NBTI mitigation by Dynamic Partial Reconfiguration

Authors: Di Carlo, S.; Galfano, S. ; Gambardella, G. ; Indaco, M. ; Prinetto, P. ; Rolfo, D. ; Trotta, P.

Published in the Proceedings of the IEEE 13th Biennial Baltic Electronics Conference

DOI: 10.1109/BEC.2012.6376823

2012 13th Biennial Baltic Electronics Conference (BEC2012)
Tallinn, Estonia, October 3-5, 2012

NBTI Mitigation by Dynamic Partial Reconfiguration[1]

Stefano Di Carlo, Salvatore Galfano, Giulio Gambardella, Marco Indaco, Paolo Prinetto, Daniele Rolfo, Pascal Trotta

Politecnico di Torino
Dipartimento di Automatica e Informatica
Corso Duca degli Abruzzi 24, I-10129, Torino, Italy
Email: {name.familyname}@polito.it

ABSTRACT: FPGAs achieve smaller geometries and their reliability is becoming a severe issue. Non-functional properties, as Negative Bias Temperature Instability, affect the device functionality. In this work a novel methodology to address this issue is described, exploiting FPGAs flexibility. Dynamic Partial Reconfiguration is used to minimize aging impact on FPGAs' configuration memory.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are integrated circuits made up of a cluster of programmable functional blocks surrounded by programmable inteconnections and I/Os that can be configured to perform user described logic functions. The flexibility enhanced by FPGAs comes at a cost in terms of performance, in comparison with Application Specific Integrated Circuit (ASIC). For this reason, Extensible Processing Platforms (EPP) have been developed, combining high performance ASIC processors with programmable logic [10][4]. These platforms are suitable for embedded systems, thanks to their reduced cost, optimized size, low power consumption, and flexibility.
In this context, Dynamic Partial Reconfiguration (DPR) is a widely used approach [6][1] to reduce power consumption while keeping a high level of flexibility. DPR allows the user to dynamically change the function implemented by a portion of an FPGA. Furthermore, this feature can be used to increase the reliability of the system [5][7].
With the shrinking of technology dimension, also in FPGAs devices, Non-Functional Properties (NFPs) are becoming a major issue in devices' dependability. In sub-65 nm technologies, aging effects due to Negative Bias Temperature Instability (NBTI) have become the primary degradation mechanism [8].
This paper proposes an innovative solution to mitigate the NBTI effect in modern programmable systems, exploiting their DPR feature. The rest of the paper is organized as follows. Section 2 gives a background of the work. The proposed methodology for aging mitigation is discussed in Section 3. A case of study is then presented in Section 4, while in Section 5 some experimental results will be shown. Finally, Section 6 concludes the paper.

[1]This projects is partially funded by Ansaldo STS SpA and FinMeccanica SpA, within framework of the "Iniziativa Software" (II ediz)

2 Background

2.1 FPGA internal architecture

FPGAs are programmable devices, configured from the user to perform a desired function. To provide such a flexibility, different programmable blocks are present in the device [9]:

- *Input-Output Blocks* (IOBs) – programmable interfaces for external connections;
- *Configurable Logic Blocks* (CLBs) – used to implement Look-Up-Table (LUT) based logic functions;
- *General Routing Matrix* (GRM) – matrix of programmable interconnections among the blocks;
- *Block RAMs* (BRAMs) – dual-port fully synchronous RAM organized in columns.

The programmable logic is configured by storing proper information in a configuration memory, which controls the FPGA's behaviour. Many configuration technologies exists, but the prevalent nowadays is the SRAM-based. In this technology, the configuration information is stored in an external non-volatile memory; the device is programmed at power-up, by writing the SRAM configuration memory inside the device with the non-volatile memory content.

2.2 NBTI: Causes and Effects

NBTI occurs when a pMOS transistor, working at high temperature, is negatively biased, i.e., a logic '0' is applied to the gate of the pMOS, resulting in a gate-source voltage (V_{gs}) equal to the supply voltage ($-V_{DD}$). NBTI manifests itself as an increase of the transistor's threshold voltage (V_{th}) in time, and it strictly depends on the probability of having a '0' at the gate inputs.
Such a V_{th} degradation induces a progressive reduction of the current capability of the pMOS transistor, resulting in

978-1-4673-2774-9/12/$31.00 ©2012 IEEE

a substantial slowdown of CMOS gates (i.e., in a reduction of the maximum circuit's clock working frequency, f_{CK}). For a pMOS transistor, there are two phases of NBTI, depending on its bias condition:

- *stress*: when $V_{gs} = -V_{DD}$, positive interface traps are accumulated;
- *recovery*: when $V_{gs} = 0$, holes are not present in the channel and no new interface traps are generated.

Considering SRAM memories, the NBTI induces a degradation of the robustness of the cells (i.e., their capability to safety store a bit). A good metric to qualify the effect of NBTI in a memory cell is the *Signal Noise Margin* (SNM), i.e., the minimum *DC* noise voltage necessary to change the stored value [2]. The smaller the SNM is, the lower the reliability of the cell becomes. Unfortunately, the V_{th} shift induced by NBTI causes an SNM degradation, which in turn reflects itself on the stability of the cell. Because of the symmetric layout of the cell, the V_{th} shift is maximum if the stored value's zero-probability is near to 0 or 1. Clearly, the best case happens when the stored value is '0' for the 50% of the time, which means that both pMOS transistors age in the same way (see Section 5).

2.3 Dynamic Partial Reconfiguration

In modern SRAM-based FPGAs, the intrinsic flexibility has been expanded, including DPR. This technique makes possible to program at run-time a configurable area, without interrupting the activities performed by the remaining parts of the programmable device. DPR is performed by downloading *partial bitstreams* inside the FPGA through a configuration port (e.g., Internal Configuration Access Port *ICAP*). The configuration memory's content is then changed with the new information stored in the partial bitstream.

3 Proposed Methodology

The proposed method is a Design for Dependability (DfD) technique aimed at extending the FPGA device life by reducing the effects of the NBTI on the SRAM configuration memory.

It takes advantage of the unused hardware resources by allocating functions to be implemented uniformly into all available resources. To be applied, the proposed method needs unused resources. Clearly, the effective usable portion of the FPGA is reduced to only a half of the available resources, while the resources allocated to a single function (e.g., assigned to the implementation of an IP-core) are doubled, time-multiplexing their usage.

Resources are then periodically switched between two statuses: "*work*" and "*rest*". In *work* status they are normally operated (i.e., they implement the function specified at design time); in the *rest* status they are, and need to be, unused. Nevertheless, not using resources is not enough to prevent them from suffering the NBTI effects: for this reason, in the *rest* status, the unused resources are properly configured to lower the effects of the aging phenomenon.

In particular, the *rest* status is designed to minimize the impact of the NBTI on the SRAM configuration memory. As stated in Subsection 2.2, to minimize the NBTI effect in SRAM cell, each bit of the memory has to store complementary values for equal times. Therefore, in the *rest* status, the whole SRAM content should be the complementary of the one stored in the *work* state. Furthermore, the time spent in *rest* and in *work* statuses must be the same. The changing of the SRAM content implies that a DPR is performed.

To ensure the correctness of the methodology, the following three Design for Dependability rules should be fulfilled.

3.1 DfD#1: Static connection avoidance

Usually, Place and Route (PAR) tools, generate some (long) static connection crossing different portions of the FPGA. These static connections are controlled by some of the configuration memory bits.
The proposed methodology requires to flip the whole memory content. However, these static connections may be unintentionally broken if all the configuration memory bits are flipped.
To avoid such threats:

- the whole FPGA is partitioned into several partitions, one for each sub-function instance (counting also all their replica);
- setting proper options, while synthesizing the design, statical connections are forbidden to cross such partition boundaries [3]

3.2 DfD#2: Using different interfaces

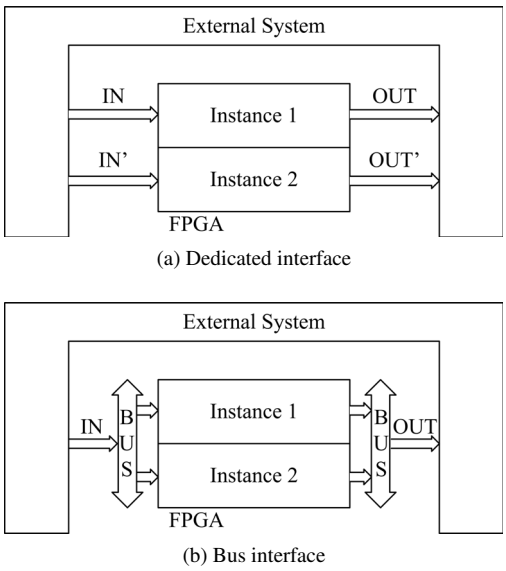
Since the whole configuration memory content must be flipped, all partitions of the FPGA have to change. Therefore, the programmable logic cannot contain static portions.

If all the instances of a module use the same communication ports (i.e., the same hardware), some configurable logic will be statically configured (so fixed values inside the corresponding memory bits), conflicting with the above statement.

To solve this issue, each instance of a module should use its own ports toward the external world. There are then different chances on how the external world should interface with the FPGA:

- the external user system (i.e., the system the FPGA is interfaced with) should provide the double of the minimum required number of ports and connect to each instance of a module in a FPGA with dedicated ports (Fig. 1(a));
- exploit an external bus (not laying in the configurable area) to connect the corresponding module ports and possibly, the ports of all the modules (Fig. 1(b)).

The former requires doubling the pins, so the related costs may be too high. Also, it would imply that the external system should make data flow alternating through



(a) Dedicated interface

(b) Bus interface

Figure 1: Possible connection schemes

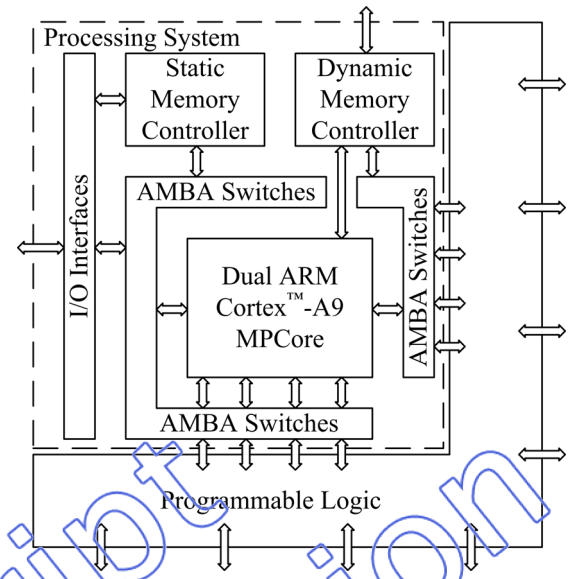


Figure 2: Zynq™Architecture

two sets of I/O interfaces (e.g., *IN-OUT* and *IN'-OUT'* of Fig. 1(a)). In the latter, instead, the external system has only one set of I/O interfaces, so pin-associated costs and input/output management complexity are reduced.

3.3 DfD#3: Smart external controller

A controller is of course needed to manage the DPR. The controller cannot be implemented in the programmable logic, because it would imply static configuration of the FPGA. Clearly, in order to assure this, the DPR controller must be implemented outside the FPGA device, so that it will not use programmable logic.

The controller has to decide when to reconfigure each module. To guarantee the correct behaviour of the overall system, the controller can just reconfigure modules when they are idle. Therefore, the controller needs to interface itself with the external user system to catch when the module to be reconfigured is unused. When this happens, it will have to reconfigure (through DPR) the module's instance, which was previously in *rest* status, to *work* status, and vice-versa (switching the instance from *work* status to *rest* status).

4 Case study

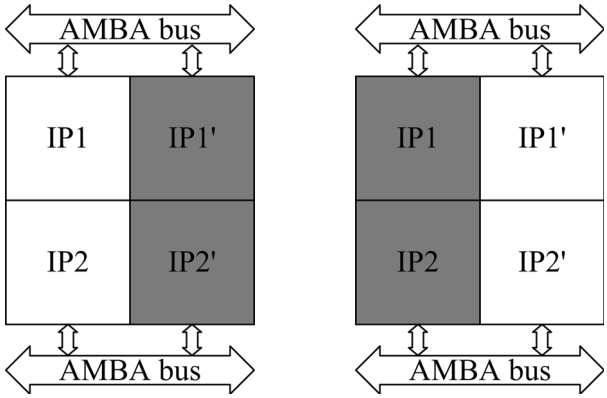
The proposed method has been implemented and applied to an existing application in order to assess its correct work. The considered case study is the use of a Zynq™-7000 [10] EPP by Xilinx to implement an embedded system having two IP-core modules.

The Zynq™-7000's architecture perfectly applies to the conceived method. As a matter of fact, as shown in Fig. 2, it includes a Dual Arm Cortex™-A9 MPCore processor, equipped with its own memory controllers, I/O logic, AMBA switches and some programmable logic. More in detail, the programmable logic is a Kintex™-7 (or Artix™-7) FPGA. According to Subsection 3.2, AMBA is used to deal with communication between the FPGA and the external system.

The function of the controller, described in Subsection 3.3, was here executed by the on-chip processor in time-sharing with the main program execution. To efficiently implement the controller functions, internal facilities like timer counter and DMA controller were used. Moreover, problems related to the controller to external system communication are solved easily because they are physically the same entity.

Hereafter an example of the operation of the system will be shown. Let us assume that the first IP-core has two instances, named *IP1* and *IP1'*, and the same is for the second, that has *IP2* and *IP2'* instances. Each instance is contained in a dedicated partition, in order to assure that no static connection lays in different modules' partitions, as explained in Subsection 3.1. Let us consider that the system is initially configured in the state (*IPx*, *IPx'*)= (*work*, *rest*) for each IP-core, as shown in Fig. 3(a). Please note that the greyed portions of the FPGA represent modules in *rest* status: they are configured with bitstreams ($\overline{BS1'}$ and $\overline{BS2'}$), i.e., the content of the SRAM configuration memory, which are respectively obtained by bit-wise inverting that ones which are used in *work* status ($BS1'$ and $BS2'$).

After a certain time, the timer counter expires and triggers an exception. This makes the CPU act as the controller of the DPR: *IP1'* and *IP2'* are reconfigured with the *work* state bitstreams ($BS1'$ and $BS2'$), while *IP1* and *IP2* are reconfigured with the inverted bitstreams, $\overline{BS1}$ and $\overline{BS2}$, leading the FPGA in the state (*rest*, *work*) (Fig. 3(b)): then new data now flow to and from these reconfigured modules. After a time delay, similar to the first one, everything repeats in the opposite way: the system goes again to (*work*, *rest*) state. Everything is periodically repeated over time so that each module's instance stays in the two states for equal times. This leads the configuration memory's bits



(a) State: (*work*, *rest*) (b) State: (*rest*, *work*)

Figure 3: Graphic representation of IP-core states within the FPGA

to have a stored value equal to '0' for 50% of the time, which minimizes the degradation due to NBTI.

Please note that the two IP-cores not necessarily switch at the same time (as shown in the Fig. 3). The switching frequency of each module was chosen to be compatible with the foreseen idle times of the cores.

5 Experimental results

To quantify the benefits of the proposed solution, the variation over time of the SNM in SRAM cells has to be considered. Further, a low SNI value negatively influences the dependability. Since the measure cannot be performed over time easily, a study using cell library models has been done. *STMicroelectronics*' 45 nm standard cells library was used because the 28 nm standard cells library, that is used in Xilinx 7 family [9], is not available: this does not invalidate the benefits lead by the proposed method as with 28 nm only the magnitude and speed of degradation over time changes. The reported data are evaluated at a temperature $T = 25°C$, $V_{DD} = 1.1V$ and channel width $W_{pMOS} = 0.21\mu m$.

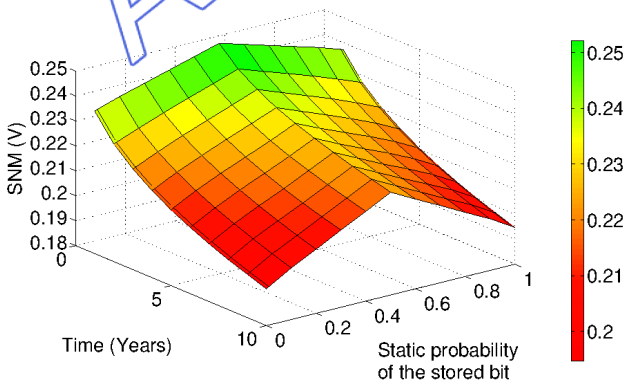


Figure 4: Signal Noise Margin degradation in function of time (measured in years) and static probability (probability to have '1' in a SRAM cell)

Fig. 4 clearly shows that the SNM of the cell decreases (worsens) during the years, but, with a static probability of 0.5, which is achieved using the proposed methodology, the decrease is considerably less than with any other probability.

6 Conclusions

In this work a DfD methodology to mitigate the NBTI effect in SRAM based FPGA is presented. Exploiting the DPR is possible to achieve the 50% probability of having '0' in the configuration memory bits. The proposed approach comprises three different DfD rules to assure that all the SRAM configuration memory's bits age the same. Thanks to this solution, the configuration memory of the device will have the minimum effect of aging during the device lifetime.

Acknowledgment

The authors would like to express their sincere thanks to the whole design team of Ansaldo STS SpA for their helpful hints and guidelines.

References

[1] N. Abel, S. Manz, F. Grull, and U. Kebschull. Increasing design changeability using dynamic partial reconfiguration. *Nuclear Science, IEEE Transactions on*, 57(2):602–609, April 2010.

[2] A. Calimera, E. Macii, and M. Poncino. Analysis of nbti-induced snm degradation in power-gated sram cells. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 785–788, June 2010.

[3] S. Di Carlo, G. Gambardella, M. Indaco, P. Prinetto, D. Rolfo, and P. Trotta. Dependable dynamic partial reconfiguration with minimal area & time overheads on xilinx fpgas. In *Submitted to Test Symposium 2012. ATS 2012. IEEE Asian*, 2012.

[4] Intel Corporation. *Intel Atom Processor E6x5C Series-Based Platform for Embedded Computing*, 2010.

[5] F. Lahrach, A. Doumar, and E. Chatelet. Fault tolerance of sram-based fpga via configuration frames. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on*, pages 139–142, April 2011.

[6] W. Lie and W. Feng-yan. Dynamic partial reconfiguration in fpgas. In *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, volume 2, pages 445–448, November 2009.

[7] M. Straka, J. Kastil, and Z. Kotasek. Fault tolerant structure for sram-based fpga via partial dynamic reconfiguration. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 365–372, September 2010.

[8] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao. The impact of nbti effect on combinational circuit: Modeling, simulation, and analysis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(2):173–183, February 2010.

[9] Xilinx Corporation. *7 Series FPGAs Overview*, March 2012.

[10] Xilinx Corporation. *Zynq-7000 Extensible Processing Platform Overview*, March 2012.