtex II Pro FPGA in order to use, as a voter, the softcore MicroBlaze Processor. As explained further on, that would make it easier to implement our proposal.

The bits used to implement each $\mathbf{C_n}$ block in the TRM system, before it is placed into use, are called the *hardware context*. Usually, the hardware context should not change, unless a fault should occur. In this case, a change in its pattern related to that context would occur. Since this pattern is static, we call it hardware context. When the FPGA is powered and the computers **C**'s start executing, the computational state of each **C** changes as a result of its register's values being modified. This change of state resulting of the computation along the time, we call it *software context*, since the pattern of the bits that represents this context is dynamic.



B_n
FPGA Block n
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9
P_1 M_1 P_2 M_2 P_3 M_3 V
Content refresh
Crc B_0 Crc B_1 Crc B_2 Crc B_3 Crc B_4 Crc B_5 Crc B_6 Crc B_7 Crc B_8 Crc B_9
Crc B_n
CRC of Block n

Figure 2: A TMR system implemented in a FPGA.

The CRC of each block $\mathbf{B_n}$ would change as a result of both changes in the hardware and software contexts. In another words, CRC changes would result either due to faults or due the evolution of the computation in each **C**. In a faultless scenario, the CRC of each computer $\mathbf{C_n}$ (which is a combination of the CRC of $\mathbf{P_n}$ and $\mathbf{M_n}$) should be the same in each $\mathbf{C_n}$ that make up the TMR system. Should any **C** module present a different CRC than the other two modules, this could indicate the presence of a fault in that **C** module. In our proposal, each module should send the result of its computation to the voter (in our case, the MicroBlaze). The Microblaze[8][9][10] stops the processors' clock and calculates the CRC of each **C** module. The reason for stopping the clock is to make sure that the vote acquires coherent CRCs, in relation to each **C** module. After that, the Microblaze carries out the voting process. However, the voting is taken place not on the results furnished by the computers **C**'s, but on the CRC's of those computers. As explained before, those CRC's are acquired by the Microblaze from the blocks' **CRC_Bn's**. If those values happen to be the same, it means that not only no fault occurred with any C hardware (because the hardware context did not change), but the results of the computation for all three modules are the same as well (software status did not change). In that case, the voter should use as the result of the TMR computation any module's result. Should one **C** module produce a different result, the majority vote would be the value used by the Voter as the TMR's result of the voting process, and the result of the agreeing modules (with the same CRC's) should be used as the result of the computation. As the FPGA has two MicroBlaze processors, redundancy at the Voter's level would be possible. Figure 3 shows the voting process.

The CRC Fault Detection approach is able to detect any kind of fault, both those occurring in the computer's core and those affecting its computation. In addition, this approach makes it possible to tell if the problem occurred either in the block $\mathbf{M_n}$ or block $\mathbf{P_n}$ of the processor $\mathbf{C_n}$, providing that these two units are implement in disjoint blocks $\mathbf{B_n}$**'s**. In case this is not necessary, we can alleviate the initial restriction for these units to be implemented in disjoint blocks.

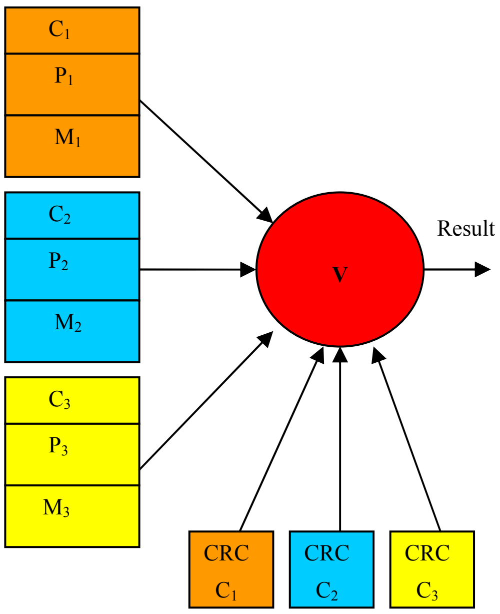


Figure 3: CRC's Voting Approach.

In order to implement our proposal, some obstacles have to be removed. In this session we will explain each one of them as well as the way to work around the problems. As we are working with Xilinx Virtex II Pro[11] FPGAs, we will describe the difficulties of implementing our proposal on them. In the first place, Xilinx does not recommend read-back operations on memory implementations in its FPGA's. As a result, that approach should be used with reservation on modules **M**'s of figure 2. As a work-around on this issue, a CRC calculation on the contents of those blocks could be made, as usually is done with memory contents. The second obstacle is that the designer has to make sure that each module fits in its block, with no external connections been routed through it. Xilinx routing tool provided to developers does not obey restrictions related to areas where the signals are routed. SEDCOLE et al [12][13] made use of a routing tool that does that job, but that tool was designed to be used only by Xilinx designers or people authorized by them. Thus, although it may take some effort it is possible to guarantee that restriction, especially if one can use the tool used by SEDCOLE.

As it can be seen in figure 2, the CRC needed to be generated is that associated with modules $\mathbf{P_n}$ and $\mathbf{M_n}$ (for each computer $\mathbf{C_n}$). However, those modules may not use the whole block (column) size where it is located (these block are referred usually as *frames*). Thus, in order to calculate the CRC of each module we need to use only the bits in the frame that represent its implementation. There lies the third problem with implementing this approach on one-only FPGA. Xilinx does not provide information on the meaning of each bit in the configuration frame for Virtex-II/Virtex-II PRO family. Upegui in (UPEGUI; SANCHEZ, 2005)[14] gives some details on the organization of these bits within a frame, allowing to identify the bits which describe the equation implemented by a LUT. However, more details concerning the rest of this bit stream (routing and other FPGA configuration's resources) is not disclosed by Xilinx.

For all these reasons we were not able to test the idea on a setup containing one-only FPGA, but it should be clear that, providing we have the pieces of information and tools above mentioned, we should be able to do so. As a result we decided to test the idea on a similar configuration that strives to work around the problems aforementioned. This setup is described in the next session.

3. AN EXPERIMENT TO VALIDATE THE PROPOSAL

In order to reproduce the environment needed to test our proposal, we used on our experiment three XUP-V2Pro boards [15][16], each one corresponding to a computer **C** in the TMR architecture, as shown in figure 4. A personal computer (PC) is connected to each board through RS232 serial ports, whereby the CRC of each computer in the TMR architecture is collected by an application being executed in the PC, which acts as the Voter. The CRC associated with the bit stream of each computer in the FPGA is read via a *readback* operation[17].

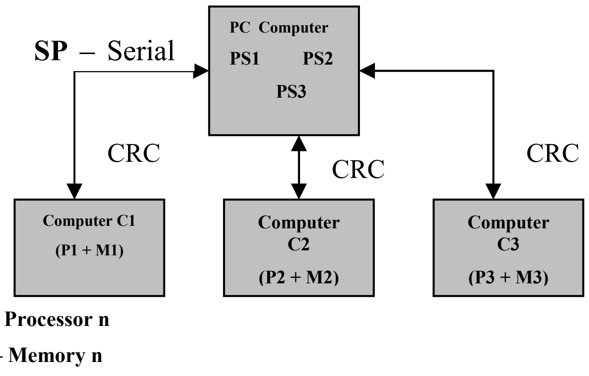


Figure 4: Setup of our experiment. A TMR System with PC Computer acting as Voter.

In each FPGA we embedded a MicroBlaze™ soft processor, developed by Xilinx and made it available in the Software Xilinx

Embedded Development Kit (EDK)[18]. The MicroBlaze has the architecture shown in figure 5.

When the CRCs of all FPGAs are received by the voter application, it votes these results. Should one CRC be in discordance with the others, it is understood as the computer corresponding to that FPGA being faulty. When a fault on a computer is detected via CRC comparison, the voter application automatically discards its computation. In addition to isolating the faulty module, as part of the fault-tolerance recovery action, it could copy both hardware and software bit stream of any faultless FPGA into the faulty one. After that reconfiguration process the Voter could try to use the recovered module in the next cycle. Figure 6 shows that process.

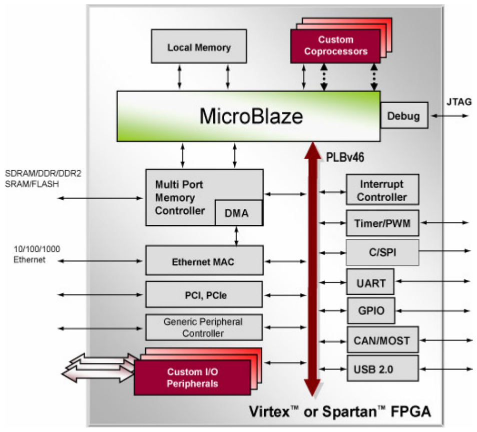


Figure 5: Soft Core MicroBlazer Processor.

Event though the fault occurring in the FPGA does not turn into an error in the computation (i.e. the faulty area was not accessed yet), the fault is detected, since the change in the bits that implement the faulty logic circuit will be reflected on the FPGA's CRC read by a readback operation.

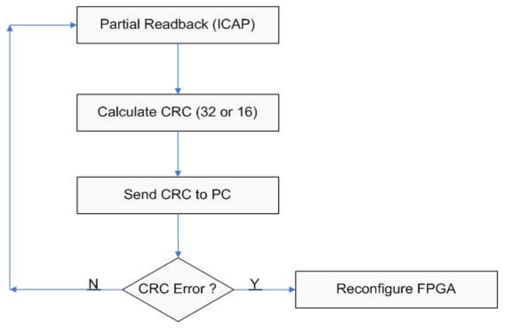


Figure 6: Voter compare the bit stream's CRC of the modules.

The partial FPGA reconfiguration prior to Virtex II/Virtex-IIPro families was realized via the SelectMAP[19] interface or via the JTAG. However, that requires an external controller to send the reconfiguration data. Virtex II/Virtex-IIPro families have an internal interface known as ICAP . ICAP is a reduced set of SelectMAP, but with the advantage of having distinct input and output ports for data, whereas SelectMAP has one bidirectional port.

The Xilinx's EDK (*Embedded Development Kit)* made it easier to use ICAP by integrating a device named HWICAP, which allows the operation of ICAP via a MicroBlaze softcore processor. HWICAP is connected to the OPB bus (a proprietary Xilinx bus) [21] and has a control logic and a small cache memory for configuration implemented in a BRAM. The EDK makes available an API [21][22] that can be used to program these devices and that define methods for transfering data between the configuration cache memory and the configuration memory. That facility allows one to perform readback operations, frame by frame (block by block), as well as partial FPGA's reconfigurations. The process running in the PC/Voter was developed by using the ICAP Reconfiguration's API.

Figure 7 shows a block diagram representing each implementation of a computer module in our TMR architecture. Note that we can choose to perform the voting process into the internal MicroBlaze instead of using an external computer (in fact, as a way of increasing the reliability, we could even replicate the MicroBlaze into the FPGA). However, in order to keep firmer control of the experiment, we decide to use the arrangement shown in figure 4.

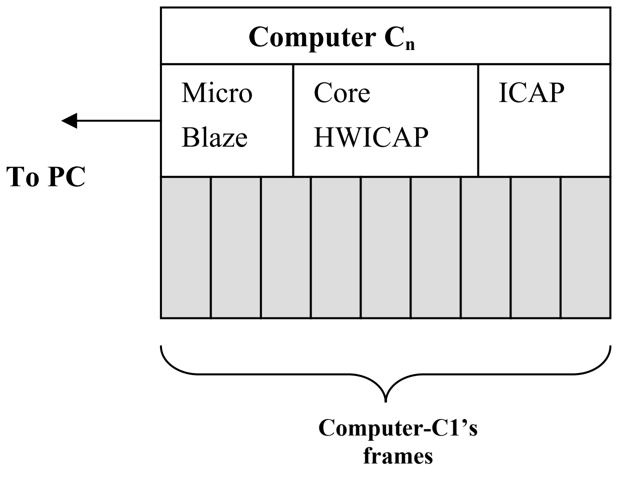


Figure 7: Each FPGA sends its bit streams processed by ICAP via its internal softcore Microblaze.

4. CONCLUSION AND FUTURE WORK

The use of CRC as a signature of FPGA's healthiness is a very interesting approach. In fact, when this technique is used on a TMR, a change in any FPGA's CRC will indicate both a fault occurring in the memory configuration that implements the system's circuitry and a fault in the computation being carry out by it. As a result, we can use this approach to monitor the status of each module in the TMR. Besides, as the FPGA's configuration memory contains the present status of not only the circuitry but also the status of the computation, a recovery action can be performed by copying the configuration memory contend of a faultless module into a faulty one. Thus this technique can be used with the system on the fly. While some techniques count on the replication of each part of a module and on voting the results of those parts, as part of the computation, our approach makes the whole fault detection mechanism transparent to the application being performed. This contributes to separate what is application to what is fault tolerance in the system.

As Xilinx does not openly provide information about each bit in each frame configuration of the Virtex-II/Virtex-II PRO FPGA's family, we had to test the proposal by implementing the TMR in different FPGA's. However, the results of our experiment prove that, providing we have those pieces of information we can use it on a one-only implementation. In addition to that, as the routing tools do not obey user's restrictions concerning the way the signals must be routed (which is important to guarantee that the components of each redundant computer is in different frames), we had also to work around that limitation. SEDCOLE [13] used successfully a tool that meets those restrictions. However, this tool is for Xilinx's internal use only and it is not available for outside users. However, this problem can also be worked around providing we have that tool.

The problems mentioned above, of course, do not prevent one from implementing the technique in a setup consisting of only one FPGA.. Our work proved that the technique works and, should we obtain the information above, we could easily use the technique in a one-only FPGA. As a next step on our research we will strive to implement this technique in such a configuration. We also intend to perform the CRC calculation into a specific hardware (VHDL) that will read the CRC via ICAP and will perform the calculation. We believe that CRC calculation will be performed faster, as well as we think that less space will be needed in the FPGA, since the calculation will be performed by the softcore microblaze microprocessor.

It should be said that the approach described in this paper is only a piece of a larger research project which intends to deal with hardware's resources management in the context of fault tolerant reconfigurable systems (mainly by using FPGA's). The way the Voter is handled is an issue that belongs to that scope and it will be discussed in another paper to be published soon. We are also conceiving several set-ups that will measure the latency in the voting process. That, of course, will depend on the way the hardware management deals with the detection and recovery steps of the fault tolerance process. Finally, although we had not provided a formal comparison of our approach with the ones used by other authors, we can say that our approach is superior to the ones quoted in this paper, in the sense that it can be used as a general approach, since it count only on the CRC of each module (regardless if it is a processor or other digital circuit). That will be discussed more extensively in another paper.

5. REFERENCES

[1] Javier Castilho, Ivan González, Pablo Huerta, J.I. Martinez, "Auto-Reconfiguración sobre FPGAS".

[2] Blodget B., James-Roxby P., Keller E., McMillan S., Sundararajan P. A, "Selfreconfiguring Platform", FPL'03, pp. 565- 574, Sept 2003.

[3] M. G. Gericota, G. Alves, M. L. Silva, J. M. Ferreira, "Active Replication: Towards a Truly SRAM-based FPGA On-Line Concurrent Testing", *Proc. IOLTW*, pp. 165-169, 2002.

[4] KASTENSMIDT, F. L. ; REIS, Ricardo . Designing Single Event Upset Mitigation Techniques for Large SRAM-based FPGAs. Porto Algre: PPGC, 2003.

[5] TAKAHARA, T. et al. Embedded computer system with soft core cpu for space application. The Military and Aerospace Programmable Logic Device (MAPLD) International Conference, n. P70, p. 1{6, September 2003}.

[6] CARMICHAEL, C. Triple Module Redundancy Design Techniques for Virtex FPGAs. [S.l.], July 2006. (xapp197). Application note, v1.0.1.

[7] CARMICHAEL, C.; CAFFREY, M.; SALAZAR, A. Correcting Single-Event Upset Through Virtex Partial Reconguration. [S.l.], June 2000. (xapp216). Application note, v1.0.

[8] XILINX, INC., MicroBlaze Processor Reference Guide EDK, 2005. DOI= http://www.xilinx.com/ise/embedded /mb _ref_guide.pdf

[9] XILINX, INC., MicroBlaze Product Brief, 2005. DOI= http://www.xilinx.com/bvdocs/ipcenter/data_sheet/MB_sell_sh eet.pdf

[10] XILINX, INC., http://www.xilinx.com/products/design_re sources/proc_central/microblaze.htm

[11] XILINX, INC., "Virtex-II Platform FPGA User Guide", June 2003.

[12] SEDCOLE, N. P. Recongurable Platform-Based Design in FPGAs for Video Image Processing. Tese (Phd Thesis) | Imperial College of Science, Technology and Medicine, January 2006.

[13] SEDCOLE, P. et al. Modular dynamic reconguration in virtex fpgas. IEE Proceedings Computers and Digital Techniques, v. 153, n. 3, p. 157{164, May 2006.

[14] UPEGUI, A.; SANCHEZ, E. Evolving hardware by dynamically reconguring Xilinx FPGAs. In: MORENO, J. (Ed.). Evolvable Systems: From Biology to Hardware. Berlin Heidelberg: Springer-Verlag, 2005. (LNCS, v. 3637), p. 56-65.

[15] DIGILENT, INC., http://www.digilentinc.com/Products/Det ail.cfm?Nav1=Prod ucts&Nav2=Programmable&Prod=XUP V2P.

[16] XILINX, INC.,http://www.xilinx.com/univ/xupv2p.html

[17] XILINX, INC., "Virtex FPGA Series Configuration and Readback", Apllication note XAPP502. November 2001.

[18] XILINX, INC., "Getting Started with the Embedded Development Kit (EDK)"

[19] XILINX, INC., "Using a Microprocessor to Cofigure Xilinx FPGAs via Slave Serial or SelectMAP Mode", Apllication note XAPP502. December 2007.

[20] XILINX, INC., "Designing Custom OPB Slave Peripherals for MicroBlaze", February 2002. Tutorial Report.

[21] XILINX, INC., "Xilinx 8.2 Libraries Guide".

[22] XILINX, INC., "Development System Reference Guide 8.2i"