FPGA Redundancy Recovery based on Partial Bitstreams for Multiple Partitions

Victor M. Gonçalves Martins*†, João Gabriel Reis*, Horácio C. C. Neto†, Eduardo Augusto Bezerra*
*Department of Electrical and Electronic Engineering
Federal University of Santa Catarina
88040-900 Florianópolis, Santa Catarina - Brazil
{victor.martins, joao.reis, eduardo.bezerra}@eel.ufsc.br
†Electronic System Design and Automation - INESC-ID
Rua Alves Redol, 9; 1000-029 Lisboa, Portugal
{marte, hcn}@inesc-id.pt

***Abstract*—The *Triple Modular Redundancy* (TMR) strategy is a common and good option for a system to recover from possible faults. However, in case of a permanent fault in the TMR hardware, the redundancy advantage can be destroyed. In this paper, we present a low cost solution to enable *Fault Detection, Isolation and Recovery* (FDIR) in a TMR system. Our methodology uses the *Assisted Design Flow* (ADF) that implements *Partial Bitstream for Multiple Partitions* (PB4MP). This method allows to relocate modules in different *Reconfigurable Partition*s (RPs) using only one bitstream per module. The results show that with a minor increase in the program memory of the digital design, it is possible to increase considerably the TMR availability.**

Keywords*—*Bistream for Multiple Partitions; FPGA Recovery

I. Introduction

The usage of *Field Programmable Gate Array*s (FPGAs) in critical applications, where eventual faults may result in a system crash, is increasing. Even when all the tests in the manufacturing process are performed successfully, there is no assurance that the design is perpetually immune to permanent faults. In fact, these devices also suffer from aging, which results in a natural physical degradation process [1]. The aging speed depends on several factors such as the maturity of the technology used, manufacturing quality control, temperature and thermal shock (related to the environment in which it operates), operating voltage, dimensions, and even design complexity [1]. As the FPGA resources density keeps increasing in each successive technology generation, reliability concerns, mainly due to the *Negative Bias Temperature Instability* (NBTI), become a relevant factor [2]. Faults created by NBTI do not radically damage the FPGA resources, but they increase its response time [2].

In this paper, a new strategy based on *Triple Modular Redundancy* (TMR) [1] is proposed. The TMR architecture has autonomous capabilities to FDIR [3]. To make this possible, an ADF has been developed that implements the proposed PB4MP mechanism. Our approach is mainly applied to embedded systems in FPGA devices using hardcore or softcore processors [4]. Although it has been developed targeting the Virtex-6 Xilinx XC6VLX240T device, the proposed methodology can be used in the latest Xilinx families with *Partial Reconfiguration* (PR) support. The results show that it is possible to detect and to tolerate a permanent fault with TMR, with only a slight increase in the amount of memory required by a processor.

The paper is organized as follows. Section II describes the state-of-the-art. Section III presents the proposed architecture, the design flow associated, and the memory requirements. Section IV summarizes the case studies and presents the results. Section V concludes this paper and describes the future work.

II. Related Work

This section discusses related work regarding FPGA designs based on FDIR [3] approaches. Available techniques and technology that allow the system implementation are also discussed.

A. Fault Detection, Isolation and Recovery *(FDIR)*

The use of TMR [1] approaches is not new. The work in [5] presents a system with TMR which uses the FPGA PR capacities. In their work the faults detection is performed automatically. If a faulty module is detected, the corresponding partition is excluded, the module is allocated to an alternate partition and the system returns to its normal operation. However, this approach sacrifices a considerable amount of resources for module relocation. Furthermore, the relocation process requires a partial bitstreams library, which requires external memory.

In another project [6], although without TMR, a framework called *Dynamic Partially Reconfigurable* (DPR) was developed. This platform is made of several RPs called slots. These slots are allocated hardware modules, which are identified by the author as *Collaborative Macro-Functional Unit*s (CMFUs). Each CMFU has a *Built-In Self-Test* (BIST) module allowing periodic testing. When a fault is detected by the test, the CMFU is relocated to another free slot, and the previous slot is marked as faulty.

B. FPGA Configuration Manipulation

Tools as PR PlanAhead from Xilinx [7] allow to create RPs individually and to generate a partial bitstream for each module in each RP.

Xilinx provides the *Isolation Design Flow* (IDF) [8]], which can be used to enssure that the implemented module in an RP stays completely isolated from the rest of the FPGA implementation.

The work in [9] presents a flow that tries guarantee the compatibility of RPs interfaces. Although very interesting, this process is manual and does not cover the following: situations where one output signal from one partition has a fanout greater than one; situations where the source of the input signal of a partition has a fanout greater than one; and situations where the clock tree is not relatively the same.

C. Contributions

The main contribution of this work is to propose a way to implement a TMR system with permanent fault recovery capability. Another important contribution is the increased reliability using the ADF-PB4MP flow which, with extra precision constraints, gives us the ability to swap modules between RPs. With this new mechanism it is possible to provide the TMR system with the capability to recover from permanent faults without wasting FPGA resources by using extra partitions as in [5] [6].

III. Proposed TMR based on PB4MP

In this paper, a system using TMR with permanent fault recovery capability is proposed. As it is developed using the PB4MP tool, its reliability increase is obtained with no addicional FPGA resources. The PB4MP tool uses the capabilities of PR [7] and IDF [8] to perform the desired implementation.

A. TMR Architecture

Basically, the TMR mechanism implies implementing the same module/function three times. The redundant modules run in parallel, performing the same computation. The results from the three modules are analyzed by another module, responsible for voting the correct result. This way, even if a module produces wrong results, the other two will guarantee the correct operation. The weak point in this strategy is the voter, as if it fails, the whole TMR strategy will also fail. The major advantage of TMR is its error detection and correction feature. An important drawback is the need to employ three times more hardware [1].

B. The PB4MP Adaptation for TMR Purpose

Solutions like [5] [6] that use the normal PR flow imply an unavoidable limitation: it is necessary to generate a partial bitstream for each module in each RP. It means that, if there are N modules, M RPs, and we wish to allocate any module in any RP, it will be necessary to create a library with $N * M$ partial bitstreams.

The PB4MP allows to overcome this major limitation. Using a compilation of knowledge, from IDF [8], previous third party work [9], and Xilinx documentation [10], we developed this tool used in an *Assisted Design Flow* (ADF).

The ADF has the following requirements:

- All RPs need to include exactly the same FPGA resources with the same physical distribution;
- The vertical RP limits (up and down) need to be coincident with the FPGA clock regions;
- The RPs resource areas only can include I/O resources if they are not used by the system;

Following these constraints, the ADF permits implementing an FPGA system where all RPs have the same physical interface, which allows to use only one partial bitstream for each module that can be allocated in any RP. Therefore only N partial bitstreams are required, instead of the NxM required in the aforementioned works.

To take advantage of the PB4MP in the implementation of the TMR system, an extra constraint list is added to the ADF process. This list is built using the constraint `CONFIG PROHIBIT` which give us the option to select the FPGA resources that we do not want to use. In the selected policy implemented for this first version, a full resource column is removed in each three columns. To guarantee the desired reliability, this process never excludes a column with the same relative position of another column already excluded in another RP, as shown in Fig. 1.

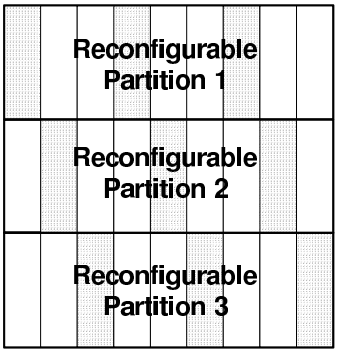

Fig. 1. TMR RPs resources distribution

With this distribution, if a module suffers one permanent fault, it is possible to replace the RP with another module that does not use the same resource where the permanent fault is.

This distribution can look like a waste of 33% of the RP resources, however, in an FPGA system as a result of the routing congestion this is not a real overhead. When the routing is limited to the RP size, this congestion is bigger. Therefore and in practice, the proposed approach does not imply additional hardware consumption.

C. Assisted Design Flow *(ADF) for the proposed TMR*

Fig. 2 shows the full design flow of a PB4MP-based FPGA system. It is based on the standard Xilinx tools flow (ISE and PlanAhead) and requires only a small user intervention at the initial phase of the hardware design, as in the IDF process [8]. The remaining steps are automated.

The base of the flow is the sequence Synthesis $\rightarrow$ Floor Plan $\rightarrow$ Translation $\rightarrow$ Mapping $\rightarrow$ Placement and Routing $\rightarrow$ Configuration Bitstream Generation. It is important that the Hardware Description Language specification is structured to correctly assign each module to the appropriate RP. The Floor Plan, that defines the RPs locations, is the other step that needs

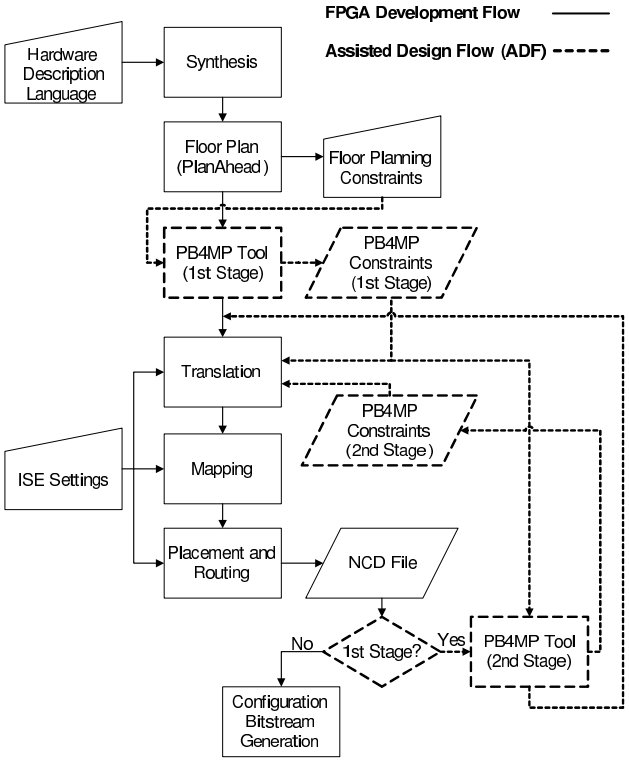

Fig. 2. FPGA Development + *Assisted Design Flow* (ADF)

the designer attention. In this case, it is imperative to follow the requirements listed in Section III-B.

After this stage, the PB4MP Tool generates the first constraints group. These constraints will define a window of possible locations for the RP input/ouput buffers, such that the following steps maintain some freedom to obtain a good placement and routing.

Initially, the Placement and Routing operation generates an NCD file. The PB4MP tool is then executed again and the nets that interconnect the modules in the RPs to the remaining parts of the system are extracted using the Directed Routing Constraints tool from `fpga_edline.exe`. With this information, a second constraints group is created and the sequence, Translation $\rightarrow$ Mapping $\rightarrow$ Placement and Routing is run again. After these two stages the final bitstream is generated and is ready to be downloaded to the FPGA.

D. Memory Usage for Recovery Ability

The memory usage of the software that implements this recovery ability depend on various details of the adopted strategy. The basis of the strategy is the same, swapping the modules between RPs, but can be done in two different ways. The modules are available externally to the FPGA in a memory with the partial bitstreams, or the swap is implemented by moving the correspondent part of the FPGA configuration memory. Our preferential option is the second one, as it does not need extra hardware. Nevertheless, this solution can be more or less demanding on the memory requirements.

In order to evaluate the memory consumption, a set of equations was defined based on TABLE I and TABLE II. TABLE I describes all software functions and its memory (compiled using gcc for MIPS32 architecture with -O3 optimization flag) requirements for the Virtex-6 family. The Memory Usage column shows the amount required for the code part (M_{CODE}). TABLE II details all hardware parameters that influence the memory usage (M_{DATA}).

TABLE I. SOFTWARE ROUTINES WITH INDIVIDUAL MEMORY CHARACTERISTICS

Software Routine Description	Bytes
RP_swap(rp1, rp2): Function that uses all routines below to implement the algorithm to swap two RPs.	418
XHwICAP Base Driver: Base Driver to communicate with XHwICAP (include Init(), SelfTest() and GetConfigReg() functions).	7012
XHwICAP DeviceReadFrame(): Routine to read one frame from the FPGA configuration memory.	554
XHwICAP DeviceWriteFrame(): Routine to write one frame in the FPGA configuration memory.	784
Total M_{CODE}	**8768**

TABLE II. PARTITION CHARACTERISTICS

RP Parameters	Parameters description for each hardware module
$Size_{Frame}$	Number of bytes in one frame.
N_{RPs}	Number of RPs in the system.
$N_{RPFrames}$	Number of frames that have one full RP configuration.

The memory required is divided in two parts: code (M_{CODE} detailed on TABLE I) and temporary data (M_{DATA}). The Data section stores the initial addresses of the RPs, four bytes for each one, and temporarily backups the FPGA configuration memory content from the RPs being swapped. In the low cost option, where there is one read/write operation for each frame, it is required twice the frame size $Size_{Frame}$ (equation 1a). For the worst case, when the swap is done with only one whole read/write operation for each RP, it is required the double of the size of the configuration memory of an RP. This means twice the frame size $Size_{Frame}$ for each one of the $N_{RPFrames}$ frames (equation 1b). Equation 1c gives the amount of memory required to implement the recovery ability. The total amount depends on the FPGA family, as the frame size $Size_{Frame}$ varies, and on the number of frames to be moved in each read/write operation.

$$M_{DATA} = N_{RPs} \times 4 + Size_{Frame} \times 2 \quad (1a)$$

$$M_{DATA} = N_{RPs} \times 4 + Size_{Frame} \times N_{RPFrames} \times 2 \quad (1b)$$

$$MEM_{Total} = M_{CODE} + M_{DATA} \quad (1c)$$

IV. EXPERIMENTAL RESULTS AND DISCUSSION

In order to explore and evaluate this new approach we implemented the System-on-Chip Plataform [4] shown in Fig. 3. Our platform supports dynamic reconfiguration and was implemented in a Xilinx's XC6VLX240T FPGA.

It is based on an implementation of the MIPS32 ISA, the Plasma softcore, which is freely available at Opencores [11]. The internal structure of a CPU node is based on the AXI4 family of protocols, which is becoming the industry's standard for bus-based interconnections. The hardware reconfiguration

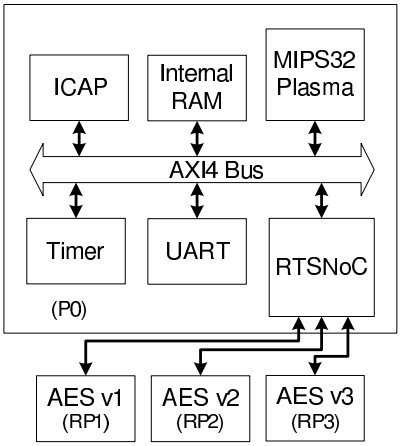

Fig. 3. System Diagram

is done by the *Internal Configuration Access Port* (ICAP) interface, that is interconnected through the AXI4 and that is managed by the *Operating System* (OS).

RP1, RP2 and RP3 represent the RPs that contain in this case, one *Advanced Encryption Standard* (AES) component each one. The three AESs are implemented following the resources distribution shown in Fig. 1.

We have chosen a *Real-Time Star Network-on-Chip* (RTSNoC) [12] based interconnection scheme for the reconfigurable partitions. Each AES is connected to a port of an RTSNoC router and one of the RTSNoC routers ports is connected to an AXI4 bridge.

The input data is sent to the three AESs modules. After the processing time, all results are retrieved from the same modules. The three results are analysed in one vote function. If a result mismatch is detected, the erroneous module is identified and another RP is selected to swap with it. Before performing the exchange, the input clock signal is disabled in both RPs.

Each AES module occupies 4172 *Look Up Table*s (LUTs), which in the Virtex-6 family means that a minimum of 522 *Configurable Logic Block*s (CLBs) is required. Because a section of a CLB column has 40 CLBs, at least 14 of such CLB columns are needed. In practice and due to Placement and Routing requirements (without any `CONFIG PROHIBIT` constraint), the module implementation required 23 CLB columns. This fact gave us the opportunity to implement the distribution presented in Fig. 1, without wasting extra resources in the second stage and with no routing congestion. The characteristics of each RP are presented in the TABLE III.

TABLE III. AES RP CHARACTERISTICS

RP Parameters	Parameters Values
$Size_{Frame}$	324
N_{RPs}	3
$N_{RPFrames}$	1744 (1232 for CLBs + 512 for BRAMs)

Using equation 1a we obtained $M_{DATA} = 660$ bytes, and from equation 1c we concluded that our TMR approach needs only 9498 bytes of extra memory from the system to provide better availability.

V. CONCLUSIONS AND FUTURE WORK

The main objective of this work was to propose a TMR system using the developed ADF-PB4MP tool, which is based on the Xilinx ISE flow. The proposed approach increases considerably the availability of the TMR system with limited memory increment, and with a residual hardware increment.

The PB4MP approach can be used in other FPGA systems where it is important to expand the life cycle. Without a TMR implementation we must provide the modules with a test mechanism, such as [13], to implement the FDIR policy.

As a future work, we are planning to migrate the ADF to the Vivado flow, and start investigating how the PB4MP approach can enhance the FPGA life cycle by reducing the stress effects caused by NBTI.

ACKNOWLEDGEMENTS

This work has been partly funded by the National Counsel of Technological and Scientific Development (CNPq).

REFERENCES

[1] Israel Koren and C. Mani Krishina. *Fault-Tolerant System*, chapter 2-Hardware Fault Tolerance. Morgan Kaufmann, 2007.

[2] Rajeev Kumar Mishra, Amritanshu Pandey, and Sarfraz Alam. Analysis and impacts of negative bias temperature instability (nbti). In *Proc. of the IEEE Students Conference on Electrical, Electronics and Computer Science (SCEECS)*, pages 1–4, Bhopal, India, March 2012.

[3] Andrea Guiotto, Andrea Martelli, and Carlo Paccagninis. Smart-fdir: use of artificial intelligence in the implementation of a satellite fdir. ftp://ftp.estec.esa.nl/pub/wm/anonymous/wme/Web/SmartFDIR2003.pdf, June 2003. Project coordened by Alenia Spazio (ALS) with participation of Politecnico di Milano (POLIMI).

[4] Tiago Rogério Mück and Antônio Augusto Fröhlich. Seamless integration of hw/sw components in a hls-based soc design environment. In *Proc. of the International Symposium on Rapid System Prototyping (RSP)*, pages 109–115, Montreal, Canada, October 2013.

[5] David P. Montminy, Rusty O. Baldwin, Paul D. Williams, and Barry E. Mullins. Using relocatable bitstreams for fault tolerance. In *Proc. of the Adaptive Hardware and Systems (AHS 2007). NASA/ESA Conference*, pages 701–708, Edinburgh, Scotland, August 2007.

[6] Victor Dumitriu and Lev Kirischian. Soc self-integration mechanism for dynamic reconfigurable systems based on collaborative macro-function units. In *Proc. of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '13)*, pages 1–7, Cancun, Mexico, December 2013.

[7] Xilinx Inc. Partial reconfiguration tutorial: Planahead design tool v14.5, April 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/PlanAhead_Tutorial_Partial_Reconfiguration.pdf.

[8] Xilinx Inc. Isolation design flow. http://www.xilinx.com/applications/isolation-design-flow.html, 2014.

[9] Yoshihiro Ichinomiya, Sadaki Usagawa, Motoki Amagasaki, Masahiro Iida, Morihiro Kuga, and Toshinori Sueyoshi. Designing flexible reconfigurable regions to relocate partial bitstreams. In *Proc. of the IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, page 241, Toronto, Canada, May 2012.

[10] Xilinx Inc. Constraints guide v14.5 (ug625). http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/cgd.pdf, April 2013.

[11] OpenCores. Opencores. http://opencores.org, November 2014.

[12] Marcelo Daniel Berejuck. Dynamic Reconfiguration Support for FPGA-based Real-time Systems. Technical report, Federal University of Santa Catarina, Florianópolis, Brazil, 2011. PhD qualifying report.

[13] Victor M. G. Martins, Frederico Ferlini, Djones V. Lettnin, and Eduardo A. Bezerra. Low cost fault detector guided by permanent faults at the end of fpgas life cycle. In *Proc. of the IEEE Latin America Test Workshop (LATW)*, pages 1–6, Fortaleza, Brasil, March 2014.