Dynamic Partial Reconfiguration for Dependable Systems

Giulio Gambardella

Giulio.Gambardella@polito.it

Submitted in total fulfillment of the requirements
of the degree of Doctor of Philosophy

January 2014

Faculty of Computer Engineering
Department of Control and Computer Engineering
Politecnico di Torino

Herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This thesis has not previously been presented in identical or similar form to any other Italian or foreign examination board. The thesis work was conducted from 01/2011 to 12/2014 under the supervision of Prof. Paolo Prinetto at Politecnico di Torino.

LIST OF PUBLICATIONS

Journal papers

S. Di Carlo, **G. Gambardella**, P. Prinetto, D. Rolfo, P. Trotta: *SATTA: a Self-Adaptive Temperature-based TDF awareness methodology for dynamically reconfigurable FPGAs*, ACM Transactions on Reconfigurable Technology and Systems (TRETS)

S. Di Carlo, **G. Gambardella**, P. Prinetto, D. Rolfo, P. Trotta: *SA-FEMIP: a Self-Adaptive Features Extractor and Matcher IP-core based on Partially Reconfigurable FPGAs for Space Applications*, IEEE Transactions on Very Large Scale Integration Systems (TVLSI)

Conference papers

S. Di Carlo, **G. Gambardella**, M. Indaco, D. Rolfo, P. Prinetto: *A unifying formalism to support automated synthesis of SBSTs for embedded caches*, on Proceedings of the 9th East-West Design & Test Symposium (EWDTS), pages 39–42, 9^{th} – 12^{th} September 2011, Sebastopol (Ukraine).

S. Di Carlo, **G. Gambardella**, M. Indaco, D. Rolfo, P. Prinetto: *MarciaTesta: an automatic generator of test programs for microprocessors' data caches*, on Proceedings of the 20th Asian Test Symposium (ATS), pages 401–406, 20^{th} – 23^{rd} November 2011, New Delhi (India).

S. Di Carlo, **G. Gambardella**, M. Indaco, D. Rolfo, P. Prinetto: *Validation & Verification of an EDA automated synthesis tool*, on Proceedings of the 6th International Design and Test Workshop (IDT), pages 48–52, 11^{th} – 14^{th} December 2011, Beirut (Lebanon).

S. Di Carlo, **G. Gambardella**, M. Indaco, D. Rolfo, G. Tiotto, P. Prinetto: *An Area-Efficient 2-D Convolution Implementation on FPGA for Space Applications*, on Proceedings of the 6th International

Design and Test Workshop (IDT), pages 88–92, 11^{th} – 14^{th} December 2011, Beirut (Lebanon).

S. Di Carlo, S. Galfano, **G. Gambardella**, M. Indaco, P. Prinetto, D. Rolfo, P. Trotta: *NBTI Mitigation by Dynamic Partial Reconfiguration*, on Proceedings of the 13th Biennal Baltic Electronics Conference (BEC), pages 93–96, 3^{rd} – 5^{th} October 2012, Tallinn (Estonia).

S. Di Carlo, **G. Gambardella**, T. Huynh Bao, P. Prinetto, D. Rolfo, P. Trotta: *ZipStream: improving dependability in Dynamic Partial Reconfiguration*, on Proceedings of the 7th International Design and Test Symposium (IDT), pages 1–6, 15^{th} – 17^{th} December 2012, Doha (Qatar).

S. Di Carlo, **G. Gambardella**, P. Lanza, P. Prinetto, D. Rolfo, P. Trotta: *SAFE: a Self Adaptive Frame Enhancer FPGA-based IP-core for real-time space applications*, on Proceedings of the 7th International Design and Test Symposium (IDT), pages 1–6, 15^{th} – 17^{th} December 2012, Doha (Qatar).

S. Di Carlo, **G. Gambardella**, M. Indaco, I. Martella, P. Prinetto, D. Rolfo, P. Trotta: *A software-based self test of CUDA Fermi GPUs*, on Proceedings of the 18th European Test Symposium (ETS), pages 1–6, 27^{th} – 30^{th} May 2013, Avignon (France).

S. Di Carlo, **G. Gambardella**, M. Indaco, I. Martella, P. Prinetto, D. Rolfo, P. Trotta: *Increasing the robustness of CUDA Fermi GPU-based systems*, on Proceedings of the 19th International On-Line Testing Symposium (IOLTS), pages 234–235, 8^{th} – 10^{th} July 2013, Chania (Greece).

S. Di Carlo, **G. Gambardella**, P. Lanza, P. Prinetto, D. Rolfo, P. Trotta: *FEMIP: A high performance FPGA-based features extractor & matcher for space applications*, on Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL), pages 1–4, 2^{nd} – 4^{th} September 2013, Porto (Portugal).

S. Di Carlo, **G. Gambardella**, M. Indaco, P. Prinetto, D. Rolfo, P. Trotta: *Dependable Dynamic Partial Reconfiguration with minimal area & time overheads on Xilinx FPGAs*, on Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL), pages 1–4, 2^{nd} – 4^{th} September 2013, Porto (Portugal).

S. Di Carlo, **G. Gambardella**, I. Martella, P. Prinetto, D. Rolfo, P. Trotta: *Fault mitigation strategies for CUDA GPUs*, on Proceedings of the International Test Conference (ITC), pages 1–8, 6^{th} – 13^{th} September 2013, Anaheim (USA).

T. Sanislav, G. Mois, S. Folea, L. Miclea, **G. Gambardella**, P. Prinetto: *A cloud-based Cyber-Physical System for environmental monitoring*, on Proceedings of the 3rd Mediterranean Conference on

Embedded Computing (MECO), pages 6–9, 15th – 19th June 2014, Budva (Montenegro).

S. Di Carlo, **G. Gambardella**, P. Prinetto, D. Rolfo, P. Trotta, A. Vallero: *A novel methodology to increase fault tolerance in autonomous FPGA-based systems*, on Proceedings of the 20th International On-Line Testing Symposium (IOLTS), pages 87–92, 7th – 9th July 2014, Platja d'Aro (Spain).

S. Di Carlo, **G. Gambardella**, P. Prinetto, F. Reichenbach, T. Løkstad, G. Rafiq: *On enhancing fault injection's capabilities and performances for safety critical systems*, on Proceedings of the 17th Euromicro Conference on Digital System Design (DSD), pages 583–590, 27th – 29th August 2014, Verona (Italy).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Confidence is what you have before
you understand the problem

Woody Allen

CHAPTER 1

INTRODUCTION

Customer needs and market requirements have acted as a constant push to technology during the past 40 years, somehow following Moore's prediction. This driving force had, on the one hand, led to great benefits in terms of computational power and lower cost, while, on the other hand, caused several headaches to dependability engineers. The

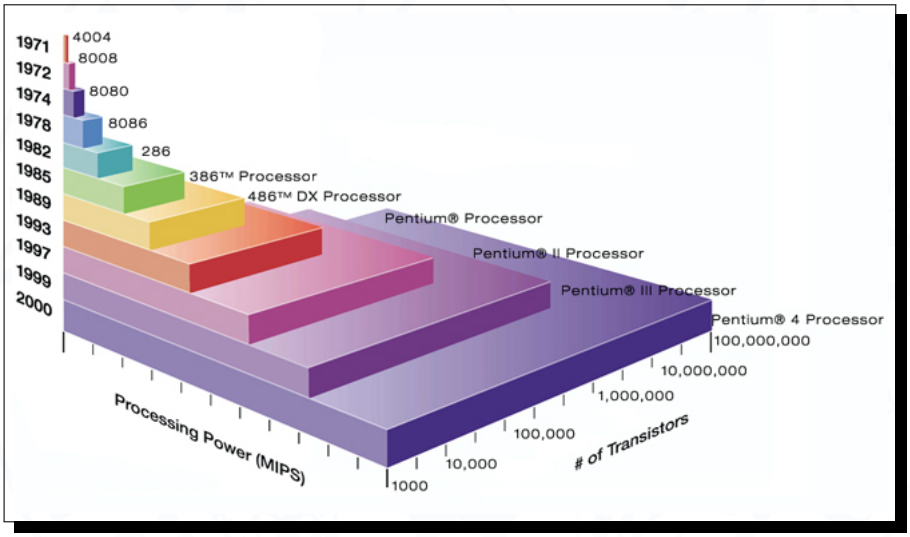


Figure 1.1: Moore's Law applied at Intel's processors

so-called die shrinking for digital logic, as summarized by the ITRS, can be defined as *"scaling of the MOSFETs for leading-edge logic technology in order to maintain historical trends of improved device performance"* [34].

Such a continuous scaling has been characterized of several phases, depending on the methodology adopted to achieve the scaling. As shown in Figure 1.2, three main Technology Enablers

have been identified:

1. **Lithography**: optical lithography scaling has been the exploited by the semiconductor industry for several generations, and enabled classical geometrical CMOS scaling by economically decreasing the resolution, feature size and die area of devices at every subsequent technology node. Due the challenges associated with scaling the wavelength of the light source used for modern optical lithography systems and theoretical optical limits [36], industry had to make a step forward to the basic geometrical scaling;

2. **Materials**: in order to follow Moore's law beyond the limits of wavelength scaling, new materials have been employed in technology nodes after 65 nanometers. Among them, the most important innovations, as reported by Intel in [45], have been the introduction of: High-k dielectric replacing silicon dioxide to guarantee higher impedance; metal gates for planar CMOS technologies replacing poly-silicon to decrease electrical resistance; strained silicon, in order to increase channel mobility by mean of compressive (for PMOS) or tensile (for NMOS) strain;

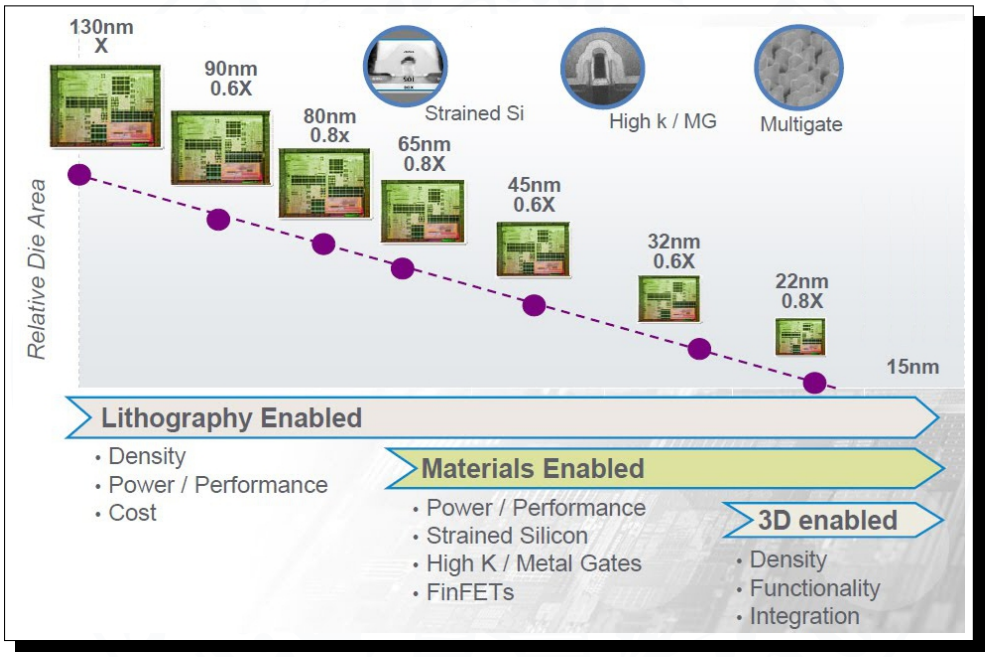


Figure 1.2: 3 different technology enablers in recent nodes (by Global Foundries)

3. **3D**: the traditional flat two-dimensional planar gate has been replaced with a thin three-dimensional silicon fin that rises up vertically from the silicon substrate, allowing the insertion of multiple gate accesses. Further then to create the so-called FinFET transistor, 3D

technology has been exploited also to build IC composed of multiple layers of active components. The scaling is then emulated, since the actual transistor dimension is not-scaled or scaled very little, but the components per area is anyway increasing.

In the past, the technological reliability margins that were available to achieve the required failure rate levels were always sufficiently high. Also reliability could be guaranteed at the 130nm technology level, based on accelerated stress experiments [25]. When the device geometries are scaled into the range of 65nm CMOS technologies and below, however, the available reliability margins are strongly reduced, implying new paradigms in reliability approaches 1.3.

At every technology node, as it may be expected, the gain in terms of performances, speed and

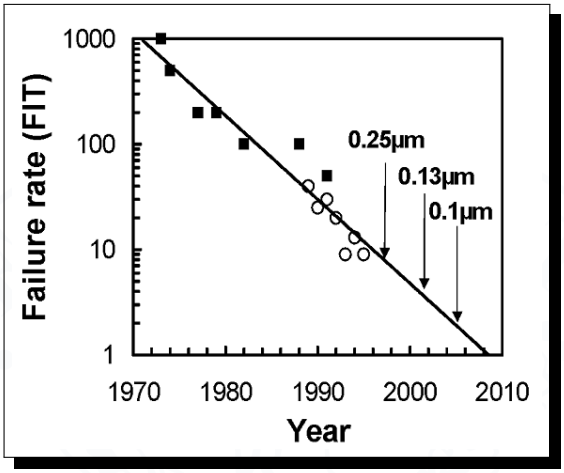


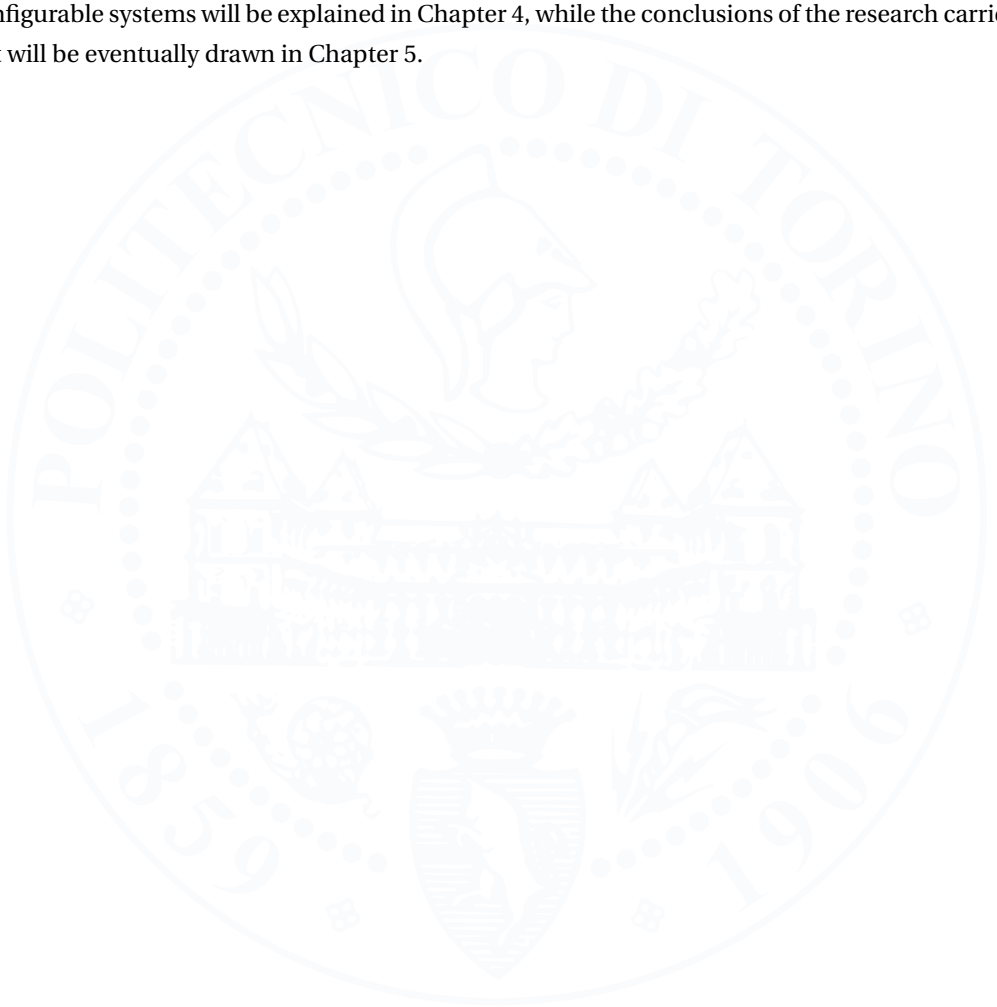
Figure 1.3: Required failure rates over years [25]

possibly cost of ICs does not come for free. In the sub-micron era, miniaturization affects several aspects of the produced device: MOS leakage current has been proved to increase due to direct tunnelling [60], thus raising static power consumption.

As a secondary effect, the increase of die temperature in modern silicon-based devices act as a major hard failure cause. Experimental results shows that scaling has a significant and increasing impact on processor hard failure rates, due to the increase in processor temperature. The maximum temperature reached by a 65nm processor is 15 degrees Kelvin higher than that reached by a 180nm processor. The failure rate for a 65nm processor is 316% higher than the failure rate at 180nm, with similar reliability qualification [35].

Another major issue due to technology scaling and dramatically enhanced at high temperature has been identified as process variation. Advanced CMOS technology includes two major types of process variability: local (intradie) and global (interdie). Local variability is the parametric changes of identical MOSFETs across a short distance. Global variability refers to such changes for identical MOSFETs separated by a longer distance or fabricated at a different time [52]. The

effects at system level of both intradie and interdie variation feature another criticality: non predictiveness. Thus, no countermeasures further then worst case design has been ever taken into account for safety critical applications.
The works presented in this thesis tries to face new dependability issues in modern reconfigurable systems, exploiting their special features to take proper counteractions with low impact on performances. The thesis is organized following the reasoning that have been made during the research phases. First of all, the most critical dependability issues of recent technology nodes are evaluated in Chapter 2, highlighting the two main effects that the research tackled.
Following, Chapter 3 will give the reader the chance of understanding the reconfigurable architecture, namely FPGA, that has been adopted in the research experiments, focusing on the Dynamic Partial Reconfiguration feature.
The methodologies adopted to increase the overall dependability of dynamically and partially reconfigurable systems will be explained in Chapter 4, while the conclusions of the research carried out will be eventually drawn in Chapter 5.



Simplicity is prerequisite for reliability

Edsger W. Dijkstra

CHAPTER 2

DEPENDABILITY ISSUES IN DIGITAL SYSTEMS: AN OVERVIEW

Dependability has been defined as *the quality (previously referred as trustworthiness and continuity) of the delivered service such that reliance can justifiably be placed on this service* [42].

More recently, the term dependability has described as the composition of four main properties, namely:

- **R**eliability: the probability that a piece of equipment or component will perform its intended function satisfactorily for a prescribed time and under stipulated environmental conditions;
- **A**vailability: the probability that the system will be functioning correctly at any given time;
- **M**aintenability: the ease with which a product can be maintained in order to cope with a changed environment;
- **S**afety: the freedom from undesired and unplanned event that results in a specific level or loss (i.e. accidents).

that are usually referred as RAMS.

The achievement of high level of dependability in digital systems is critical whenever the life of a human being is in charge of the digital system itself. Ad-hoc devices must be adopted in those situations, guaranteeing a low level of risk.

Accordingly, safety critical systems often rely on the adoption of old technology nodes, even if they introduce longer design time w.r.t. consumer electronics. In fact, functional safety requirements, due to international directives like IEC-61508 [32] for industrial applications and ISO-26262 [33] for automotive, are increasingly pushing industry in developing innovative method-

ologies to design high-dependable systems with the required diagnostic coverage.
More recently, commercial off-the-shelf (COTS) devices adoption began to be considered for safety-related systems. Real-time requirements, the need for the implementation of computationally hungry algorithms and lower design costs are pushing industries in this direction. These design choices have a significant impact on the dependability capabilities and diagnostic coverage measurements. FPGAs are more and more adopted in safety-critical applications thanks to their flexibility, high parallelism and increased performances. These features have been achieved because FPGAs have been aggressively moving to lower gate length technologies. The scaling of technology has an adverse impact on the reliability of the underlying circuits in such architectures. Various different physical phenomena have been recently explored and demonstrated to impact the reliability of circuits in the form of both transient error susceptibility and permanent failures.
On November 5, 2001, a processor reset occurred on board the Microwave Anisotropy Probe (MAP), a NASA mission to measure the anisotropy of the microwave radiation left over from the Big Bang. The reset caused the spacecraft to enter a safe mode from which it took several days to recover. NASA assembled a team of engineers, including experts in radiation effects to tackle the problem. They observe that the processor reset occurred during a solar event characterized by large increases in the proton and heavy ion fluxes emitted by the sun [11]. As dimensions shrink to below 90 nm, SEEs in all devices (ASICs and FPGAs) and in all safety critical applications must be considered [65]. Section 2.1 will give an overview, from the causes to the effects, of Single Event Effects (SEE).
Apart from the increased vulnerability of the circuits to transient errors, there is an increase in the impact of different aging phenomena that result in permanent failures of the devices and interconnect. Therefore, the aggravation of such aging phenomena like Hot-Carrier Effect (HCE), Time Dependent Dielectric Breakdown (TDDB), Electromigration (EM), Thermal Cycling, Stress Migration, and Negative Bias Thermal Instability (NBTI) need to be analysed carefully in such FPGAs designed using newer technologies. Among them, NBTI has been forecast to be main aging phenomenon in future technologies and has been addressed during my research. Section 2.2 will underline NBTI root causes and device-level effects to better understand how to tackle them.

2.1 Single Event Effects

Single Event Effects (SEE) in microelectronics are caused when highly energetic particles present in the natural space environment (e.g., protons, neutrons, alpha particles, or other heavy ions) strike sensitive regions of a microelectronic circuit (see Figure 2.2). Depending on several factors, the particle strike may cause no observable effect, a transient disruption of circuit operation, a change of logic state (i.e., nondestructive SEE or soft errors), or even permanent damage to the device or integrated circuit (IC), the so called destructive SEE or hard errors [20].

Soft errors are upsets to the device operation and are self-correcting in time or are correctable

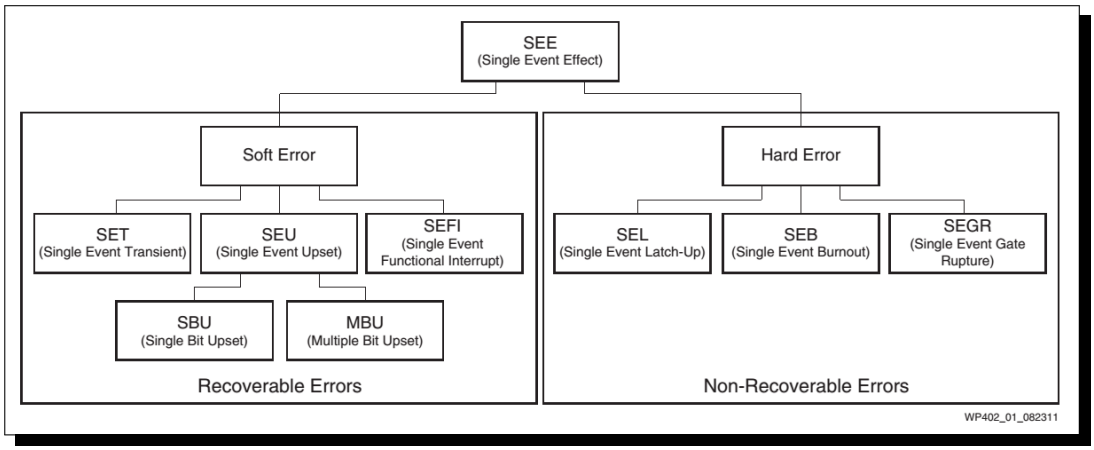


Figure 2.1: Types of Single Event Effects [65]

by rewriting a memory element, and they can be classified as:

- Single-event transients (SETs) result when a high-energy particle impacts a combinatorial path of a device and can induce a voltage/current spike. If the pulse-width of this spike is sufficient and at the right time, it can propagate through the logic;
- Single-event upsets (SEUs) are the result of high-energy particles causing a change in the state of a memory element (SRAM, flash, flop, or latch). SEUs can be categorized as single-bit or multi-bit upsets (SBUs or MBUs);
- Single-event function interrupts (SEFIs) are disruptions to normal device operation (beyond a simple corruption of user data). These types of effects alter the functionality of the circuit and typically require reconfiguration/reset or power cycling for recovery.

Errors that cause lasting damage to the device are classified as hard errors. The three subclasses of hard errors are:

- Single-event latch-up (SEL) is a circuit latch-up induced by radiation. This latch-up can be either permanent or disappear with power cycling;
- Single-event burnout (SEB) is a short-circuiting caused when a high-energy ion impacts a transistor source, causing forward biasing. SEBs are typically a threat to power MOSFETs, high-voltage diodes, and similar circuits;
- Single-event gate rupture (SEGR) is a plasma spiked caused by a high-energy ion impact, resulting in rupture of the gate oxide insulation [65].

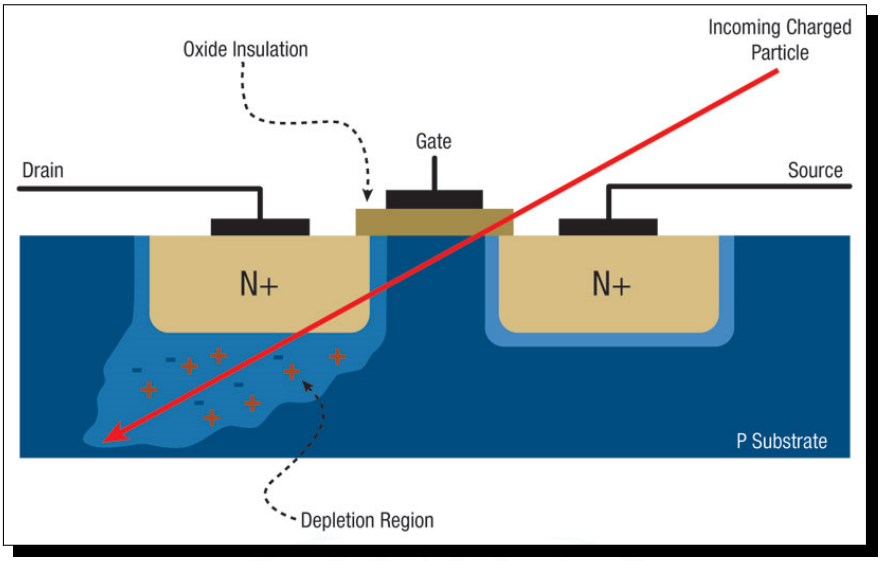


Figure 2.2: Impact of a high-energy particle in a CMOS structure [54]

Natural radiation that causes SEE in digital circuits may come from various sources. At ground level, one can distinguish two major sources of radiation:

1. *atmospheric radiation environment*: elementary particles and electromagnetic radiation are produced in the Earth's atmosphere when a primary cosmic ray (of extraterrestrial origin) enters the atmosphere. The term cascade means that the incident particle (generally a proton, a nucleus, an electron or a photon) strikes a molecule in the air so as to produce many high energy secondary particles (photons, electrons, hadrons, nuclei) which in turn create more particles, and so on (see Figure 2.3).

2. *telluric radiation sources*: Natural radioisotopes contained in the Earth's crust are the principal natural sources of α, β and γ radioactivity but only the alpha-particle emitters present a reliability concern in microelectronics. Beta and gamma processes are indeed not able to deposit a high enough amount of energy susceptible to significantly impact the microelectronic circuit operation. The presence of alpha-particle emitters in electronic devices can be classified as materials that are naturally radioactive (they contain a fraction of radioactive nuclei) or materials that contain residual trace of radioactive impurities. Currently, several types of alpha particle emitters have been identified at wafer, packaging and interconnection levels, including lead in solder bumps, uranium and thorium in silicon wafers and in molding compounds, more recently hafnium in new high-Îž gate and platinum in silicide materials [7].

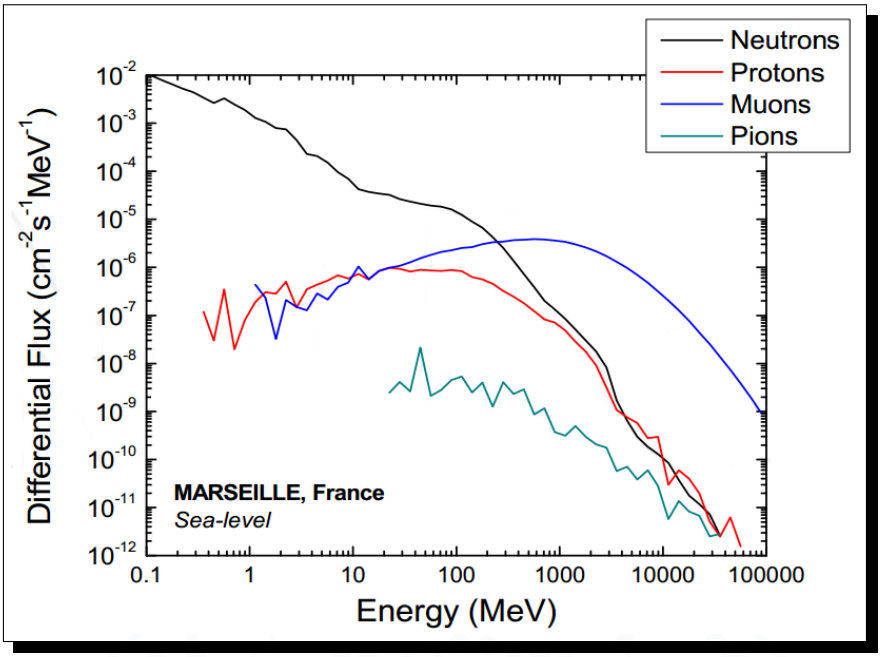


Figure 2.3: Differential flux for atmospheric neutrons, protons, muons and pions at ground level in Marseille, France [7]

Nondestructive SEE can lead to different system malfunction. From what concerns FPGA devices, particular emphasis has been established in Single (Multiple) Event Upset (SEU and MEU) in static random access memories (SRAM) and single-event transients (SET) in logic.

2.1.1 Physical causes

There are two primary methods by which ionizing radiation releases charge in a semiconductor device: direct ionization by the incident particle itself and ionization by secondary particles created by nuclear reactions between the incident particle and the device:

- *Direct Ionization*: it appears when an energetic charged particle passes through a semiconductor material it frees electron-hole pairs along its path as it loses energy. When all of its energy is lost, the particle comes to rest in the semiconductor, having travelled a total path length referred to as the particle's range. The term *linear energy transfer* (LET) is used to describe the energy loss per unit path length of a particle as it passes through a material. Direct ionization is the primary charge deposition mechanism for upsets caused by heavy ions, where we define a heavy ion as any ion with atomic number greater than or equal to two (i.e., particles other than protons, electrons, neutrons, or pions). Lighter particles such as protons do not usually produce enough charge by direct ionization to cause upsets

in memory circuits, but recent research has suggested that as devices become ever more susceptible, upsets in digital ICs due to direct ionization by protons may occur [20];

- *Indirect Ionization*: even if direct ionization by light particles does not usually produce enough charge to cause upsets, protons and neutrons can both produce significant upset rates due to indirect mechanisms. As a high-energy proton or neutron enters the semiconductor lattice it may undergo an inelastic collision with a target nucleus. These reaction products can now deposit energy along their paths by direct ionization. Because these particles are much heavier than the original proton or neutron, they deposit higher charge densities as they travel and therefore may be capable of causing an SEU. Inelastic collision products typically have fairly low energies and do not travel far from the particle impact site. They also tend to be forward-scattered in the direction of the original particle.

After a particle hits the device, the charge is then released and collected in the device structure. When a particle strikes a microelectronic device, the most sensitive regions are usually reverse-biased p/n junctions, as shown in Figure 2.4. The high field present in a reverse-biased junction depletion region can very efficiently collect the particle-induced charge through drift processes, leading to a transient current at the junction contact. Strikes near a depletion region can also result in significant transient currents as carriers diffuse into the vicinity of the depletion region field where they can be efficiently collected. Charge generated along the particle track can locally collapse the junction electric field due to the highly conductive nature of the charge track and separation of charge by the depletion region field. as shown in Figure 2.5 This funnelling effect can increase charge collection at the struck node by extending the junction electric field away from the junction and deep into the substrate, such that charge deposited some distance from the junction can be collected through the efficient drift process.

2.1.2 Single Event Transient in digital logic

A Single Event Transient (SET) is any temporary voltage disturbance that occurs in an integrated circuit following the passage of an ionizing particle through the IC. SETs occur in both digital and analog ICs (SETs in digital circuits are referred to as DSETs and those in analog circuits as ASETs). SETs occur in devices manufactured with silicon CMOS, bipolar, or BiCMOS, or III-V technology. SETs are generated at internal circuit nodes, but whether they are detected depends on a number of factors. [11] DSETs are momentary voltage or current disturbances that, although they don't cause an upset in the circuit actually struck by an energetic particle, may propagate through subsequent circuitry and eventually cause an SEU when they reach a latch or other memory element. At least four criteria must be met for a DSET to result in a circuit error:

1. The particle strike must generate a transient capable of propagating through the circuit

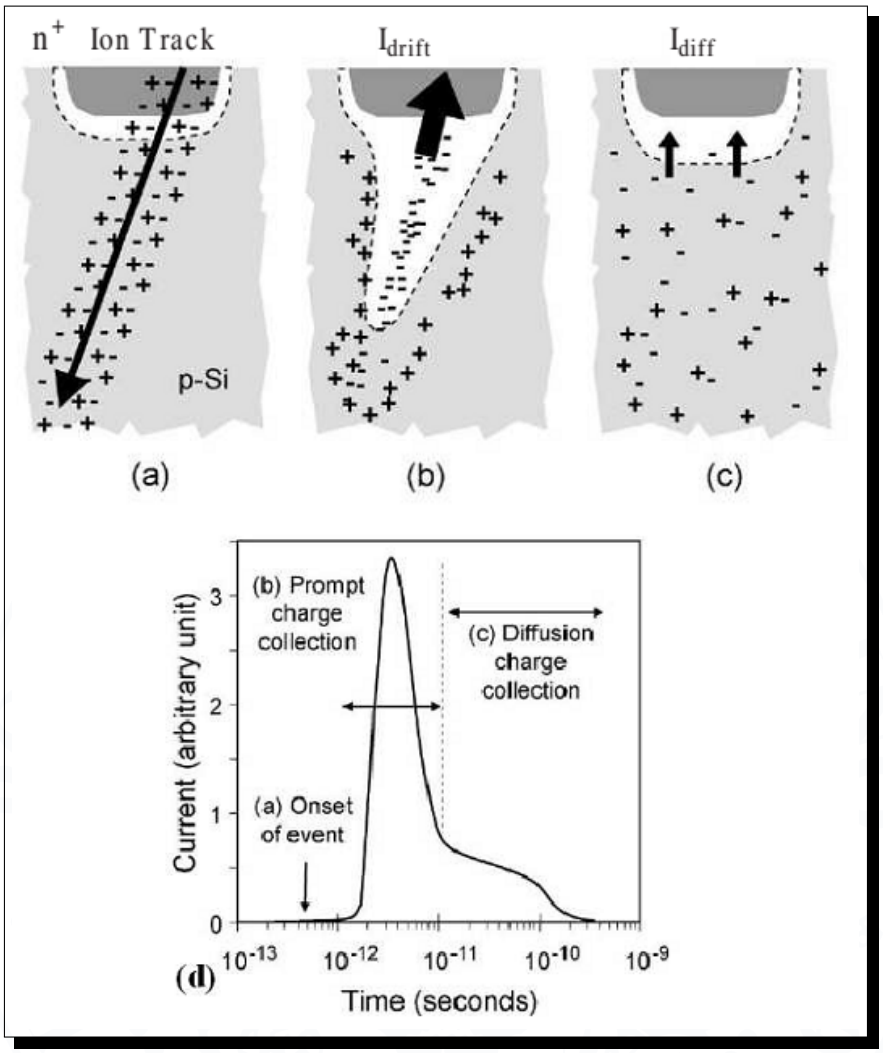


Figure 2.4: Representation of charge collection in a silicon junction immediately after (a) an ion strike, (b) prompt (drift) collection , (c) diffusion collection, (d) the junction current induced as a function of time [9]

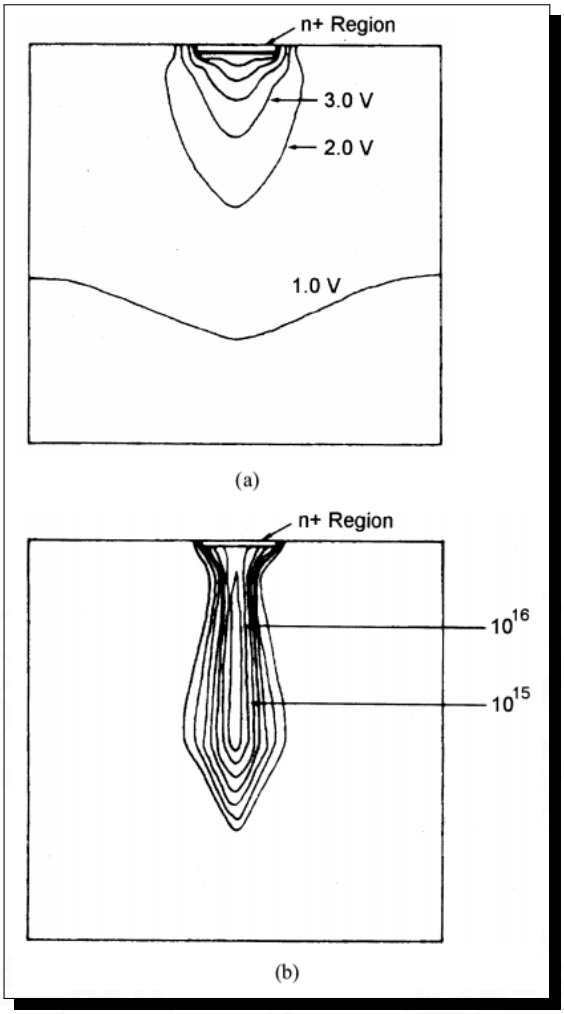


Figure 2.5: Funneling in an n+/p silicon junction following an ion strike: (a) electrostatic potential and (b) electron concentration [30]

2. There must be an open logic path through which the DSET can propagate to arrive at a latch or other memory element

3. The DSET must be of sufficient amplitude and duration to change the state of the latch or memory element

4. In synchronous logic, the DSET must arrive at the latch during a clock pulse enabling the latch

The probability that momentary glitches will be captured as valid data in combinational logic increases linearly with frequency because the frequency of clock edges increases [21]. With the

increase of circuit speeds, also the ability of a given transient to propagate increases. Nonetheless, also the duration of transients decreases. Due to both their greater ability to propagate in high-speed circuits and their higher probability of capture by subsequent storage elements such as latches, DSETs have been predicted to become endemic in deep sub-micron digital ICs. In

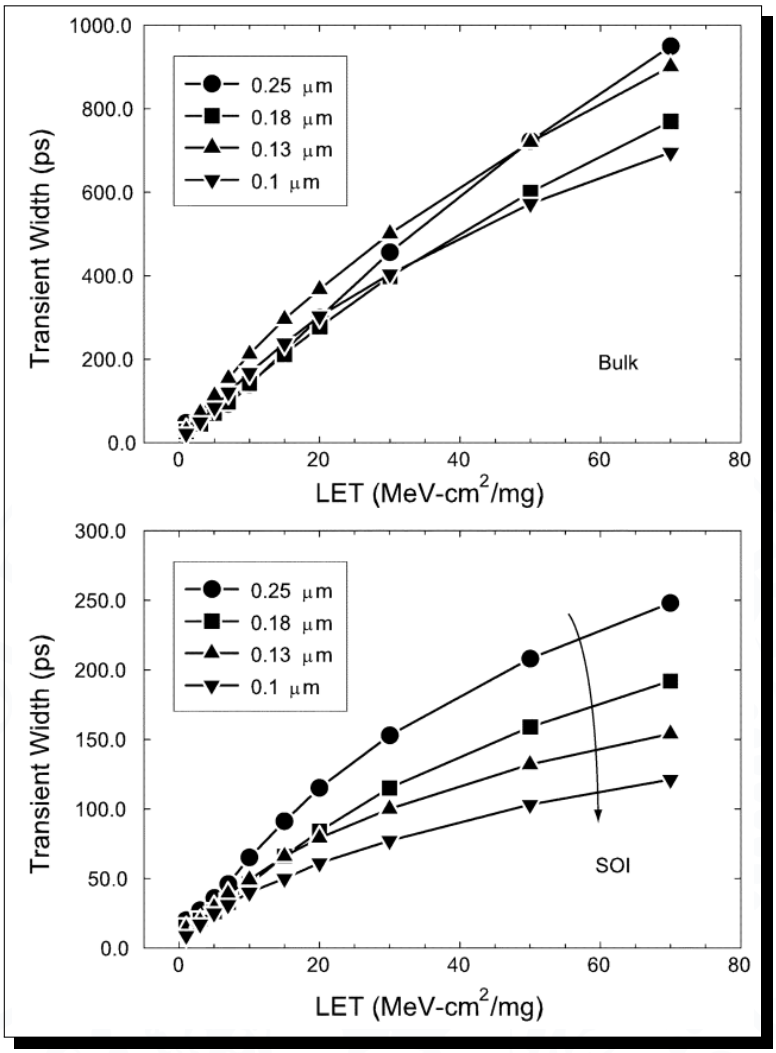


Figure 2.6: Single-event transient width in bulk and SOI devices as a function of LET and technology scaling [21]

bulk CMOS devices, a clear scaling trend emerges, namely that peak current significantly decreases with technology scaling.This decrease to be primarily due to the smaller cross-sectional area presented by the struck drain junction for diffusive charge collection (more than 6 times smaller at 0.1 m compared to 0.25 m), in combination with the higher well doping levels of the

scaled technologies (which further decrease charge collection efficiency).
The situation is different for Silicon over Insulator (SOI) technologies, where the buried oxide prevents charge collection from the substrate and therefore the size of the drain is unimportant. Figure 2.6 indicates that DSETs in SOI technologies are generally less than 250ps in duration, much faster than in the bulk devices. In addition, the SOI DSET durations decrease with technology scaling, consistent with the fact that SOI collection is dominated by fast collection within the body volume, which shrinks with gate length. These differences in transient pulse widths have significant implications for DSET propagation in bulk and SOI technologies. Once a DSET has been produced in the struck device, its evolution through the digital logic can be predicted. Research usually exploit inverter chains to understand the behaviour and the propagation of DSETs. Transients that can propagate without attenuation are obviously a concern, although it must be remembered that propagation of the DSET is only the first of four conditions that must be met for a DSET to become an SEU. The DSET must still have an open logic path to a latch, meet the timing parameters of the latch, and arrive during a sensitive timing window to become an error.

2.1.3 Single Event Upset in SRAM

Single event upsets in memory circuits are a key issue for advanced CMOS technologies. A single particle hitting a very dense device, like SRAM, can produce severe consequences.
Single-bit upsets (SBU) and multiple-bit upsets (MBU) are important in both terrestrial and space applications. With technology scaling, the number of upsets per chip will typically increase due to higher circuit density and may also increase due to higher circuit sensitivity. For SRAM circuits, MBUs are particularly important since the MBU sensitivity can limit the effectiveness of error correcting codes (ECC) [29]. Early SRAM was more robust against single event because of high operating voltages and the fact that data was stored as the state of a bi-stable circuit made up of two large cross-coupled inverters, each strongly driving the other to keep the bit in its programmed state. However, with each successive SRAM generation, reductions in cell collection efficiency due to the shrinking cell depletion volume have been swamped out by big reductions in operating voltage and reductions in node capacitance. Thus SRAM single bit Single Error Rate (SER) increased with each successive generation [8]. Figure 2.7 shows the SEU and MBU cross sections for a chosen energy of 400 MeV. As expected from nuclear physics, the behaviours of neutrons and protons are nearly identical. Experiments take advantage of this similar behaviour of neutrons and protons above 100 MeV (i.e., protons, easier to produce and control, replace neutrons).
It is seen that the increase of the MBU cross section when the critical energy is decreasing is faster than the SEU one, and that the ratio of these two cross sections exhibits large variation. Starting from three orders of magnitude, it tends to only one order of magnitude. The fast increase of the two cross sections at very low critical energy is due to the increasing participation of the light

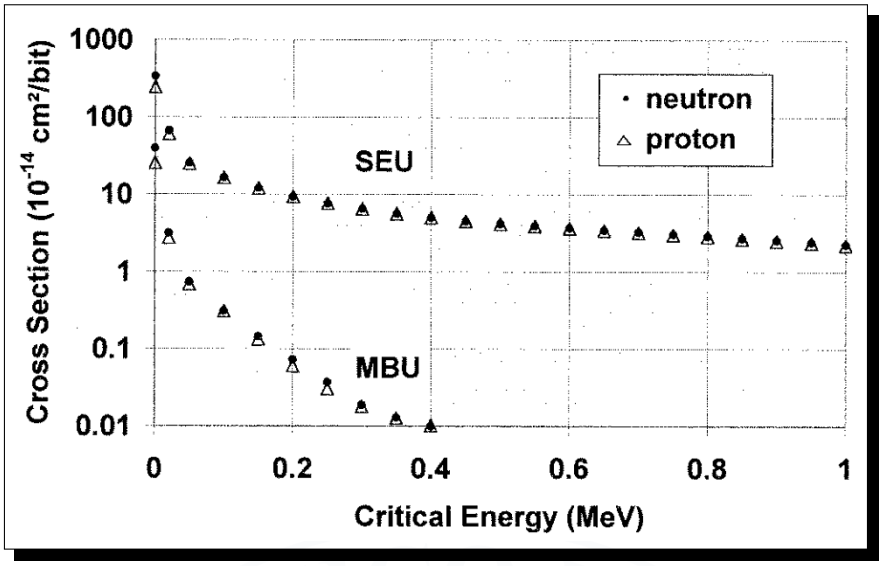


Figure 2.7: SEU and MBU cross sections versus critical energy [61]

particles. At very low critical energies, the SEU and MBU cross sections are almost governed by geometrical probabilities. The number of SEUs depends on the size of the sensitive volume and the deposited energy in this volume. As far as the MBUs are concerned, the distance between the bits is a third important characteristic. Device downscaling thus clearly induces soft error rate concerns. MBUs are particularly worrying because they are the most difficult to correct since they effects multiple bits in the same word [61].

Recent Soft Error Rate (SER) testing result of SRAM-based FPGAs from Actel [1] shows a significant and growing risk of functional failures due to the corruption of configuration data, especially when the system has higher densities. The number of upsets per 1 million gates per day increases because of the altitude dependent increase in neutron flux density. It is expected that neutron-induced soft errors get worse by a factor of two as we move from 0.13μ to 0.09μ technology.

The radiation induced soft errors have become one of the most important and challenging failure mechanisms in modern electronic devices. SER of commercial chips is controlled to within 100-1000 FITs. Compared to most hard failure mechanisms that produce failure rates on the order of 1-100 FIT, the SER of a low-voltage embedded SRAM can easily be 1000 FIT/Mbit. Therefore, a four-phase approach to deal with them is in progress [58]

1. Methods to protect chips from soft errors (prevention).
2. Methods to detect soft errors (testing).
3. Methods estimate the impact of soft errors (assessment).
4. Methods to recover from soft errors (recovery)

2.2 NBTI: Causes and Effects

Bias temperature instability (BTI) is a degradation phenomenon occurring mainly in MOS Field Effect Transistors (MOSFETs).
Even though the exact root causes of the degradation are not yet well understood, it is now commonly admitted that under a constant gate voltage and an elevated temperature a build up of positive charges occurs either at the interface Si/SiO2 or in the oxide layer leading to the reduction of MOSFET performances. Nevertheless, this degradation remained marginal for many years, but with aggressive scaling of MOS transistors this phenomenon became important.
In order to improve the transistors performances, nitrogen atoms were introduced into the oxide layer by different nitridation processes, but mostly by thermal annealing. This nitridation step is intended both to give a better control on the gate leakage current and to avoid the boron atoms, used to dope the poly-silicon gate, to flow through the oxide into the substrate.
Besides, the general trend was that most of the devices turned out to be surface-channel devices instead of buried-channel ones in recent technologies to counter the short-channel effects inherent of the downscaling process and to improve the performances.
As a consequence of both the introduction of the nitridation process step and the use of surface-channel devices, researchers ascribed an enhanced BTI-like degradation of p-MOSFETs under negative bias and elevated temperatures, the so-called **NBTI effect**.
By definition, bias temperature instabilities are observed when either a capacitor or a transistor is stressed at relatively high temperatures (typically ranging from $80°C$ to $150°C$) under a low and constant gate voltage while the source/drain and well electrodes are grounded.
The symmetry of stress conditions along the channel proves that this degradation is not related to channel carrier transport. Generally, it is observed that after an NBTI stress, the saturated drain current value (I_{dsat}) is reduced. The forward and reverse values of the saturated drain current both degrade identically, demonstrating the symmetry of the stress. By the same time, the threshold voltage (V_{th}) is increased as well as the S/D series resistance.
Altogether, the degradation of these parameters demonstrates the build-up of positive charges close or at the interface of $Si - SiO_2$ and yields to a lower level of performance for the transistor. For short stress times, the off-leakage current is decreasing due to the shift of the I_d/V_g curves linked to the threshold voltage shift. Nevertheless, in some cases, an increase of the GIDL observed at low fields may overcome this decrease and limit the performances of the circuit by an increased consumption.
The instabilities exist in most of the configurations, for either p-MOSFETs or n-MOSFETs, and whatever a negative bias and/or a positive bias is applied, except for the n-MOSFET under positive bias, which does exhibit almost no degradation. Nevertheless, as shown in Figure 2.8 , applying NBTI stress conditions (i.e. negative gate voltage) on p-MOSFETs represents the most degrading case.

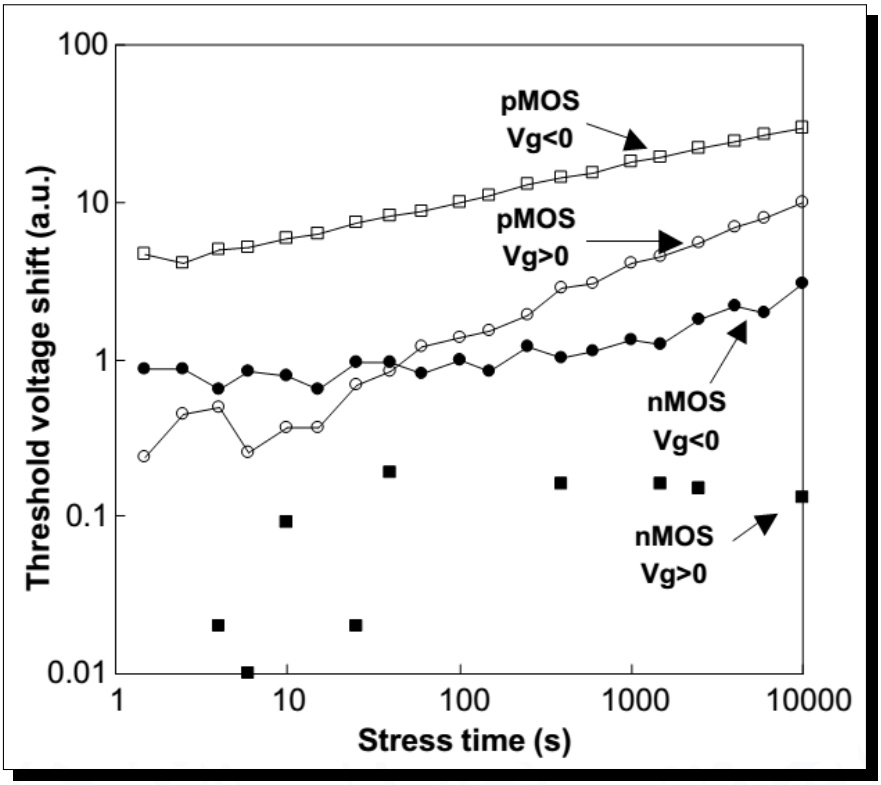


Figure 2.8: Vth shift as a function of stress time [31]

So far, the microscopic details of the NBTI degradation are not clearly understood but there is a general agreement to say that there is generation of traps at the $Si - SiO_2$ interface during negative BTI aging.

But, to understand the impact of the NBTI degradation up to circuit level, it is important to make the link with device parameters such as threshold voltage. As NBTI degradation is mainly a build-up of charges at the interface in a symmetrical configuration along the channel, the threshold voltage parameter is more relevant to describe the degradation than other parameters such as the saturated drain current.

As already discussed above, the NBTI degradation is known to be activated with temperature. For many researchers, the degradation of electrical parameters is mostly linked to the creation of interface traps, especially in ultra thin oxides.

Figure 2.9 shows the temperature dependence of the interface traps creation and the threshold voltage shift for a given set of stress conditions and varying temperatures. If the threshold voltage is only influenced by the creation of interface traps at the interface, these two parameters should have similar apparent temperature activations.

If the threshold voltage shift would have been explained solely by the interface traps creation,

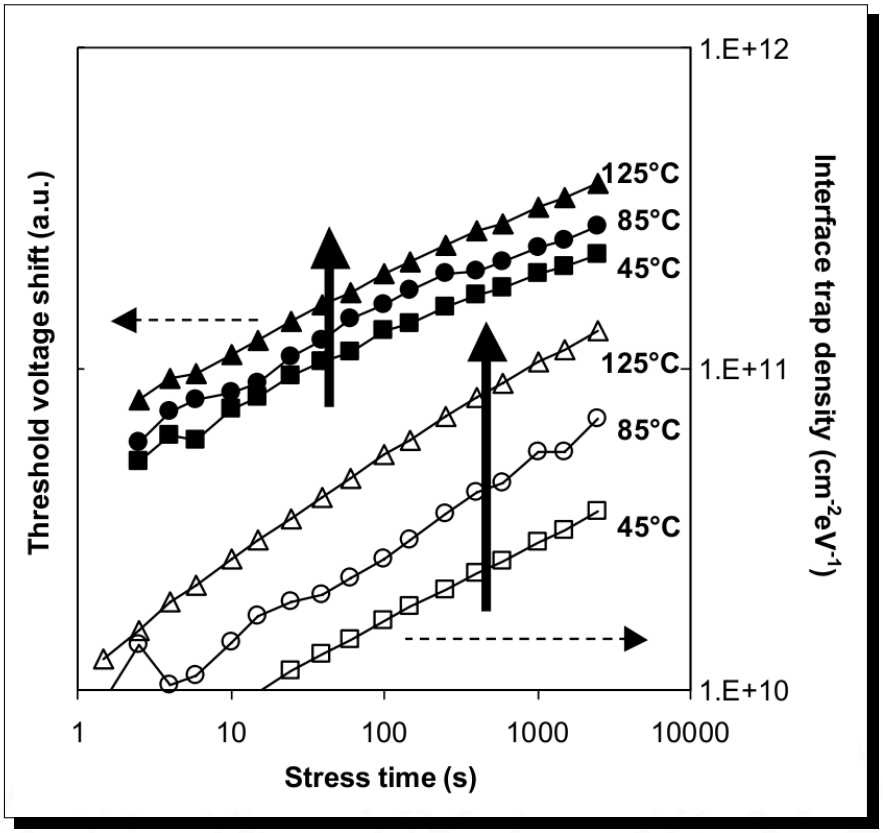


Figure 2.9: Vth shift and interface traps creation as a function of stress time for three different temperatures[31]

similar activation energies would have been found for both phenomena. Different activation energies imply that another process has to be taken into account in order to explain the threshold voltage shift. It is reasonable to think that this second component is related to charges into the oxide.

Effectively, under a constant voltage stress such as under NBTI stress conditions, an increase of the number of oxide defects is known to occur, which leads the oxide to break once a critical limit is reached. But defects can also be present in the oxide previously to the stress, induced either by impurities or strain relaxation during the latter process phases.

In this case, applying an oxide field would increase the trapping efficiency of charges in those defects. Thus, the second component of the NBTI degradation could then be either holes trapped onto oxide defects, pre-existent to the stress or created during the stress, or slow states present close to the interface.

Applying a positive gate bias consecutively to a negative bias stress is known to neutralize both trapped holes and/or positively charged slow states.

In order to determine the relative importance of the interface traps and the positive charges into the oxide on the threshold voltage, an electrical neutralization phase, where the device is positively gate biased, is introduced after a normal, continuous negative gate bias stress.
The relative shifts for the interface trap density (open symbols) and for the threshold voltage (filled symbols) versus the stress time are presented in Figure 2.10.
The main feature is a strong reduction (more than 50%) of the threshold voltage under the posi-

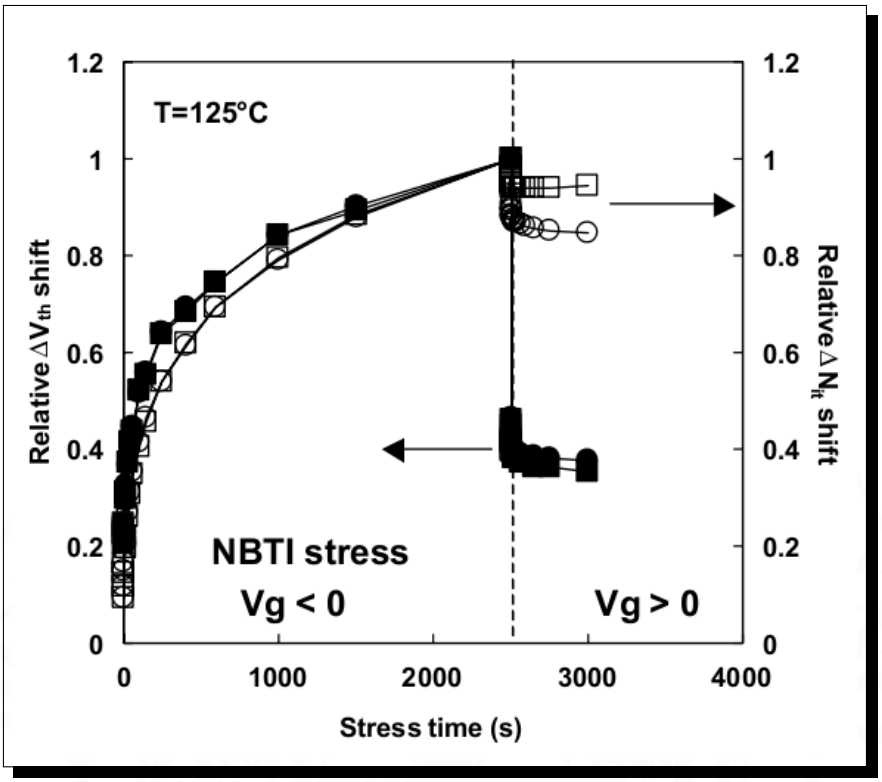


Figure 2.10: Relative shifts for traps and Vth versus stress time [31]

tive bias, when the interface trap density remains almost unchanged. First, the absence of a large variation of the trap density under the positive bias is worth noticing. The possible repassivation of the Si dangling bonds, as the hydrogen species might move back to the SiO_2/Si interface when the bias polarity is changed, appears to be negligible. This result, in agreement with the results obtained by adding additional delay between stress and measurement as described above, is another clue that the threshold voltage does not depend only on the interface trap density and the recovery effect should not be addressed with a diffusion controlled annealing model.
Consequently, the threshold voltage shift that consists of the difference of the threshold voltage shifts before and after the electron injection represents the contribution of positive charges induced during the stress by hole trapping into the oxide.

This shift presents almost no temperature dependence, which is expected for defect generation by hole trapping as the tunnelling effect involved in the charging is hardly temperature dependent. The small temperature dependence observed experimentally is only to be explained by a variation of the effective hole capture cross-section with temperature.
The positive charges build-up within the oxide observed experimentally can still be explained either by holes trapped in oxide defects or by the charging of slow states. Both trapped holes and positively charged slow states can be neutralized by a further electron injection under positive bias for few minutes as in Figure 2.10. Though, they should present a different recharging behavior when the slow states can be recharged under a small negative bias whereas holes trapped into the oxide cannot, as they need larger energies to get back into the oxide and to restore the positive charge.

2.2.1 Effects on digital logic

NBTI occurs under negative gate voltage (e.g., $V_{gs} = -V_{DD}$) and is measured as an increase in the magnitude of threshold voltage. It mostly affects the PMOS transistor and degrades the device drive current, circuit speed, noise margin, and the matching property. Indeed, as gate oxide gets thinner than 4nm, the threshold voltage change caused by NBTI for the PMOS transistor has become the dominant factor to limit the life time, which is much shorter than that defined by hot-carrier induced degradation (HCI) of the NMOS transistor.
For a PMOS transistor, there are two phases of NBTI, depending on its bias condition. These two phases are illustrated in Figure 2.11, assuming the substrate is biased at VDD. In Phase I, when $V_g = O$ ($V_{gs} = -V_{DD}$), positive interface traps are accumulating.

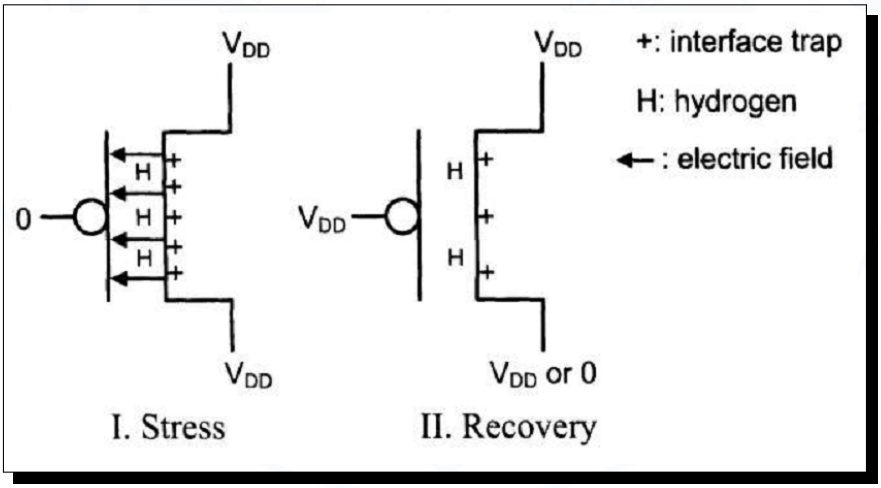


Figure 2.11: Stress and recovery phases on transistor [57]

This phase is usually referred as "stress" or "static NBTI".
In Phase II, when $V_g = V_{DD}$ ($V_{gs} = O$), holes are not present in the channel and no new interface traps are generated, instead, H diffuses back and anneals the broken Si-H. As a result, the number of interface traps is reduced during this stage and the NBTI degradation is recovered. Phase II is usually referred as "recovery" and has a significant impact on the estimation of NBTI during the dynamic switching.

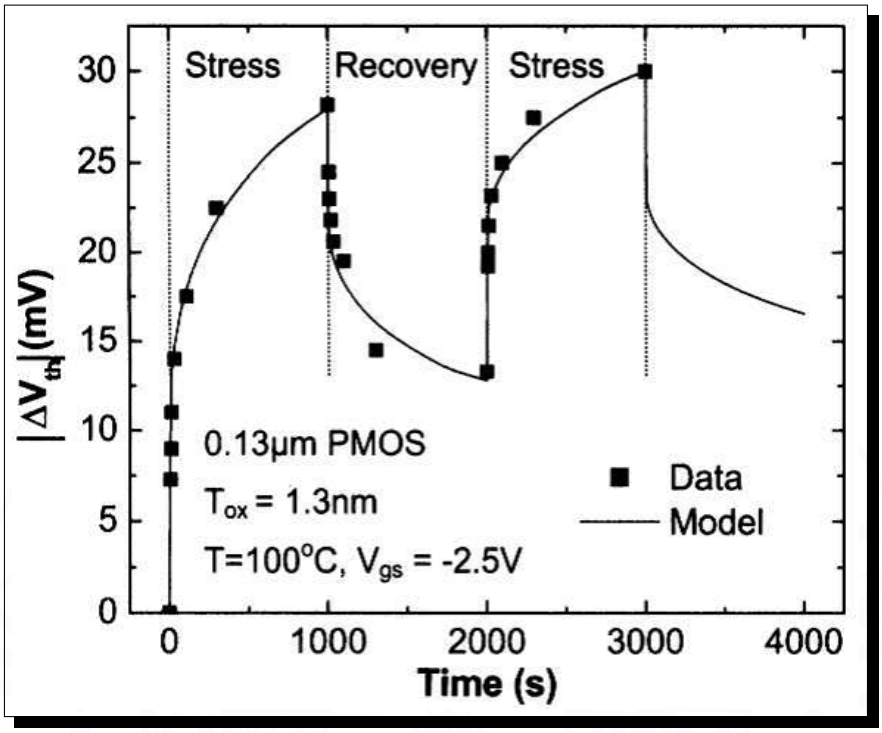


Figure 2.12: Threshold voltage degradation during stress and recovery phases [57]

In a realistic circuit, the gate switches between 0 and V_{DD}. For a PMOS transistor, the condition of $V_g = V_{DD}$ removes NBTI stress and anneals interface traps. The degradation of V_{th} on the PMOSFET devices causes changing on the behavior of the gates. First of all, NBTI degrades the drain current of the transistor, and this causes a decrease of rising time of the gates.
The degradation in delay shows a power dependency to time with a fixed exponent of 1/6. Several method are studied at gate-level to decrease the delay degradation, in example transistor resizing of only the pull-up side of the gate, sleep mode with lower V_{DD}, power-gating like V_{DD} clamping with virtual supply voltage etc.
The drain or on current is important in analog and digital circuits. In digital circuits, with MOSFETs being switches, charging and discharging capacitors, higher drain current leads to faster capacitor charging and higher frequency operation.
The delay time is

$$t_d = \frac{C|V_{DD}|}{I_D} = \frac{2LC}{W\mu_{eff}C_{OX}(V_{DD} - V_{th})^2} \tag{2.1}$$

where C is the capacitance and V_{DD} the supply voltage [59].
NBTI stress leads to l_{eff} reduction and V_{th} increase, both giving delay time increases. As we can see in the equation, the circuit impact will be greater for lower operating voltage, because of the reduced $V_g - Vth$, so the impact is predicted to increase in new technology nodes.

2.2.2 Effects on SRAM memories

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 *Static Noise Margin* (SNM), i.e., the minimum *DC* noise voltage necessary to change the stored value [12].
Figure 2.13 above shows a 6T cell with the wordline, bitlines , and internal nodes. During a read operation, the bitline $\overline{BL}$ is discharged through the access transistor and the pull down transistor which causes the voltage at node Q to rise. In order for the cell to maintain its state, it is necessary that the voltage at Q not rise above the trip point of the inverter formed by $M3$ and $M4$. The voltage difference between this inverter trip point and the voltage at node Q is the read SNM of the cell.
SNM can be computed as the length of the side of a maximum square nested between the two voltage transfer characteristic (VTC) curves (i.e., for each back-to-back inverters) of SRAM cell. Simple simulation has shown that SNM of an SRAM cell can degrade with time under NBTI [37]. SNM of SRAM is particularly important during two operations, that is, the HOLD (i.e., stand-by mode) and READ phases. When the word line is turned off and cell is holding the data, HOLD SNM can be computed. During the HOLD phase the cell state is relatively insensitive to the variation in the threshold of PMOS transistors due to NBTI.
The smaller the SNM is, the lower the reliability of the cell becomes. Unfortunately, the V_{th} shift

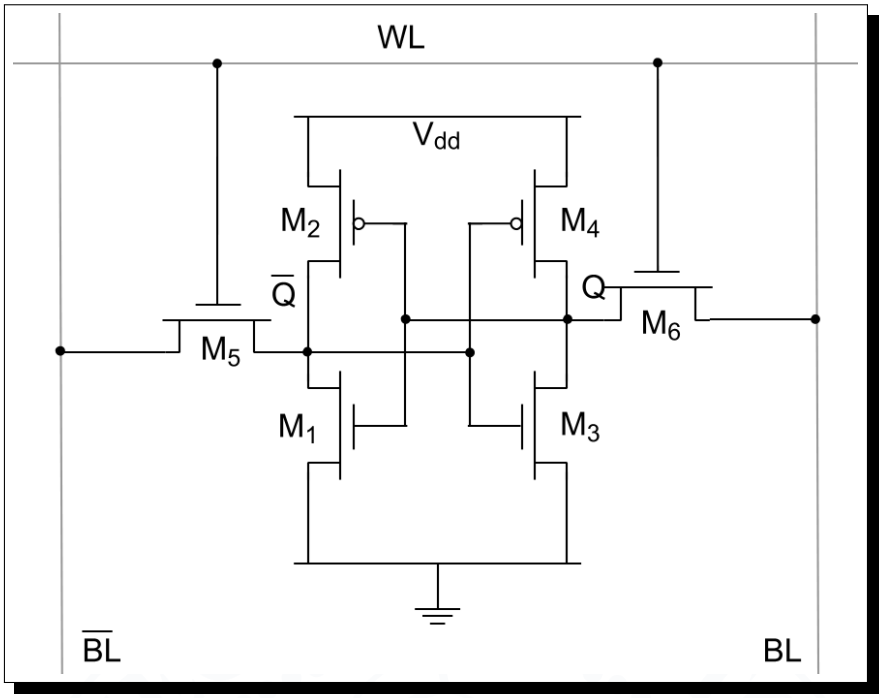


Figure 2.13: 6 Transistor SRAM cell

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.
Figure 2.14 depicts the relationship between the read failure probability and the signal probability. As predicted, failure probability becomes higher when the signal probability is unbalanced.

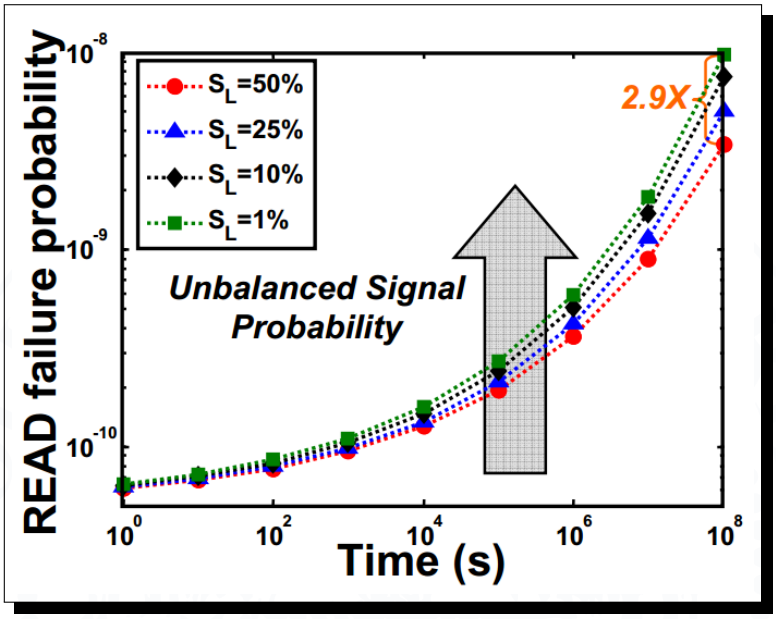


Figure 2.14: Read failure probability with different static probablitye s [37]

You do not really understand
something unless you can explain it
to your grandmother.

Albert Einstein

CHAPTER 3

FPGA ARCHITECTURE AND DYNAMIC PARTIAL RECONFIGURATION

Field Programmable Gate Arrays (FPGAs) are integrated circuits designed to be configured by the user (hence field-programmable) one or multiple times. Configuration is generally specified using a hardware description language that is an input to synthesis tool that creates a binary configuration file. This file can be loaded through a configuration port to the FPGA device. This semiconductor device is thus not restricted to a specific hardware function, but the user can freely implement any logical function that can be performed by an ASIC, by properly configuring the FPGA. Designers must then properly evaluate when to adopt an FPGA in their design or rather select an ASIC-based design flow.
FPGA based systems are usually characterized of:

- low non recurrent engineering, due to low development and tool costs;
- reconfigurability and rapid prototyping, allowing to verify and validate hardware implementation in a fast and accurate way ;
- simpler and faster design cycle, since tools are managing placing, routing in a fast and accurate way allowing the faster TTM as well;
- limited resources, fixed according to the chosen FPGA device;
- higher power consumption.

On the other hand, ASIC systems are usually characterized by:

- lower recurrent engineering, since for high volume designs, unit costs are very low;
- higher performances, in terms of speed and power efficiency;

- high design flexibility, without any limit in number and kind of resources;
- very difficult design and long design time, due to floor planning, routing that must be checked and validated.

At the highest abstraction level and at the earliest design phase, it basically comes down to a scalability versus a flexibility question. ASICs are advantageous when it comes to high port density applications. FPGAs are advantageous when it comes to feature velocity with a shortened time-to-market requirement.
Clearly, in the time-to-market driven era, FPGA market share is constantly increased, with 3 major players sharing the big pot: Xilinx, Altera and Actel, as shown in Figure 3.1.
In the following, Section 3.1 will highlight the different FPGA families currently available in the

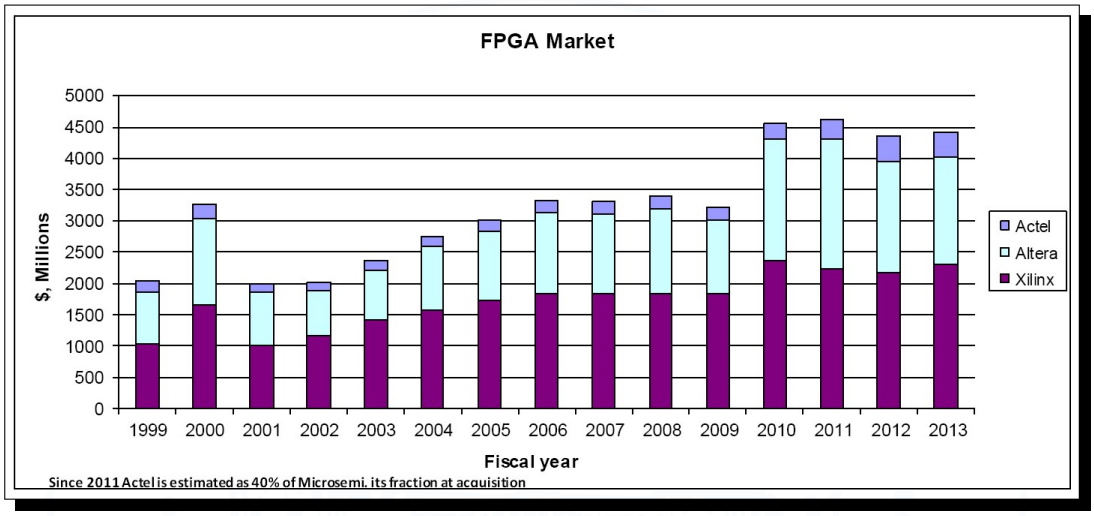


Figure 3.1: FPGA market share in recent years

market, while Section 3.2 will focus on the most widely adopted FPGA type, the SRAM-based FPGAs that allows the adoption of Dynamic Partial Reconfiguration, carefully explained in Section 3.3.

3.1 FPGA Families

A first distinction of FPGAs can be made based on where configuration is stored on-chip. FPGA's configuration can be either volatile or non volatile, and one have advantages respect to other. Modern FPGAs adopt one of these three methods to storage configuration:

- **FLASH-based**: flash-based FPGAs use flash memory as a primary resource for configuration storage, and doesn't require SRAM; this technology has an advantage of being less power consumptive and is also more tolerant to radiation effects;

- **ANTIFUSE-based**: antifuse-based FPGAs can be programmed only on "burned" to conduct current. The antifuses are normally open circuit and, when programmed, form a permanent, passive, low-impedance connection, leading to the fastest signal propagation. They provides excellent protection against design pirating and cloning;

- **SRAM-based**: configuration data is stored in Static RAM memory cells. Since SRAM is volatile and can't keep data without power source, such FPGAs must be configured upon start-up. The SRAM cells maintaining configuration require about 6 to 7 MOS per connection; these extra transistors take up extra silicon and increase area. Moreover the external memory needed to load configuration data on the internal SRAM requires extra board space which increases board and component cost to the overall system. The main advantage is the possibility to re-program FPGA at any time and configuration time is smaller than other solutions such Flash-based.

Table 3.1 lists the main features of the three different main FPGA families, comparing them and assessing their main features.

Table 3.1: Comparison among FPGA families

Feature	**Antifuse-based**	**Flash-based**	**SRAM-based**
Volatility	Non-volatile	Non-volatile	Volatile
Technology	Previous nodes	Previous nodes	State-of-art
Re-programmability	No	Yes, but limited	Yes, infinite
Area occupation	Low	Medium	High
Programmability Speed	Slow	NA	Fast
Security	High	High	Limited
Power Consumption	Low	Medium	Medium
Rad Hardness	Yes	No	No

Further then being distinguished according to the configuration methodology, there are two main architectures for FPGAs internal architecture, namely fine-grained and coarse-grained. A fine-grained architecture consists of a large number of small logic blocks (e.g. transistors, small macro cells) that allow the user to configure each block to act as any 3-input function, such as a primitive logic gate (AND, OR, NAND, etc.) or a storage element.
A coarse-grained architecture consists of a smaller number of larger and more powerful logic blocks, such as flip-flops and Look-Up Tables. A fine-grain array has many configuration points

to perform very small computations, and thus requires more data bits during configuration. The fine-grain programmability is more amenable to control functions, while the coarser grain blocks with arithmetic capability are more useful for data-path operations. An important consideration with regard to architectural granularity is that fine-grained implementations require a relatively large number of connections to connect small blocks. In coarse-grained devices the amount of connections into the blocks decreases compared to the amount of functionality they can support. This is important because interconnections are usually the major cause of signal delays and will strongly affect design timing performances.

3.1.1 Antifuse-based FPGAs

As aforementioned, antifuse-based FPGAs can be programmed only once by "burning" the connections as required at design phase. Figure 3.2 shows how connections are available in the first use of the FPGA device. All the possible links are actually left unconnected. In order to program

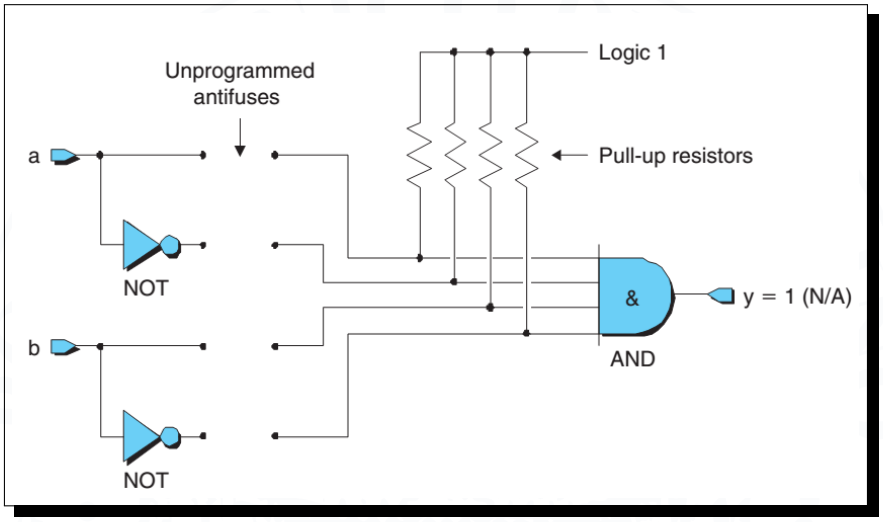


Figure 3.2: Unprogrammed antifused connections [44]

the FPGA device at performing the needed function, some of the available links are set up, by applying pulses of relatively high voltage and current to the device's inputs. This process will convert the insulating amorphous silicon in conducting poly-silicon. After the device has been properly programmed, it will look like the one pictured in Figure 3.3

Systems implemented exploiting such technology will feature relatively high speed and low power requirements. The primary advantage of anti-fuse programming technology is its low area. With metal-to-metal anti-fuses, no silicon area is required to make connections, decreasing the area overhead of programmability.

However, this decrease is slightly offset by the need for large programming transistors that supply

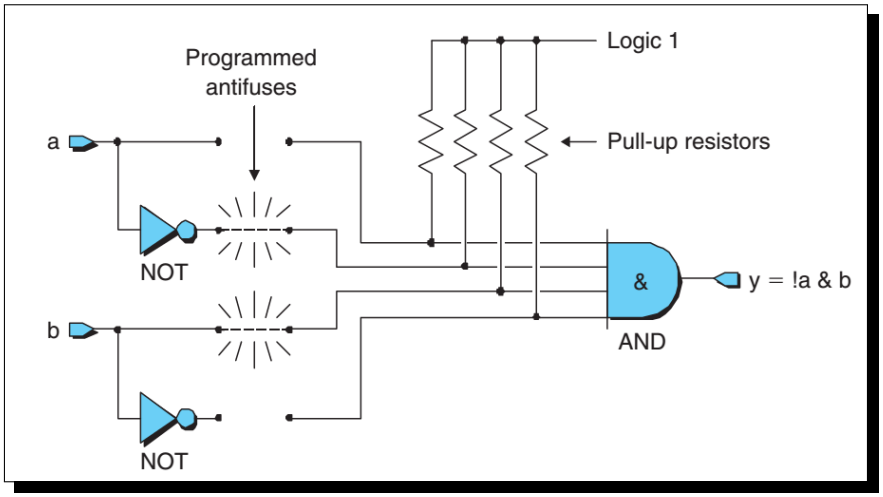


Figure 3.3: Programmed antifused connections [44]

the large currents needed to program the antifuse.
On the other hand, the achieved level of flexibility is really low, since they can be programmed only once. Furthermore, since anti-fuse-based FPGAs require a non-standard CMOS process, they are typically well behind in the manufacturing processes that they can adopt compared to other technologies. Furthermore, the fundamental mechanism of programming, which involves significant changes to the properties of the materials in the fuse, leads to scaling challenges when new IC fabrication processes are considered [40]. The main field in which these devices are used is the military one, either thanks to the intrinsic rad-hardness (they are relatively immune radiation effects like SEEs) and high level of security (their configuration data is buried deep inside them, making it almost impossible to reverse-engineer the design [44]).

3.1.2 Flash-based FPGAs

Several families of devices use Flash memory to hold FPGA configuration information. Flash memory is non-volatile array and can only be written a finite number of times, since it worn-out quite fast. The non volatility of Flash memory means that the data written to it remains when power is removed.
Flash memories technology is based on the use of floating gate transistor, as shown in Figure 3.4. In its unprogrammed state, the floating gate is uncharged and doesn't affect the normal operation of the control gate. In order to program the transistor, a relatively high voltage is applied between the control gate and drain terminals. This causes the transistor to be turned hard on, and energetic electrons force their way through the oxide into the floating gate in a process known as hot (high energy) electron injection. When the programming signal is removed, a negative charge

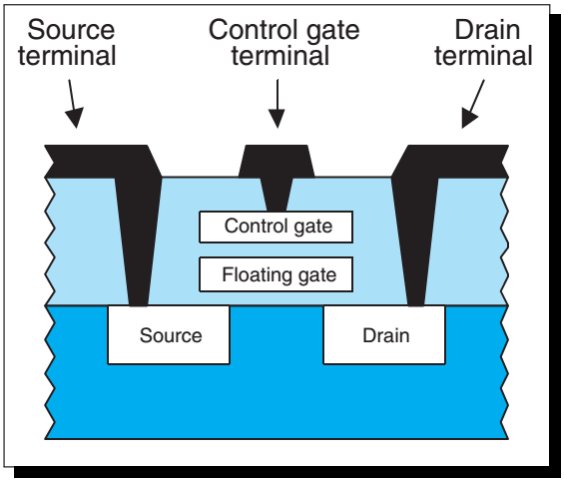


Figure 3.4: Floating Gate transistor [44]

remains on the floating gate. This charge is very stable and will not dissipate for more than a decade under normal operating conditions [44].

The programming circuitry, such as the high and low voltage buffers needed to program the cell, contributes an area overhead not present in other FPGA devices. However, this cost is relatively modest as it is amortized across numerous programmable elements. In comparison to antifuse, an alternative non-volatile programming technology, Flash-based FPGAs are reconfigurable and can be programmed without being removed from a printed circuit board.

The stored charge on the floating gate inhibits the normal operation of the control gate and, thus, distinguishes those cells that have been programmed from those that have not. This means we can use such a transistor to form a memory cell. Drawbacks to using Flash memory to store FPGA configuration information stem from the techniques necessary to write to it. As mentioned, Flash memory has a limited write cycle lifetime and often has slow write speeds. The number of write cycles varies by technology, but is typically hundreds of thousands to millions. Additionally, most Flash write techniques require higher voltages compared to normal circuits, so they require additional off-chip circuitry or structures such as charge pumps on-chip to be able to perform a Flash write. Another disadvantage of flash-based devices is that they cannot be reprogrammed an infinite number of times, due to charge build-up in the oxide that eventually prevents a flash-based device from being properly erased and reprogrammed.

On the other hand, they do not require extra storage or hardware to program at boot-up. In essence, a Flash-based FPGA can be ready immediately [28].

3.2 SRAM-based FPGAs

SRAM memory are the most widely adopted storage element for current FPGA technologies, since it provides fast and infinite reconfigurations at very low price. This is due to the fact that SRAM cells are made with the same CMOS technology on FPGA logic, thus simplifying the production process.
On the other hand, SRAM means that, once a value has been loaded into an SRAM cell, it will remain unchanged unless it is specifically altered or until power is removed from the system. So, configuration data will be lost when power is removed from the system, so these devices always have to be reprogrammed when powered on, thus requiring another non-volatile memory array to store configuration information to be loaded at start-up. Furthermore, if compared with the other aforementioned technologies, SRAM cell is large and dissipates significant static power because of the leakage current.
Altera Stratix or Xilinx Virtex families are examples of coarse-grained SRAM-based FPGAs. More specifically, Xilinx and Altera FPGAs are modular tile-based devices. No matter which device is chosen, the FPGA consists of the same basic building blocks tiled over and over again. In Xilinx FPGAs they are called CLBs (i.e., Configurable Logic Blocks), while in Altera devices are named ALM (i.e., Adaptive Logic Modules).
Altera FPGAs internal architecture is made up of logic array blocks, called LABs. LABs are made up of ALMs that can be configured to implement logic, arithmetic, and register functions. Each LAB consists of ten ALMs, various carry chains, shared arithmetic chains, control signals, a local interconnect, and register chain connection lines. The local interconnect transfers signals between ALMs in the same LAB. The direct link interconnect enables the LAB to drive into the local interconnect of its left and right neighbours. Register chain connection transfers the output of the ALM register to the adjacent ALM register in the LAB [4]. Internally, each ALM features 8-inputs with four registers, as shown in Figure 3.5.
Figure 3.6 shows the internal architecture of a CLB, adopted in Xilinx devices. that is the smallest piece of logic from the perspective of the interconnect structure. In the most recent Xilinx technology, each slice contains 4 Look Up Tables (LUT) and 8 flip-flops; only some slices can use their LUTs as distributed RAM. Each LUT can be configured as either one 6-input LUT (64-bit ROMs) with one output, or as two 5-input LUTs (32-bit ROMs) with separate outputs but common addresses or logic inputs [73]. Each LUT output can optionally be registered in a flip-flop. Four such LUTs and their eight flip-flops as well as multiplexers and arithmetic carry logic form a slice, and two slices form a configurable logic block (CLB) (see Figure 3.7). Four of the eight flip-flops per slice (one per LUT) can optionally be configured as latches. The slice also contains extra multiplexers (MUXFx and MUXF5) to allow a single slice to be configured for wide logic functions of up to eight inputs. A handful of other gates provide extra functionality in the slice, including an XOR gate to complete a 2-bit full adder in a single slice, an AND gate to improve multiplier imple-

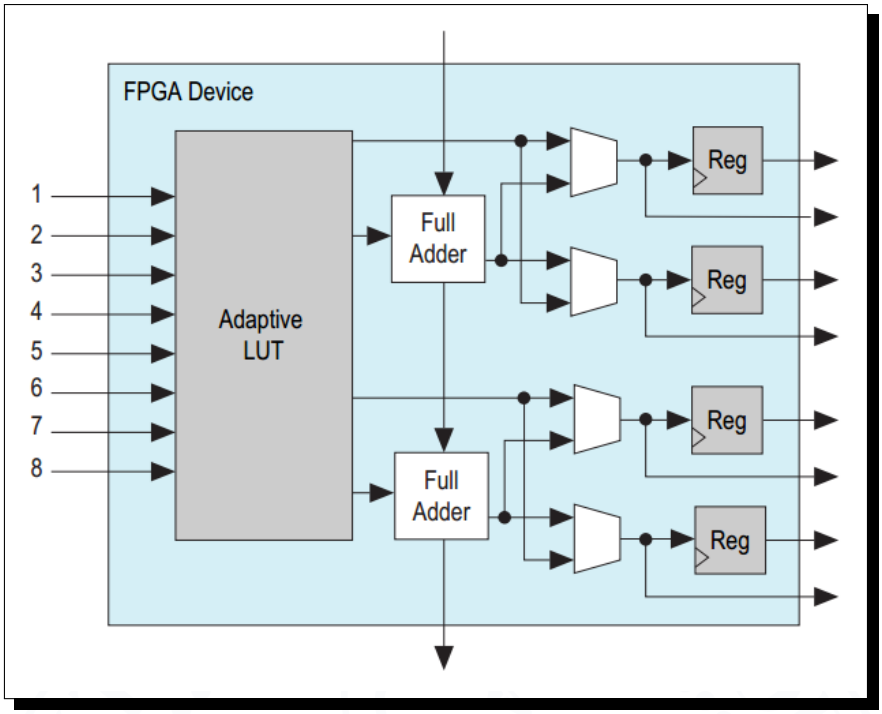


Figure 3.5: Adaptive Logic Modules architecture [4]

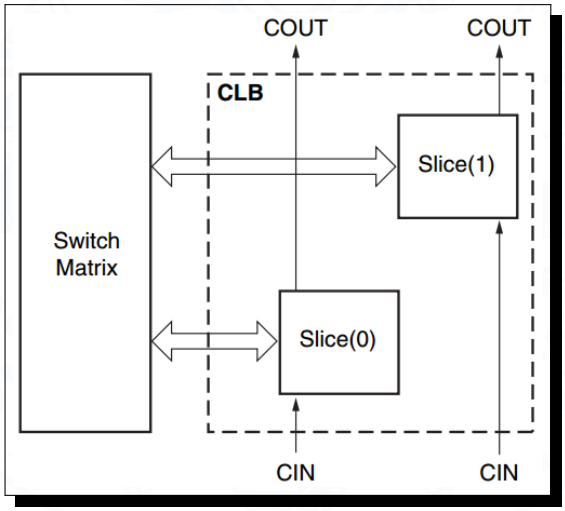


Figure 3.6: Configurable Logic Block architecture [69]

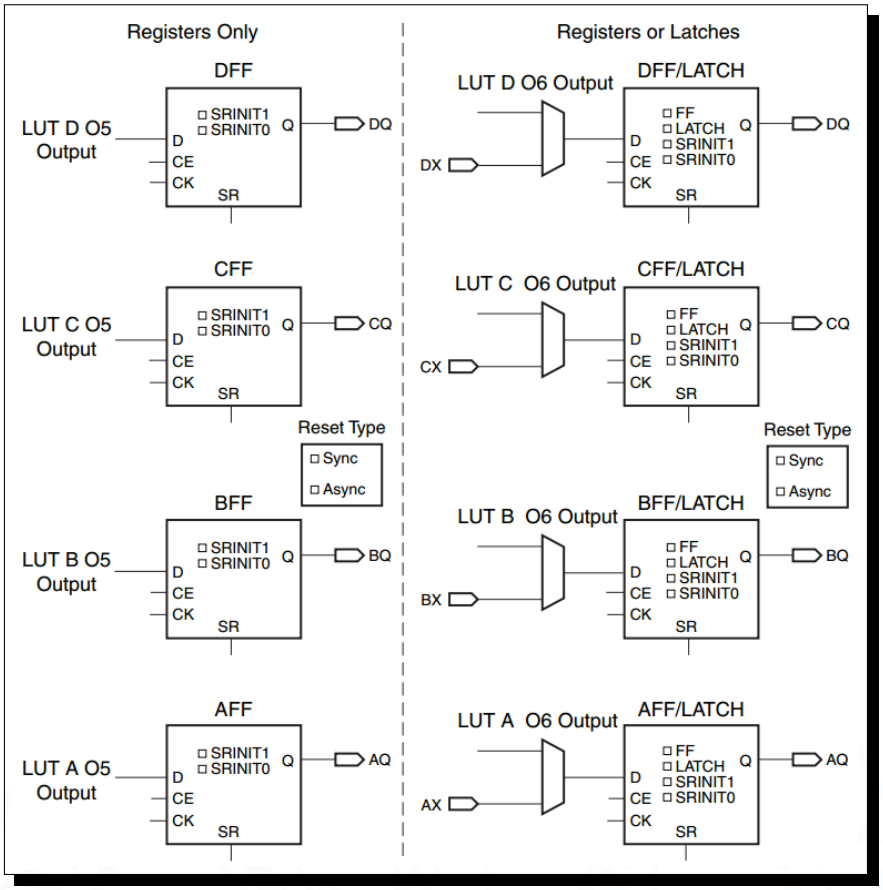


Figure 3.7: Slice architecture [69]

mentations in the logic fabric, and an OR gate to facilitate implementation of sum-of-products chains. Each slice connects to the general routing fabric through a configurable switch matrix and to each other in the CLB through a fast local interconnect.
Modern SRAM-based FPGAs, further then CLB or ALM, features special purpose architectural building blocks to enhance overall performances. These building blocks are designed to give users special bricks, like Block RAMs, clocking technology, DSP slices, SelectIO technology (for Xilinx devices), to perform ad-hoc tasks at relatively high performances.

3.3 Dynamic Partial Reconfiguration

SRAM-based FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design, as pointed out in Section 3.2. In the

aforementioned technology, Dynamic Partial Reconfiguration (DPR) takes this flexibility one step further, allowing the modification of an operating FPGA design by loading a partial configuration file, usually referred as partial bitstream. While a full bitstream file is intended to configure the whole FPGA, thus writing the information all along the configuration memory, partial BIT files can be downloaded to modify reconfigurable regions in the FPGA, only. This process is carried out without compromising the integrity of the applications running on those parts of the device that are not being reconfigured, both the static portion of the FPGA and the possible other reconfigurable modules that are not undergoing the reconfiguration process.

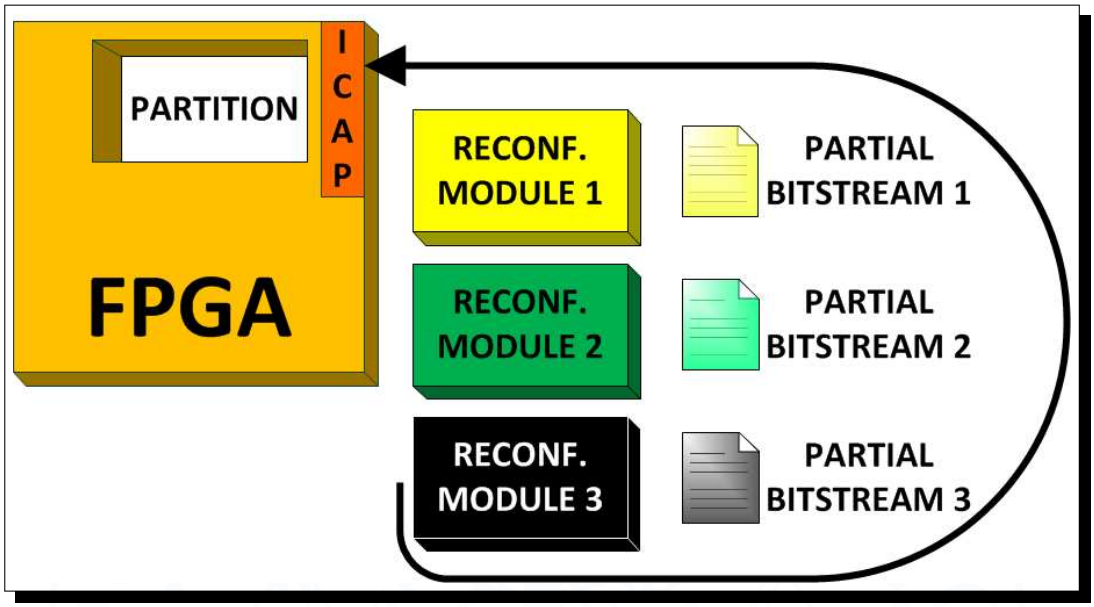


Figure 3.8: Dynamic Partial Reconfiguration - Basic Idea

As shown in Figure 3.8, the function implemented in the Partition is modified by downloading one of several Partial Bitstreams into the Internal Configuration Access Port (ICAP). The logic in the FPGA design is divided into two different types, reconfigurable logic and static logic. The dark yellow area of the FPGA block represents static logic, while the blank portion labelled Partition represents the reconfigurable logic. The static logic remains operational and is completely unaffected by the reconfiguration process. The reconfigurable logic is configured according to the contents of the partial bitstream file that has been downloaded. There are many reasons why the ability to time multiplex hardware dynamically on a single FPGA device is advantageous, like:

- Reducing the size of the FPGA device required to implement a given function by time multiplexing them, with reductions in cost and power consumption;

- Providing flexibility in the choices of algorithms or protocols available to an application, depending on the requirements;
- Enabling new techniques in design security
- Improving FPGA-based system fault tolerance, like fine-grained scrubbing;
- Accelerating configurable computing by implementing on-demand hardware accelerators;

In addition to reducing size, weight, power and cost, Dynamic Partial Reconfiguration enabled and will enable new types of FPGA designs that are impossible to implement without it. Xilinx provides a clear design flow to be followed during the design of a partially reconfigurable project. As shown in Figure 3.9, implementing a partially reconfigurable FPGA design is similar to implementing multiple non-reconfigurable designs, sharing some common logic among them. Partitions are actually used to ensure that the common logic between the multiple designs is identical. The appropriate netlists, described in HDL language, are implemented in each design to generate the full and partial bitstreams for that specific configuration. The static logic from the first implementation is shared among all subsequent design implementations. Each Reconfigurable Module is then synthesized independently from the others in a bottom-up fashion. This means that a separate netlist is written for each Partition, and no optimizations are done across these boundaries, ensuring that each portion of the design is synthesized independently. This can be done through the use of independent projects. For each module, I/O insertion must be disabled, as the ports of these modules (in most cases) do not connect to package pins, but to the static logic above it. The static modules can be synthesized all together to generate one single netlist, or individually to generate multiple static netlists. Afterwards, the static and reconfigurable modules are merged, and the Reconfigurable Partition definitions denote the interfaces between the static and reconfigurable logic.
The Partial Reconfiguration software implements a full design containing static logic and one Reconfigurable Module for each Reconfigurable Partition. Each implementation is done in context. This gives the tools a complete set of information for resource usage, global signals, design constraints, and other requirements. To implement all Reconfigurable Modules, you must choose a subset of all possible Reconfigurable Module combinations and implement them as unique designs. Each unique combination is called a Configuration. Each Reconfigurable Partition can be optionally set as a black box, leaving a "blanking" bitstream as a Reconfigurable Module ("blanking" bitstreams effectively "erase" all reconfigurable logic and routing while the static logic and routes in that region continue to operate). Xilinx's approach to DPR design first requires partition the system, at design-time, into static and reconfigurable modules. Each reconfigurable module must then be binded to a predefined *reconfigurable portion* of the FPGA. Each *reconfigurable portion* can be composed of one or more *frames*. The *frame* is therefore the smallest reconfigurable region [71].

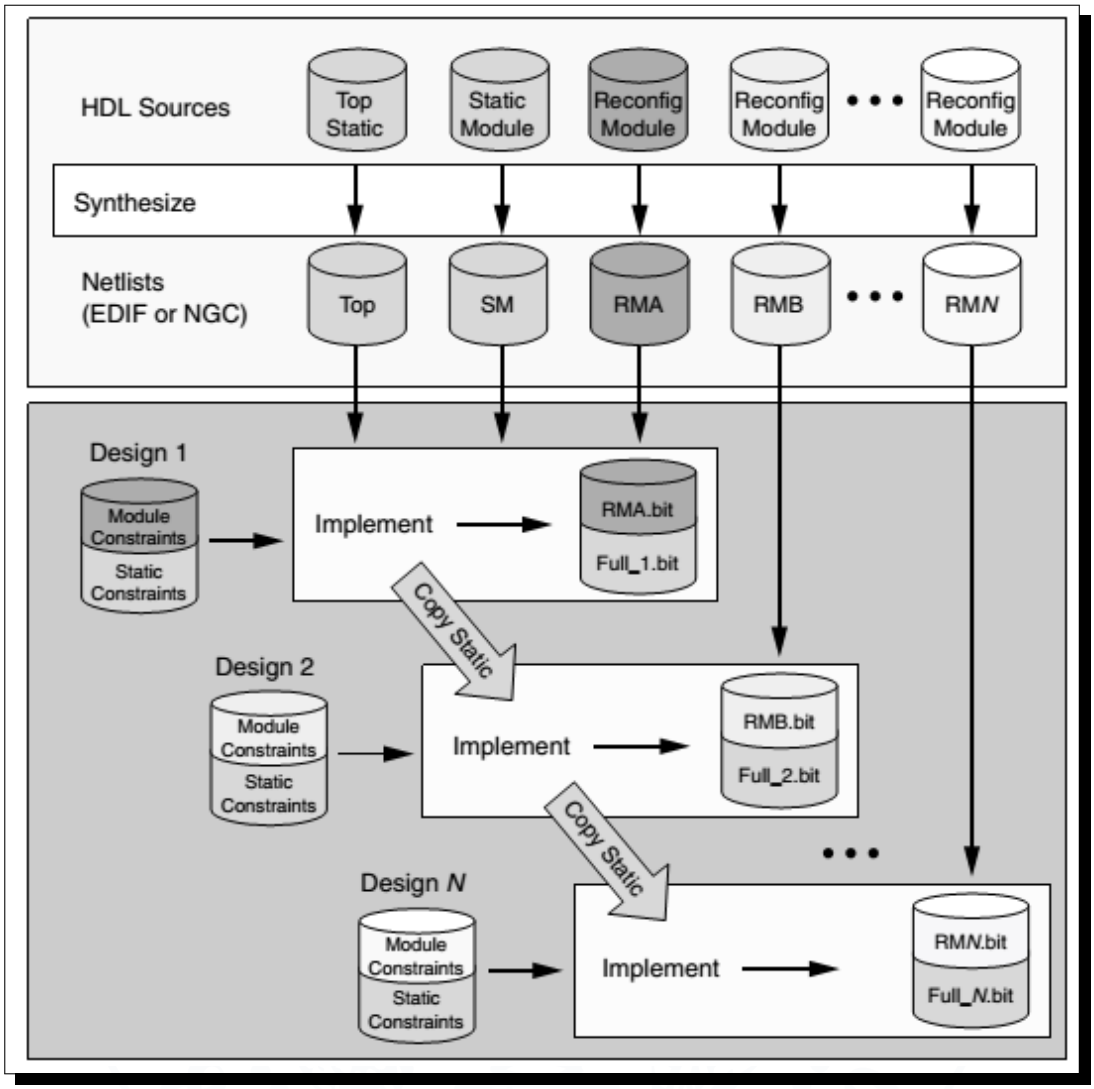


Figure 3.9: Partial Reconfiguration software flow [71]

In particular, on the contrary of older Virtex devices, in Virtex-4 and newer versions the frame has a fixed size which depends by device architecture. For instance, in Virtex-4 each configuration frame spans the height of a single row (i.e., a fraction of 16 Control Logic Blocks (CLBs) [66]). Xilinx's EDA tools generate separate configuration files (called *partial bitstream* or *partial bit file*) for each module to be mapped into a specific *reconfigurable portion*. The partial reconfiguration process can be then activated at run-time by loading a partial bitstream inside the FPGA through dedicated configuration port (see Figure 3.10). Any of the following configuration ports can be used to load the partial bitstream: SelectMAP, Serial, BPI and SPI (Virtex 7 series only), JTAG, or ICAP (Internal Configuration Access Port). ICAP is usually the best choice thanks to its higher bandwidth (3.2 Gbps).

The ICAP protocol is identical to SelectMAP and is described in the Configuration User Guide

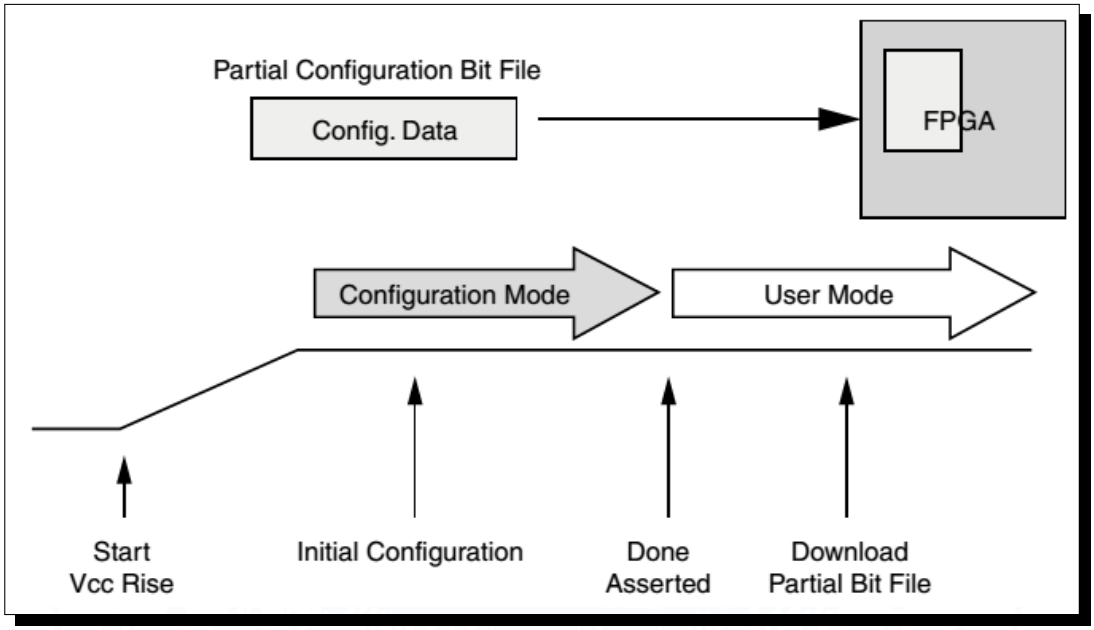


Figure 3.10: Partial Bitstream download [71]

for the FPGA device. The ICAP library primitive can be instantiated in the HDL description of the FPGA design, thus enabling analysis and control of the partial bitstream file before it is sent to the configuration port. The partial BIT file can be downloaded to the FPGA device through general purpose I/O or gigabit transceivers and then routed to the ICAP in the FPGA fabric. The ICAP must be used, with an 8-bit bus only, for Partial Reconfiguration for encrypted 7 series and Virtex-6 partial BIT files. Reconfiguration through external configuration ports is not permitted when encryption is used.

A partial bitstream file can be downloaded to the FPGA device in the same manner as a full con-

figuration. An external microprocessor determines which configuration file should be downloaded, where it exists in an external memory space, and directs the partial bitstream file to a standard FPGA configuration port such as JTAG, SelectMAP or serial interface. The FPGA device processes the partial BIT file correctly without any special instruction that it is receiving a partial bitstream. It is common to assert the INIT or PROG signals on the FPGA configuration interface before downloading a full bitstream file. This must not be done before downloading a partial bitstream file, as that would indicate the delivery of a full file, not a partial one. Any indication to the working design that a partial BIT file will be sent (such as holding enable signals and disabling clocks) must be done in the design, and not by means of dedicated FPGA configuration pins. If the design does not include a soft/hard-core microprocessors such as Microblaze or Pow-

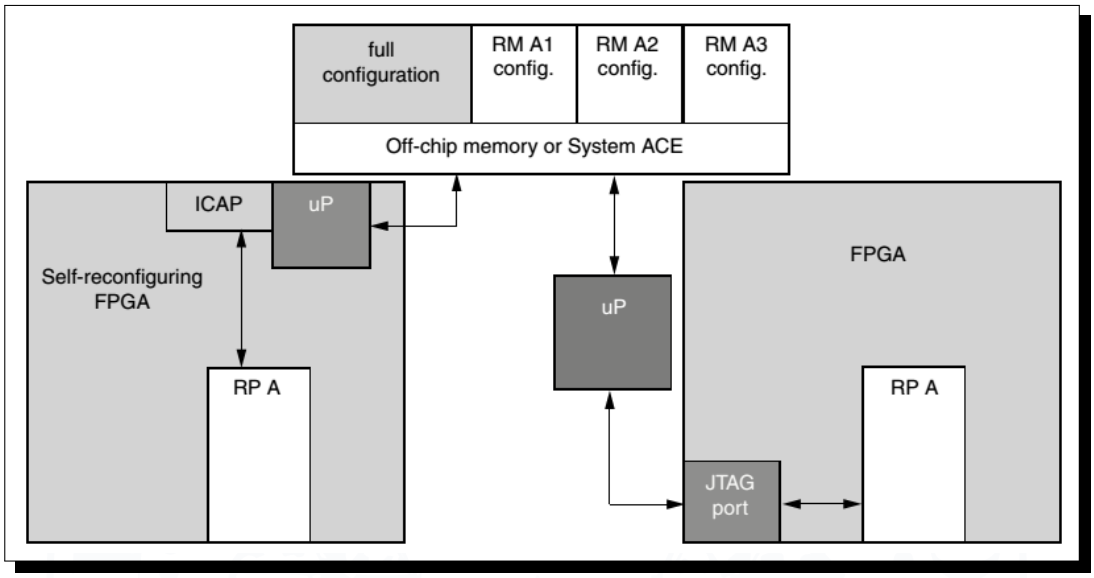


Figure 3.11: Reconfiguration by mean of external microprocessor [71]

erPC [62][68], a user-designed reconfiguration manager is required to load the partial bitstream through the configuration port.
Virtex family devices offers two different ways of implementing partial reconfiguration:

- *partition-based*: is preferred when the reconfiguration involves replacement of entire modules, since the reconfiguration is made by downloading the whole amount of information to be stored in the internal configuration memory;
- *difference-based*: is adopted when the reconfiguration process targets just small changes of some design parameters, such as single *Look Up Tables* (LUTs), since the partial bitstream stores the addresses and the new values of the memory location to be modified, only.

In both cases, the generated partial bit file has no header information and it contains just frame address, configuration data, a final checksum value, and dummy words (i.e., NO-OP).
In the sequel of the section some technical details about the dynamic partial reconfiguration, supported in current Virtex devices (i.e., Virtex-4, Virtex-5, Virtex-6, Virtex-7) are provided.
The configuration control logic consists of a *packet processor* and a set of control registers. The packet processor dispatches data from the configuration interface (e.g., ICAP) to appropriate registers. The reconfiguration process starts when the packet processor receives the *Synchronization word* (Table 3.2). After it, the bitstream words aim at indexing and writing the *frame address register (FAR)*. Configuration data are then flushed inside the FPGA and CLBs are programmed in according to the data logic. After that, the bitstream CRC signature is compared with the value computed by the built-in CRC module. The reconfiguration process ends when the last word, the *Desynchronization command*, is processed. If no problems occur, the reconfigured module is available, otherwise, the *CRC_ERROR* flag in ICAP Status Register is asserted. The reader may refer to [71] for additional information.

Table 3.2: Partial Reconfiguration command sequence

Partial Bitstream word (hex)	**Explanation**
FFFFFFFF	Dummy word
AA995566	Synchronization word
.....	.....
30002001	Write 1 word to FAR reg.
xxxxxxxx	Frame Address
.....	.....
.....	x configuration data words
.....	.....
30000001	Write 1 word to CRC reg.
xxxxxxxx	CRC value
.....	.....
0000000D	Desynchronization command
20000000	NO-OP

To reconfigure a single frame, Xilinx Planahead^{TM} Tool [67] generates a partial bit file composed of 1,067 32-bit words. The total amount of words to address the specific frame is about 2% (i.e., 21 words). The remaining 98% is composed of data logic, CRC signature, and control words. In addition to suitable code for programming CLBs, the data logic comprises a bitmask for programming *switch matrix* blocks. The *switch matrix* is a crucial block allowing CLBs to access routing resources.
Since at design-time routing resources are opportunely allocated by Xilinx Planahead^{TM} Tool to reach the fastest interconnections among static modules, also routing resources physically located into reconfigurable areas could be exploited (i.e., place&route of static modules have the highest priority). It implies that when the dynamic reconfiguration of a module is performed,

switch matrix modules are opportunely programmed to right allocate routing resources. Nowadays, Xilinx Tools do not enable to set constraints for reserving *switch matrix* blocks only for reconfigurable logic. Summarizing, static, system-level, links can route through reconfigurable partition while routes within reconfigurable modules cannot go outside [71].

The world is moving so fast these days that the man who says it can't be done is generally interrupted by someone doing it.

Elbert Hubbard

CHAPTER 4

ENHANCING DEPENDABILITY OF DYNAMICALLY PARTIALLY RECONFIGURABLE SYSTEMS

Modern safety critical systems are based on hardware redundancy. In the case of hardware redundancy the system is provided with more hardware components (e.g. channels) than it would need if the hardware were fault-free. Duplication in the form of self-checking pairs (duplex systems) and Triple Module Redundancy are classical examples of redundancy techniques, with 2 and 3 channels respectively. This led clearly to an area occupation in the selected device that is double or triple with respect to the non-redundant solution.

The enhanced flexibility featured by FPGA-based systems enable the adoption of more sophisticated techniques to achieve the desired dependability level. This chapter will list a set of Design for Dependability (DfD) techniques to be adopted in partially and dynamically reconfigurable systems. The first two methodologies' goal is to decrease the probability of having faults, due to SEEs, during the dynamic reconfiguration process itself. Section 4.1 lists the features of Dependable Dynamic Partial Reconfiguration, that is based on adding signatures in the partial bitstream to check that correct data are written in the configuration memory [17]. Section 4.2 identify a solution to misconfiguration, by storing a compressed partial bitstream internally to the FPGA to rapidly reconfigure a wrongly configured partition [16].

The last two solutions exploit DPR to mitigate aging effects, due to NBTI, in FPGA-based systems. The methodology addressed in Section 4.3 is based on continue reconfiguration to decrease aging effects in configuration memory [19], while the approach of Section 4.4 dynamically change the working frequency to avoid faults due to aging of FPGA internal logic [18].

4.1 Dependable DPR with minimal Area & Time overheads on Xilinx FPGAs

As carefully explained in Section 3.3, configuration information (i.e., partial bitstream) can be stored inside the FPGA or in an external memory. Let us suppose that the *partial bitstream* is stored in an off-chip memory (RAM or Flash), in addition to the hypothesis of fault-free internal logic. If a corrupted bitstream (i.e., soft errors on external links) is loaded into the FPGA, errors can be localized in the address part or the data part of the bitstream. If the address part is corrupted, the static portion of the design could be damaged, thus requiring an overall FPGA reconfiguration. The time overhead caused by the full reconfiguration may be unacceptable in time-critical applications such as hard real-time scenarios. The full reprogramming may also impact high-dependable applications due to the disruptive consequences on the static portion of the design.
On the contrary, if errors are localized in the data part of the bitstream, Xilinx points out that a faulty reconfigured module will be instantiated. To fix this issue, Xilinx suggests to perform again a DPR in the same reconfigurable area [71]. An intensive fault injection campaigns, proved that this is true only under specific conditions. In fact, if links between static modules are routed through the reconfigurable area, a full reconfiguration of the FPGA could be required. For instance, if interconnections between the Reconfiguration Manager (RM) and the Memory Controller (MC) cross the reconfigurable area, a faulty DPR could damage these links, isolating the RM. The RM would therefore be unable to communicate with MC to read the partial bitstream.
In general, for critical applications that do not tolerate full FPGA reconfiguration, Xilinx proposes the approach presented in next section.

4.1.1 Xilinx approach

The problem of dependable partial reconfiguration has been addressed by Xilinx in the *Partial Reconfiguration User Guide* [71], where a Partial Bitstream CRC Checking is suggested.
The main idea is to split the original partial bit file into *blocks*, and for each block to calculate a single word *CRC* signature. Finally, the partial bit file is reassembled, combining the blocks and their corresponding *CRC* signatures, as shown in Figure 4.1).
At reconfiguration time, whenever a new block is loaded from the external memory, the *CRC* signature is recalculated by a dedicated hardware component, and the block is stored into a *BRAM* inside the FPGA.
When the whole block has been transferred from the external memory, the reconfiguration manager compares the received *CRC* signature with the calculated one (Figure 4.2).
If the *CRC* comparison is successful, the block can be sent to the ICAP and the related portion of the frame reconfigured. Otherwise, the reconfiguration manager must reload the block from the external memory.
This implementation requires a *CRC* evaluator, a Finite State Machine (FSM) for the control of

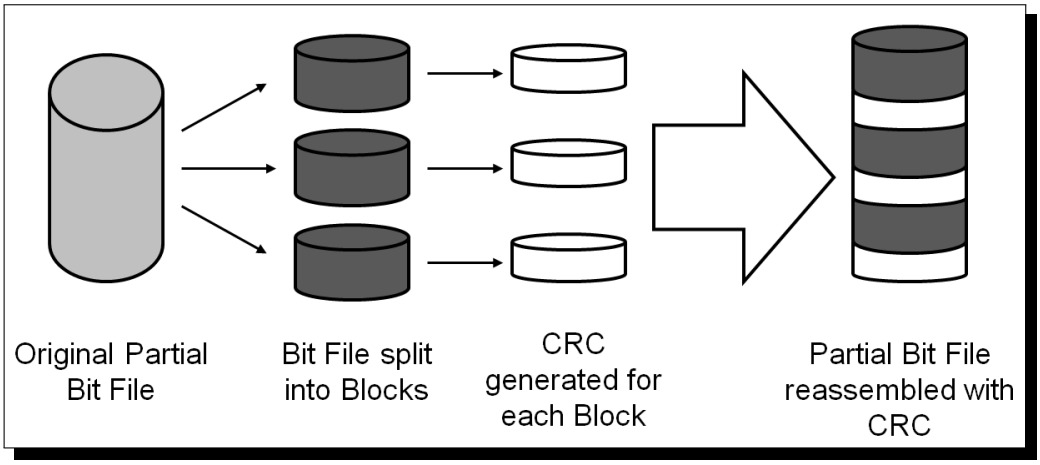


Figure 4.1: Xilinx solution - Bitstream generation

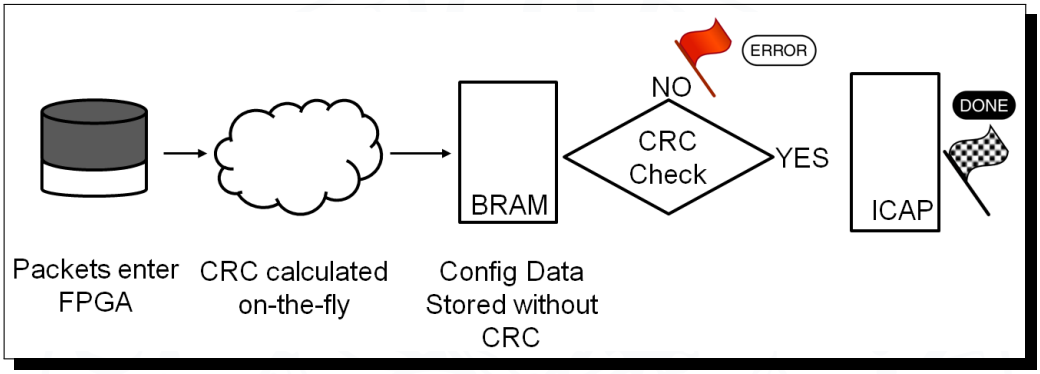


Figure 4.2: Xilinx Solution - Loading process

the reconfiguration process (i.e., a Reconfiguration Manager) and a set of BRAM(s) to buffer the data.

Depending on the available FPGA resources, the designer need to choose the best block size, that impacts on both reconfiguration time and internal memory occupation.

On the one hand, when increasing the number N of blocks (i.e., when decreasing the number of words per block), a higher number of *CRC* signatures must be stored, leading to an increase of bitstream size. At the same time, the memory occupation becomes smaller, since the required buffering capability equals the block dimension.

On the other hand, when decreasing the number of blocks, fewer *CRC* signatures must be stored in the bitstream, thus decreasing the total amount of words in the external memory. However, during the *CRC* checking process, more words must be stored in the BRAM inside the FPGA, thus increasing the area occupation.

The total reconfiguration time is due to two contributions:

$$T_{X-CRC} = T_{read} + T_{block} \tag{4.1}$$

where:

- T_{read} is the time required to load the bit file from the external memory:

$$T_{read} = \frac{K+N}{\min(f_{ICAP}, f_{Mem})} \tag{4.2}$$

  where K is the bitstream dimension in terms of 32 bit words; N is the number of *CRC* signatures in terms of 32 bit words; f_{ICAP} is the working frequency of the ICAP; f_{Mem} is the memory working frequency.

- T_{block} is the time spent loading the buffered block from BRAM to the ICAP:

$$T_{block} = \frac{K/N}{f_{ICAP}} \tag{4.3}$$

  where K/N is the block dimension in terms of 32 bit words.

Using this model, it is possible to evaluate how the reconfiguration time in Xilinx solution is influenced by the block dimension. Figure 4.3 plots the reconfiguration time as a function of the block dimension K/N. This evaluation has been performed with different bitstream file dimensions (# frames involved in the reconfiguration).
Figure 4.3 shows the reconfiguration time considering a 100 MHz working frequency for both ICAP and external memory. Note that, with a block dimension of a single word (N=K), one *CRC* signature for every word is required. In this case, T_{read} becomes the most relevant term, but no BRAM is required for buffering.
The same time overhead occurs when a block is as long as the bitstream (N=1). While there is just one *CRC* signature, the whole bistream must be buffered before being sent to the ICAP, resulting in high BRAM occupation. This significantly increases T_{block}.
From Figure 4.3, designers can identify the optimal block size. As an example, when dealing with a 1 frame reconfigurable area, the bitstream is 1,067 words long, and the optimal block size is 32 words.

4.1.2 Proposed Methodology

Despite the solution proposed by Xilinx is fairly comprehensive, it may implies significant time and area overheads. In the sequel we propose a methodology that reduces these overheads, by protecting just the critical part of the partial bitstream. Some ad-hoc Design for Dependability (*DfD*) rules are also introduced.

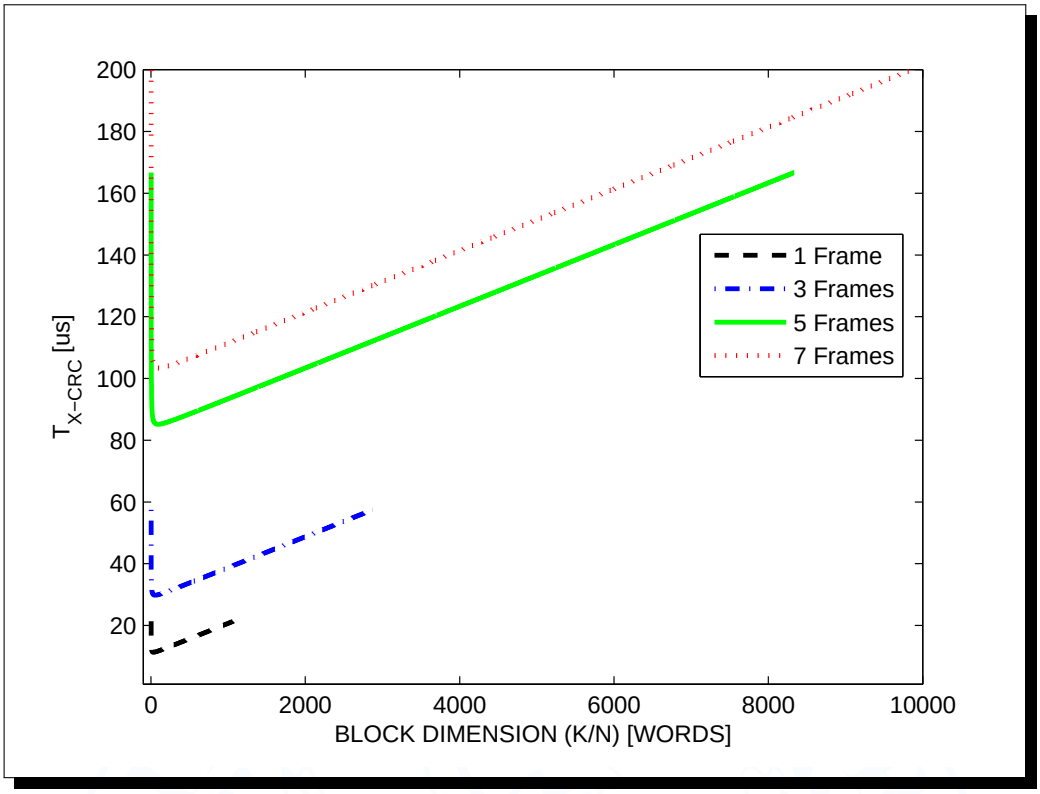


Figure 4.3: Reconfiguration time of Xilinx solution

4.1.2.1 Partial bitstream file splitting

As aforementioned, the partial bit file is composed of three main parts. The first part contains the address and control information. The second includes the data for the reconfiguration in the selected frame(s). The last part is the built-in ICAP CRC checksum.
It is straightforward that, in terms of dependability, the most critical part is the first one, since it defines the portion of the FPGA to reconfigure. In fact, if an error occurs in an address or control information, a static portion of the FPGA could be unintentionally reconfigured and the system could become inoperative.

The proposed approach deeply protects this portion of the partial bitstream that contains the most critical words, letting the rest unprotected. In the original *partial bit file*, the identified critical words are globally 21, for both control and address. This means that, in our approach, only 21 *CRC* signatures are generated. Thus, the final bitstream file is generated, adding the *CRC* signatures for critical words, as shown in Figure 4.4.
At reconfiguration time, the critical words are checked, while the non-critical words are loaded

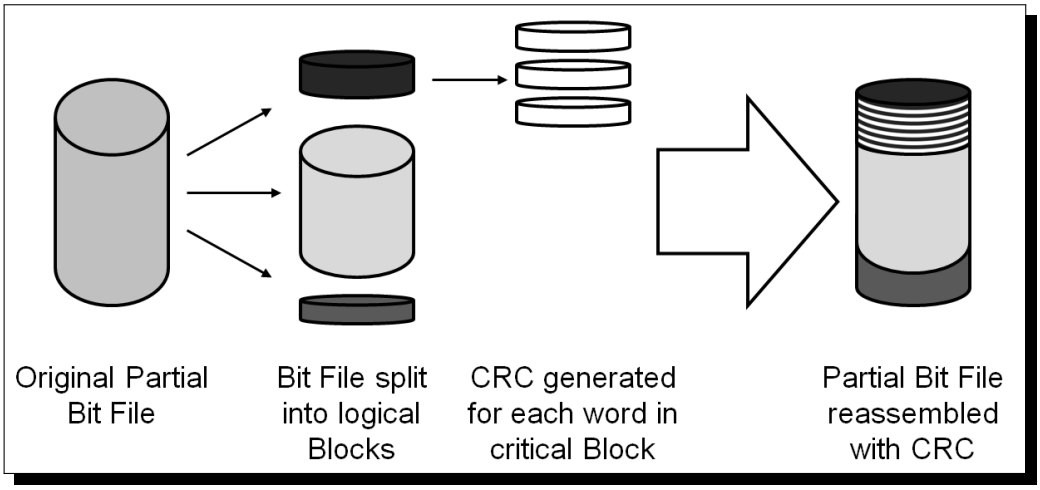


Figure 4.4: Our software solution

from the external memory and directly sent to the ICAP, without any time overhead or buffering, as shown in Figure 4.5.

The proposed solution requires a *CRC* evaluator and a Reconfiguration Manager to control the

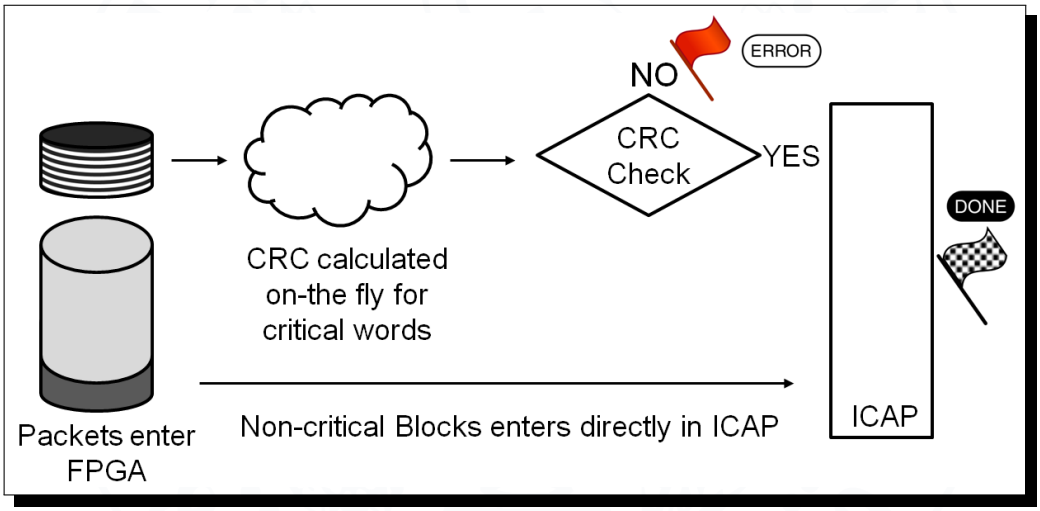


Figure 4.5: Our Hardware Solution

reconfiguration process, while no BRAMs are required.

The reconfiguration time with the proposed *CRC* checking is

$$T_{OUR-CRC} = \frac{K+C}{\min(f_{ICAP}, f_{Mem})} \tag{4.4}$$

where K is the bitstream dimension in terms of 32 bit words; C is the number of critical words; f_{ICAP} is the working frequency of the ICAP; f_{Mem} is the memory working frequency.
Figure 4.6 plots the ratio $T_{X-CRC}/T_{OUR-CRC}$, which proved to be always >1. Therefore, the proposed solution is always faster then the Xilinx one.

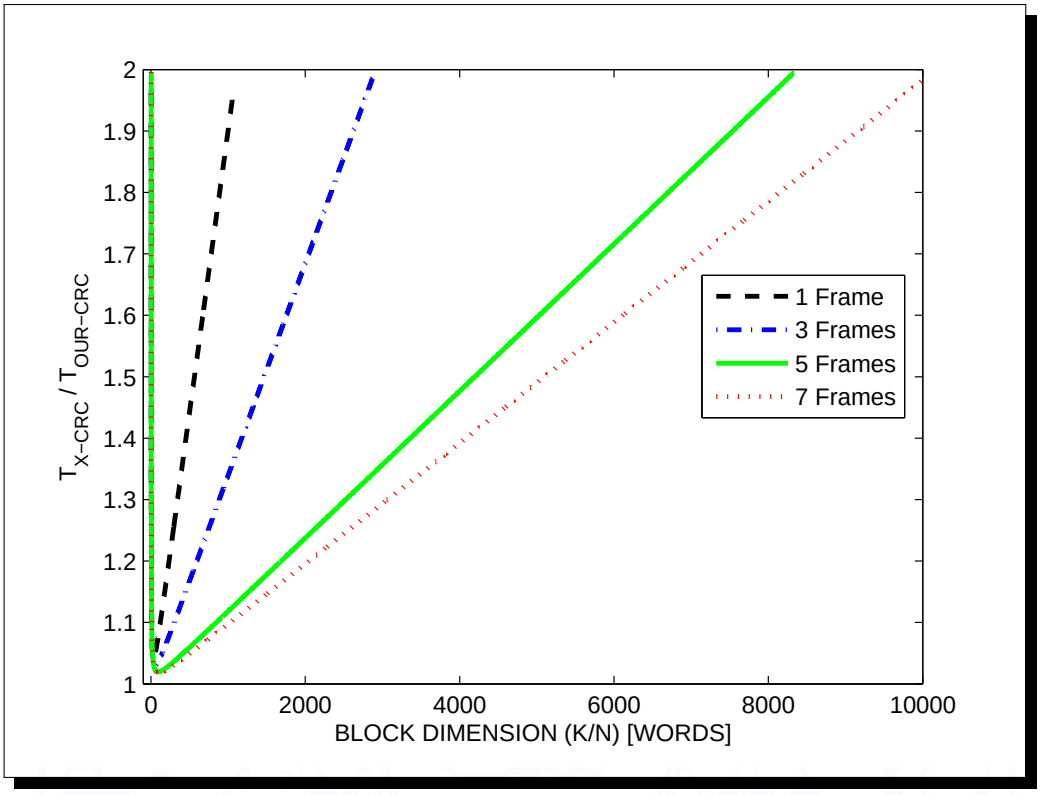


Figure 4.6: Comparison between proposed solution and Xilinx solution

Despite the proved time and area occupation advantages of the proposed solution, to assure a high dependability of the reconfiguration process, we have to guarantee that an error in the non-checked part of the bitstream file will not lead to a fault in the system. The jeopardy is that some static connections are routed in the reconfigurable area, and, due to a faulty reconfiguration process, the link between two points could be broken. This goal is achieved by fulfilling the following *Design for Dependability* rules:

1. potential critical links must not cross any reconfigurable area

2. connection inside critical modules must not cross (i.e., be routed through) reconfigurable areas.

4.1.2.2 DfD#1: Critical links protection

To ensure a dependable reconfiguration process, critical connections must be protected. These include:

- External-Memory to Memory Controller links;
- Memory Controller to Reconfiguration Manager links;
- Reconfiguration Manager to ICAP links.

In order to guarantee that these links do not cross the reconfigurable area, after the automatic routing performed by the synthesis tool, the layout must be checked and some links manually re-routed, if required.
The rationale of this *DfD* rule is that Xilinx Design tools do not allow to specify this kind of constraints at design time.

4.1.2.3 DfD#2: Critical modules protection

The second DfD rule imposes that all critical modules must be protected.
In systems which use partial reconfiguration, since bitstreams are loaded from the external memory, all modules involved in the communication between the external memory and the ICAP must be considered critical (see Figure 4.7). In addition, also design specific modules could be considered critical. Their integrity can be preserved by constraining critical modules in predefined physical region called *partitions*. Xilinx PlanAhead tool enables the user to manually place a module in a specific area, guaranteeing that all the specified hardware and the related connections are inside the physical regions.

4.1.3 Experimental results

This section reports a set of experiments performed to validate the proposed methodology and to compare its performance with the Xilinx solution.
The experimental setup includes a *Leon3* [24] based SoC, implemented on a Xilinx ML403 demo board, equipped with a Xilinx Virtex 4 FPGA device and 64 MB of DDR SDRAM [63]. The SoC contains a reconfigurable area in which it is possible to dynamically load two modules, namely the *APBUART* and the *GPTIMER* from *Gaisler Research IP Library* [2]. Both modules require a reconfigurable area composed of 2 frames. The SoC also includes an ad-hoc reconfiguration manager connected to the AMBA bus and an internal timer able to measure the partial reconfiguration time. The reconfiguration manager is able to:

- address the external memory, loading bitstream files without any CPU intervention (*Direct-Memory-Access*);

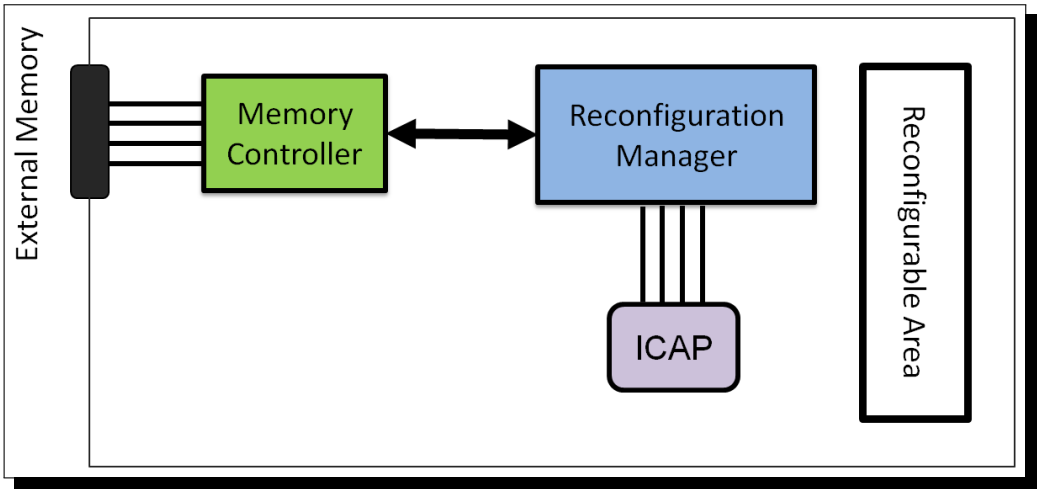


Figure 4.7: Critical connections and cores

- perform DPR through ICAP;
- perform the *CRC* check;
- automatically manage the whole reconfiguration process, even in presence of errors.

The synthesis of the overall design has been performed using Xilinx ISE Design Suite 14.4. The *Leon3* processor works at 66 MHz, the ICAP controller and, the DDR SDRAM at 100 MHz.

4.1.3.1 Xilinx approach implementation

The Xilinx protection methodology has been implemented using a parallel CRC-32 to minimize the CRC latency. The selected polynomial is *0x90022004* which guarantees Hamming distance equal to 6 [38].
For each considered reconfigurable module a partial bit file of 1,969 32-bit words has been generated.
We evaluated the reconfiguration time and the required area considering the following block sizes: 4, 16, 32, 44, 64, 128, 256, 512 32-bit words. Figure 4.8 shows the relation between the reconfiguration time and the block size. The solid line plots the reconfiguration time calculated using Equation 4.1 while the dots report the measured reconfiguration time for the 8 considered block sizes. The graph confirms that the considered mathematical model provides a good estimate of the configuration time and can be used to identify the best block size for a given design. According to Figure 4.8 the optimum configuration for the Xilinx solution is a block of 64 32-bit words with a reconfiguration time of 63.72 μs. With this architecture, the synthesized reconfiguration manager requires 290 slices and 1 BRAM.

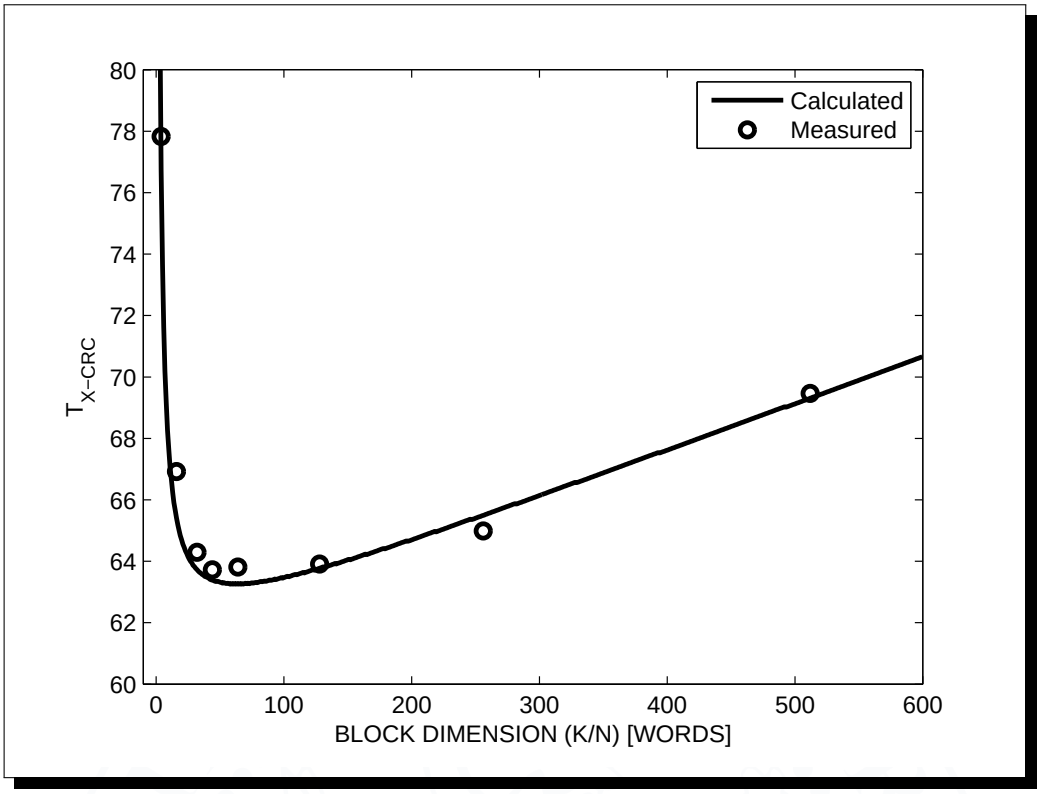


Figure 4.8: Reconfiguration time with 2 Frames

4.1.3.2 Proposed approach implementation

Differently from the Xilinx solution, the proposed approach is designed to protect 16-bit critical words. A smaller CRC can therefore be adopted. We implemented a parallel CRC-16 with polynomial equal to *0x968B* which guarantees Hamming distance equal to 7 [39].
The two presented DfD rules have been applied to the proposed design. Partitions have been created using Xilinx PlanAhead to protect the SoC critical modules (i.e., Reconfiguration Manager (RM) and Memory Controller (MC)). To ensure that the processor keeps running also after a faulty reconfiguration, the *Leon3* has been constrained in a specific region, also (see Figure 4.9). This introduces a minimal degradation (1.2%) in the maximum working frequency.
Finally, all critical connections have been checked, in order to assure that they do not cross the reconfigurable area, and only 2 links were manually re-routed using the Xilinx *FPGA Editor* Tool (see Section 4.1.2.2).
The synthesized reconfiguration manager requires 295 slices and no BRAMs. A fault free reconfiguration requires 61.78 μs.

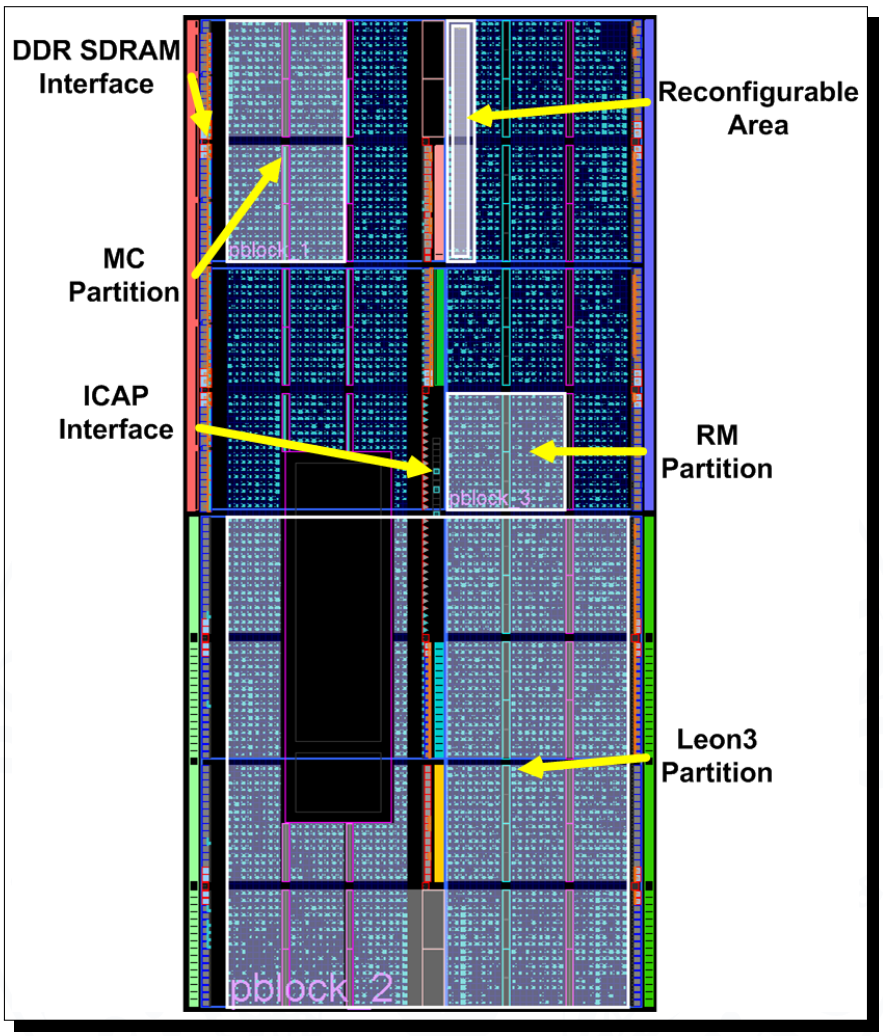


Figure 4.9: PlanAhead Device View

Table 4.1: Area occupation and reconfiguration time of different implementations

Solution	**Block Size [bit]**	**CRC [#]**	**BRAM [#]**	**RM [# slices]**	**Reconfig. Time [us]**
w/o CRC	0	0	0	197 (3.60%)	61.04
Proposed Approach	1x16	42	0	295 (5.39%)	61.78
Xilinx	64 x 32	31	1	290 (5.30%)	63.72
Xilinx	1 x 32	1,969	0	290 (5.30%)	77.88
Xilinx	1,969 x 32	1	8	290 (5.30%)	73.49

4.1.3.3 Comparison

Table 4.1 compares the two analysed solutions, in terms of area occupation and fault free reconfiguration time. The assets of the proposed solution are no BRAM occupation and a shorter reconfiguration time compared to the Xilinx solution, with a very small are overhead in the configuration manager (increase of 1.7% of slices) due to a more complex Finite State Machine. The reported reconfiguration times, computed with the model presented in Section 4.1.2, are related to:

- 2 frames reconfigurable area (1,969 words)
- 1506 Mbit/s throughput of DDR SDRAM.

For sake of completeness, Table 4.1 also provides information about the worst cases of the Xilinx solution, and a CRC free DPR system.
So far the performance comparison has considered the fault free DPR time, only. The rest of this section will compare DPR performance in case of faults in the bitstream, which have been injected in the words of the data portion, only. This condition is conservative, since it represents the worst-case condition for the proposed solution. In fact, when applying Xilinx' solution, if a faulty block is loaded, the error is detected as soon as the CRC of the block is checked. The block is immediately reloaded, thus introducing a time overhead equal to the block loading time. In the proposed solution, if an error occurs in a critical word, it is immediately detected enabling the system to reload the corrupted word. If the error instead occurs in the data portion, it will be detected only at the end of the reconfiguration process, during the ICAP CRC check. In this case the full DPR process must be restarted since the reconfigurable area has been corrupted, thus introducing a higher overall reconfiguration time overhead.
The time overhead introduced by errors in the loaded blocks for the two considered solutions is therefore influenced by the DPR rate, and by the word error probability observed when loading bitstream blocks. Figure 4.10 , Figure 4.10 and Figure 4.12 analyse the difference in system activity time spent for DPR in the two solutions over a one day observation period for different DPR rates and word error probabilities, thus enabling an easy comparison of the two solutions.

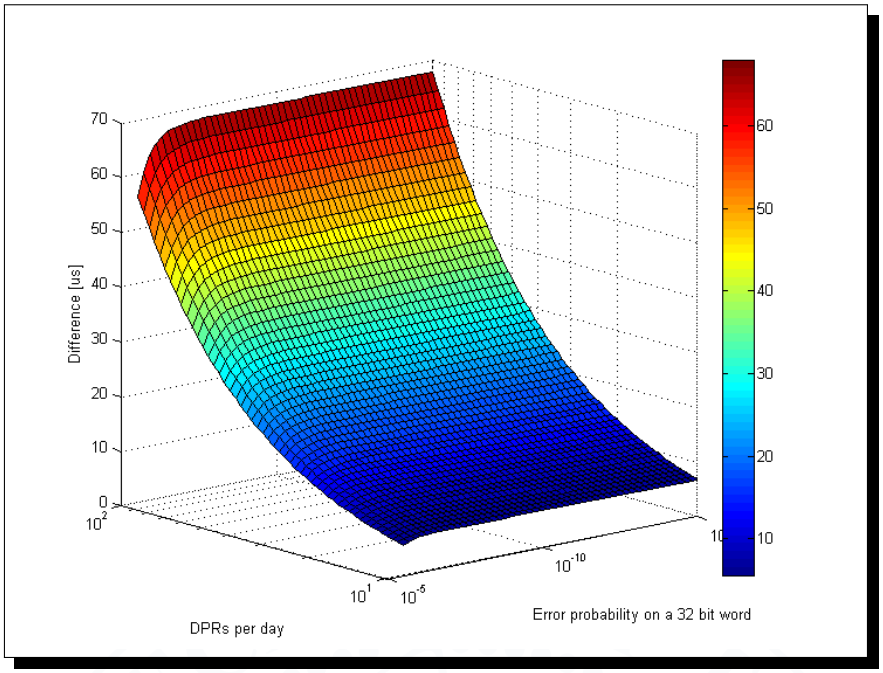


Figure 4.10: Difference of DPRs time in 1 day - 2 Frames

Figure 4.10 analyses the case of a 2 frames reconfigurable area, i.e., a partial bit file of 1,969 32-bit words. The experiments show that the proposed technology outperforms the Xilinx solution in all working conditions enabling a significant improvement in the overall reconfiguration time.
Figure 4.11 performs a similar analysis, but considering a larger reconfigurable area composed of 4 frames. Also in this case, the proposed approach should be preferred.
Figure 4.12 performs a similar analysis, but considering a larger reconfigurable area composed of 8 frames, i.e., a partial bit file of 11,040 32-bit words. In this case, when the word error probability increases over 10^{-6}, the Xilinx solution should be preferred. This is due to the additional reconfiguration process required in our solution whenever data words are corrupted. Nevertheless, decreasing the error probability or the number of reconfigurations per day, the proposed methodology is the best one.

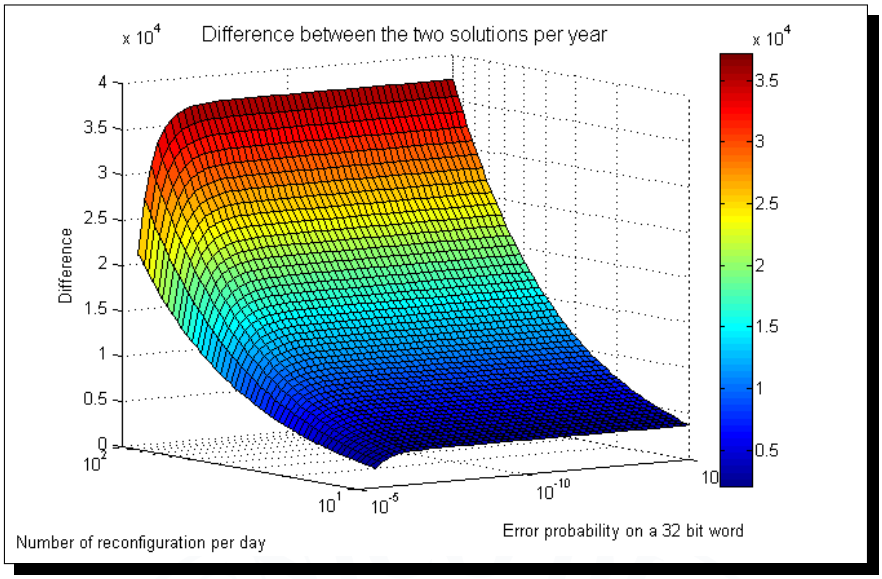


Figure 4.11: Difference of DPRs time in 1 day - 4 Frames

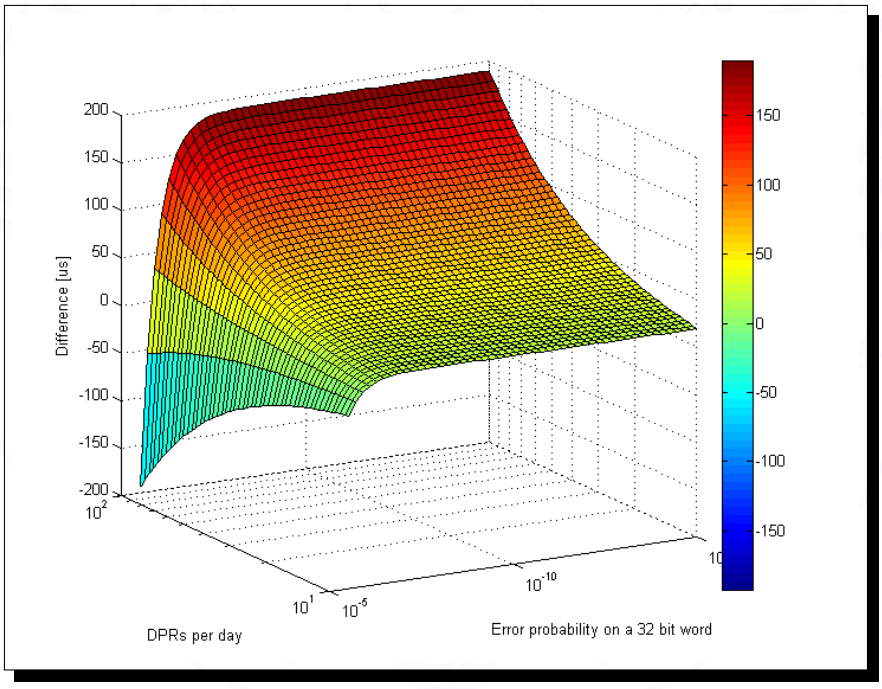


Figure 4.12: Difference of DPRs time in 1 day - 8 Frames

4.2 ZipStream: improving dependability in Dynamic Partial Reconfiguration

The solution proposed in Section 4.1 guarantees, also for faulty reconfigurations, that the rest of the system (i.e., static portion) continues to function. Nevertheless, it introduces an amount of latency due to the adopted buffering mechanisms that, in some cases, is not acceptable.
In addition, the recovery mechanism is not described by Xilinx, and the choice is left to the user. A widely used approach consists in reading again the bitstream and trying to reconfigure again with the same bit file. This process must be handled by a reconfiguration manager and leads to a very long reconfiguration time. Moreover, it does not assure that the reconfiguration process will finally ends properly.
In fact, if the bitstream read from the external memory is definitely corrupted, the reconfiguration process will always be stopped.
A widely used more dependable solution requires to store the bitstream inside the FPGA. Since the BRAMs of the device are protected by Error Correction Codes (ECCs), the bitstream will be safely stored. Nevertheless, this will come at a cost in terms of memory occupation, that in modern SoPCs is a really critical resource. An effective solution is to compress the data to be stored, in order to reduce the memory occupation.
The next section presents an overview of compression algorithms, taking into account the complexity required for the hardware implementation.

4.2.1 Compression Algorithms overview

A compression algorithm aims at decreasing the size of an original array of data, involving encoding information. The achieved compression can be either lossy or lossless.
Lossy algorithms reduce the dimension of the array by eliminating marginal information. Usually, a lossy compression achieves a high compression rate, but the original data can not be reconstructed. Examples of lossy compressions are MPEG2, JPEG and MP3 [46].
Lossless compression algorithms allow the original array to be reconstructed from the compressed data. Lossless data compression is usually based on the statistical model of the input data, exploiting redundancy to compress the original data. Therefore, the achieved compression rate is closely related to the statistics of the data to be compressed.
Two main lossless encoding algorithms are used: *Huffman Coding* and *Arithmetic Coding*. In both cases, the most frequently used characters are coded by fewer bits, while less frequently occurring characters are coded by more bits. This led to fewer bits used in total.
Huffman coding [10] is the most well-known and widely used variable-length code. It separates the input data into symbols (choosing their lengths) and replaces each symbol by a code.
Arithmetic coding, instead, encodes the entire message into a single number. Arithmetic coding [41] achieves high compression rates for particular statistical models, whereas it implies higher computational complexity. Huffman compression is simpler and faster, but produces poor re-

sults for models that deal with uniform symbol probabilities. It provides very high compression ratios when input data are very redundant.
Another kind of compression algorithm, based on the characteristics of the data to be compressed, is the *Run-length encoding* (RLE) [10]. In RLE, sequences of the same value are stored as a single data and the number of occurrences in the sequence.

4.2.2 ZipStream Methodology

ZipStream is a novel methodology to increase the level of dependability in reconfigurable FPGA systems by internally storing as backup a special compressed bitstreams. The solution is based on storing in the FPGA a compressed partial bitstream for each reconfigurable partition. When a corrupted bitstream (i.e., checked by the ICAP built-in CRC) is read from the external memory, at the end of the reconfiguration the static part of the system may be damaged). In order to restore in a very short time the static system functionalities, an internally stored *black-box* [67] bitstream configures the corrupted reconfigurable partition.
Since the main goal is to assure the proper functioning of the system, the compressed bitstream will reconfigure the target partition with a *black-box* module, able to ensure the static connections passing through the reconfigurable partition to be correctly restored.
The *black-box* partial bitstreams, generated by the Xilinx PlanAhead tool, encompass just the information of the static part of the reconfigurable partition, since there is neither logic nor nets associated with this partition. The main peculiarity of this special bitstream is that it embeds a lot of data redundancy exploitable for an efficient compression.
In general, a partial bitstream contains information items related to the logic and routing resources present in the associated reconfigurable partition. A *black-box* partial bitstream carries just the information about the static connections of the design which routes through the targeted reconfigurable partition. Since it does not contain any information associated with the logic resources, it includes long sequences of zeros. Since the ICAP CRC check does not occur until the end of a reconfiguration process, a faulty bitstream loaded from the external memory may damage the static portion of the design, including connections associated to the reconfiguration controller. In this case, the system will not be able to restore the faulty partition by a *black-box* module. Thus, to make the ZipStream methodology effective, a *Design-for-Dependability* (DfD) rule must be adopted at design time.
In addition, the features of the *black-box* partial bitstreams enable the use of an ad-hoc designed compression algorithm based on the Huffman encoding (see Section 4.2.2.1), whose decompressor requires very low hardware resources.
The ZipStream approach consists of three different steps:

- apply the optimal *Compression algorithm* to the partial bitstreams in software;
- design the *Hardware Decompressor* to be implemented in hardware;

- apply the *DfD* design rule to the system design.

4.2.2.1 Compression Algorithm

The partial bitstream file is a stream of data to be downloaded in the ICAP, composed of hundreds of 32-bit words. To compress these data, a combination of two different algorithms has been used to achieve a high compression rate.
Since the bitstreams to be compressed are characterized by many sequences of zeros, the first applied algorithm was the *Run-Length Encoding* (RLE).
The RLE encodes a sequence of consecutive zeros by a binary number. Obviously, the insertion of 1-bit flag to distinguish normal symbol or encoded symbol is required. This encoding is based on splitting the stream of bits in symbols. All symbols have the same length in terms of number of bits. The length clearly has a great impact on the compression ratio, since the symbol probability changes with the symbols' length. In order to enhance the flexibility of the methodology, the software compressor is able to automatically calculate the symbol length to achieve the best compression rate. To do this, the RLE algorithm is applied with three different symbol lengths (e.g., 8, 16 and 32 bits). The set of lengths has been chosen to better fit the data in the hardware, since the decompressor should save these data in a BRAM, 8-bit words addressable.
After applying the RLE algorithm, the software generates the compressed bitstreams, using the best symbol length configuration.
Afterwards, the Huffman encoding is applied on the RLE encoded bitstreams. Huffman coding has been chosen by evaluating the hardware resources needed in the decoder for both Huffman and Arithmetic coding. In fact, the complexity of the Huffman decoder is definitely lower than the Arithmetic one.
The Huffman encoding process consists of two steps: *Huffman tree construction* and *Single side Growing Huffman encoding*.
The *Huffman tree construction* implements the following algorithm: occurrences of all words are evaluated, then the array of words is sorted in descending order of frequency, in order to correctly generate the Huffman tree. This frequency-sorted binary tree is then used to assign the symbols to the words in the standard Huffman encoding. The more frequent is the word, the shorter will be the assigned code.
Nonetheless, since these data must be stored inside the FPGA BRAMs, the trade-off between performances and storage occupation need to be considered carefully. Considering the hardware implementation, the data structures should not be too complex to reduce the logic area occupation.
In this context, the Single-side Growing Huffman (SGH) tree [27] proved to be the most efficient in terms of memory occupation [13]. The reader may refer to [13] for more information.
The previously constructed Huffman tree is translated into a SGH tree. In the proposed method-

ology the SGH tree coding process is fully computed by software, providing as output the compressed bitstream. The translation table must be stored inside the FPGA internal memory. The memory occupation has been reduced thanks to a smart approach based on the special features of the SGH table. In fact, it is easy to group the code words depending on the number of the prefix ones. This introduces the possibility to decode the code words in a hierarchical way (see Section 4.2.2.2 and Section 4.2.3).
The hierarchical decoding is done in two different steps. Code word groups can be simply identified by using hardwired logic, while symbols decoding is done exploiting a *Look-Up-Table* approach. As example, when symbols length is 8 bits, and the code length is 12 bits, the original memory size needed to store the LUT is 4 KB. In our solution, the LUT size is only 128 B.
After the coding algorithm outputs the code word for each symbol, the *Look-Up-Table* (LUT) ROM is generated to store the data in the FPGA.

4.2.2.2 Hardware Decompressor

The decompressor (or decoder) is hardware implemented in the device logic. The decompressor performs the Huffman decoding on the input code words, and then the RLE, if necessary, providing in output the original bitstream data. Figure 4.13 shows the architecture of the hardware decoder.
Initially, the 16-bit value of the *REG C* register, that represents the number of already decoded bits of the input packet, is 0. The 16-bit packet received in input is stored in *REG B* register. The packet is passed directly to the *LUT ADDR DECODER*, without performing the shift operation. The *LUT ADDR DECODER* decodes the group which the code word belongs to, by counting the number of leading 1 in the code word. After identifying the group, the *LUT ADDR DECODER* generates the address for the *LUT ROM*, which contains the SGH table.
The *LUT ROM* outputs the symbol associated to the input code word and the relative code length. The symbol is then passed to the *RUN-LEN DECODER*, while the code length is accumulated by the 16-bit *ACCUMULATOR* and the result is stored in the *REG C* register. The value of this register controls the *BARREL SHIFTER* shift operations.
The *BARREL SHIFTER* module shifts the input packet of the number of bits indicated by *REG C*, flushing the already decoded bits. In this way, only the undecoded bits appear at the input of the *LUT ADDR DECODER*.
In case there is an overflow in the *ACCUMULATOR*, it means that a codeword is split between two consecutive packets, so the next packet is received in *REG B* register, while the previous packet is moved to the *REG A* register.
REG A and *REG B* values are concatenated and input to the *BARREL SHIFTER*. The described operations are repeated recursively until no more input packets are received.
The output symbol from the *LUT ROM* is finally decoded by the *RUN-LEN DECODER*, if neces-

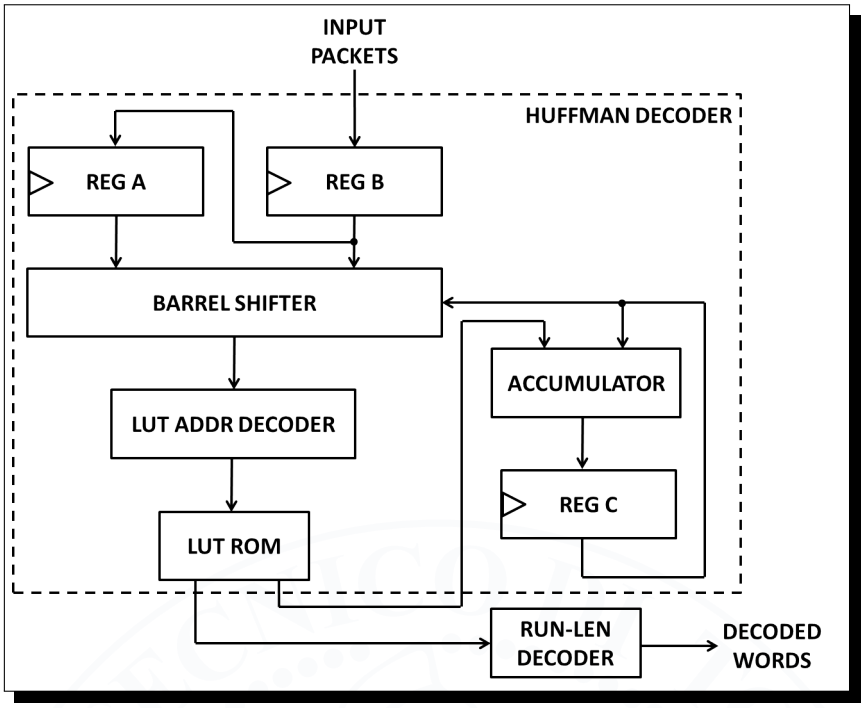


Figure 4.13: Decompressor architecture

sary. This decoder checks the flag bit in symbols. Whenever the flag is asserted, it implies that this symbol has to be decoded by the *RUN-LEN DECODER*. The *RUN-LEN DECODER* read the binary value of the symbol and output a sequence of consecutive zeros with a length equal to that value. The output of the *RUN-LEN DECODER* represents the decoded words of the bitstream, that can be passed to the ICAP interface exploiting the reconfiguration controller.
The operations of the hardware decompressor are managed by the reconfiguration controller. The controller is directly connected to the ICAP interface to download the bitstream with correct timing.
At run time, when a reconfiguration request is received, the controller read an external stored bitstream in order to reconfigure a partition. If an error at the end of the reconfiguration process is detected by the ICAP, the controller starts reading the compressed *black-box* bitstream from the internal BRAM of the FPGA. Then, it manages the decoding process and it provides the decoded words to the ICAP interface.

4.2.2.3 Design-for-Dependability rule

As mentioned above, a corrupted partial bitstream, read from the external memory, can led to a system damage. To avoid the corruption of the connections of the modules used to restore the

correct behaviour of the static part of the system, a simple *DfD* rule must be adopted.
The rule aims at enclosing all the aforementioned connections in such a way that they cannot be routed trough a reconfigurable partition. This can be achieved by including in a non-reconfigurable partition all the critical blocks:

- Reconfiguration Controller (RC)
- ICAP
- BRAMs, that store the *black-box* bitstreams
- Decompressor

This rule ensures that all connections between these blocks, that are necessary after a faulty reconfiguration, do not pass through the faulty reconfigured partition. Thus, the correct functioning of these modules is guaranteed also after a faulty reconfiguration.
Figure 4.14 shows an example of how these blocks can be protected. The highlighted block on the top of the device represents the reconfigurable partition. The non-reconfigurable partition, in the middle of the device, includes the Reconfiguration Controller, the *Hardware Decompressor*, and the BRAM that stores the *black-box* bitstream associated to the reconfigurable partition. The ICAP cannot be included in a partition, thus it is advisable to verify that the few connections between this interface and the RC do not pass through the reconfigurable partition.

4.2.3 Experimental results

The ZipStream methodology is composed by the Huffman software encoder and the hardware decoder. First of all, many partial bitstreams were generated using Xilinx PlanAhead Tool v14.2, targeting Virtex 4 XC4VFX12-FF668-10C FPGA. The design includes the Leon3 soft-core [24] and many different reconfigurable partitions.
Six different reconfigurable partition sizes (i.e., from 1 to 6 frames) have been tested. The black-box module has been placed in 10 different positions in the device, in order to have 10 different partial bitstreams for each partition.
To correctly encode the bitstream file and to find the solution that guarantees the best compression ratio, several compression algorithms have been tried. First, *black-box* partial bitstreams are encoded with the standard Huffman algorithm (Huffman 32, 16 and 8 in Figure 4.15 and Figure 4.16), splitting in different word length configurations: 32 bits, 16 bits, 8 bits. Then, a combination of Huffman and Run-length encoding is used to compress the bitstreams (Huffman+RLE 32, 16 and 8 in Figure 4.15 and Figure 4.16).
Figure 4.15 and Figure 4.16 shows the compression rate in different scenarios. The graph plots average compression ratio, between the 10 partial bitstreams considering 6 different partition dimensions. The *combination* column, in Figure 4.16, shows the average compression rate in the

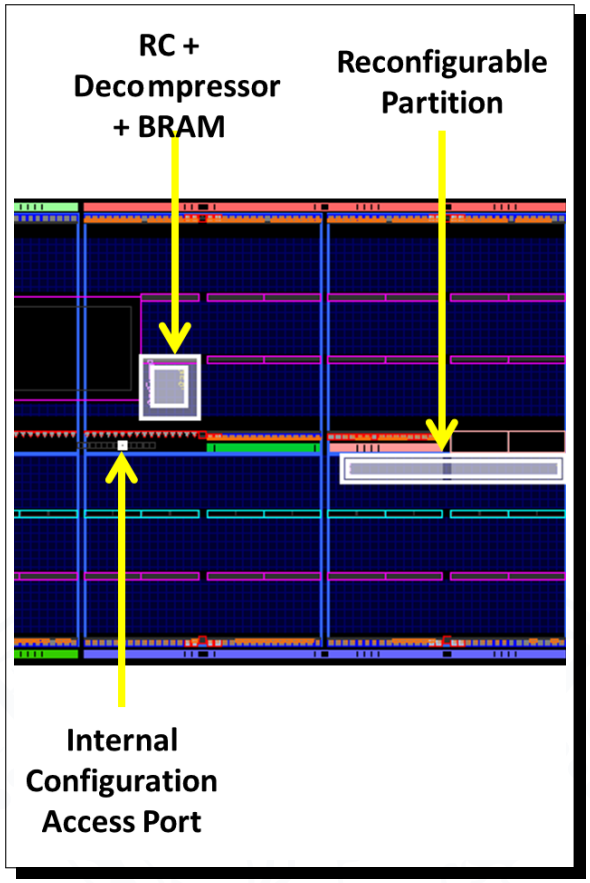


Figure 4.14: Example of protecting the critical blocks

50 different bitstreams.

The compression rate is computed as the ratio of compressed data plus the LUT (i.e., SGH tree) size over the original data.

Easily to conclude from Figure 4.15 and Figure 4.16 that, when word length configuration for Huffman encoding is larger than 8 bits, the overhead of Run-length encoding contributes a portion in the encoded bits and does not bring any benefit. However, the combination of Huffman encoding and Run-length encoding has a good impact when bit length is 8 bits. On overall, the compression rate is around 22% in all cases, it means that we could save 78% of storage size. The best compression rate that can achieved by the proposed solution is 19%, and the best choice is the combination of Huffman code and Run-length encoding with a word length of 8 bits.

The maximum working frequency of the decompressor is 163.55 MHz, with an area occupation of 189 slices (i.e., 3% of the device), including the reconfiguration controller.

Thanks to this implementation, since the ICAP works at 100 MHz, the decoder is able to provide

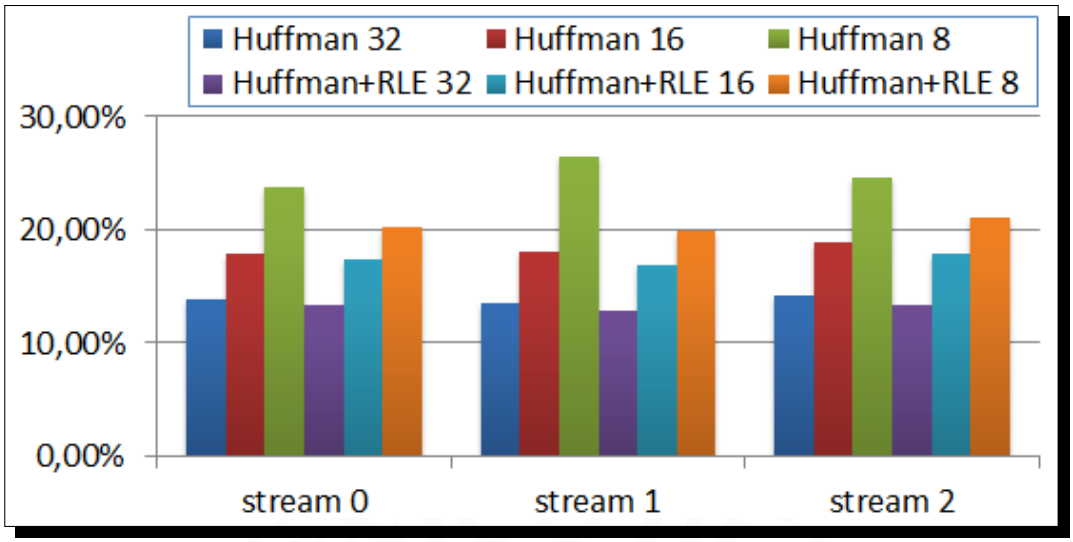


Figure 4.15: Compression rates for streams 0, 1 and 2

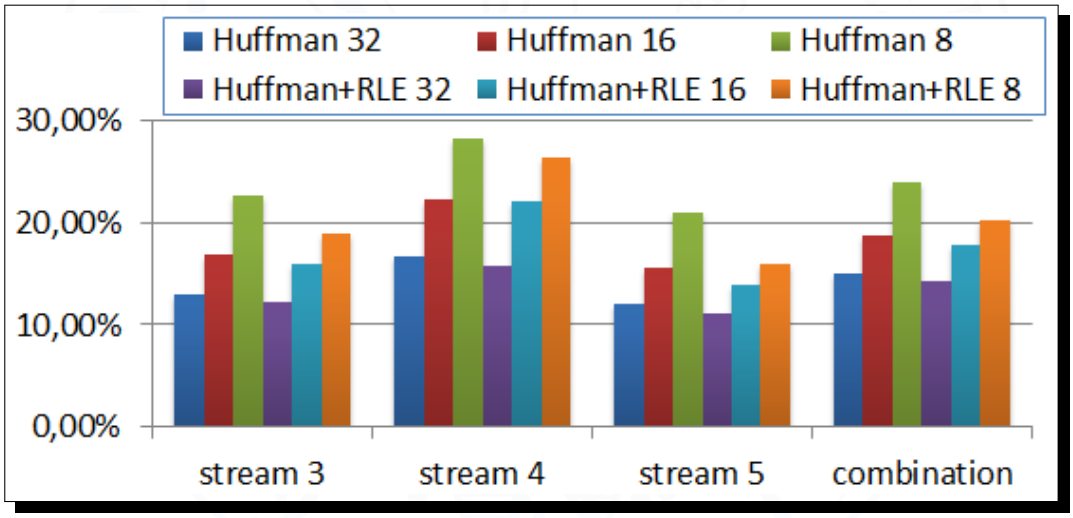


Figure 4.16: Compression rates for streams 3, 4, 5 and average

a 32-bit word each clock cycle, i.e., with the same throughput required by the configuration port, reaching the highest reconfiguration data throughput.
Finally, to prove the effectiveness of the ZipStream methodology different faulty bitstreams have been stored in an external memory and loaded to the ICAP. The internal 32 bit CRC module, which polynomial is *0x8F6E37A0*, is able to detect 3 faults in each bitstream [38]. In all cases the reconfiguration controller was able to restore the proper function of the static portion of the system, after the error notification asserted by the ICAP built-in CRC.

4.3 NBTI Mitigation by Dynamic Partial Reconfiguration

4.3.1 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. 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.

In order to assure that the function of the system is anyway carried out, in each couple of resources assigned to the same sub-function at least one of them has to be in *work* status. Since the resources working status has to overlap for a certain time, the static probability of the contents is slightly higher (or lower, depending if considering '1' or '0' probability and the specific memory cell) than 50%. To ensure the correctness of the methodology, the following three Design for Dependability rules should be fulfilled.

4.3.1.1 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 [17].

4.3.1.2 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:

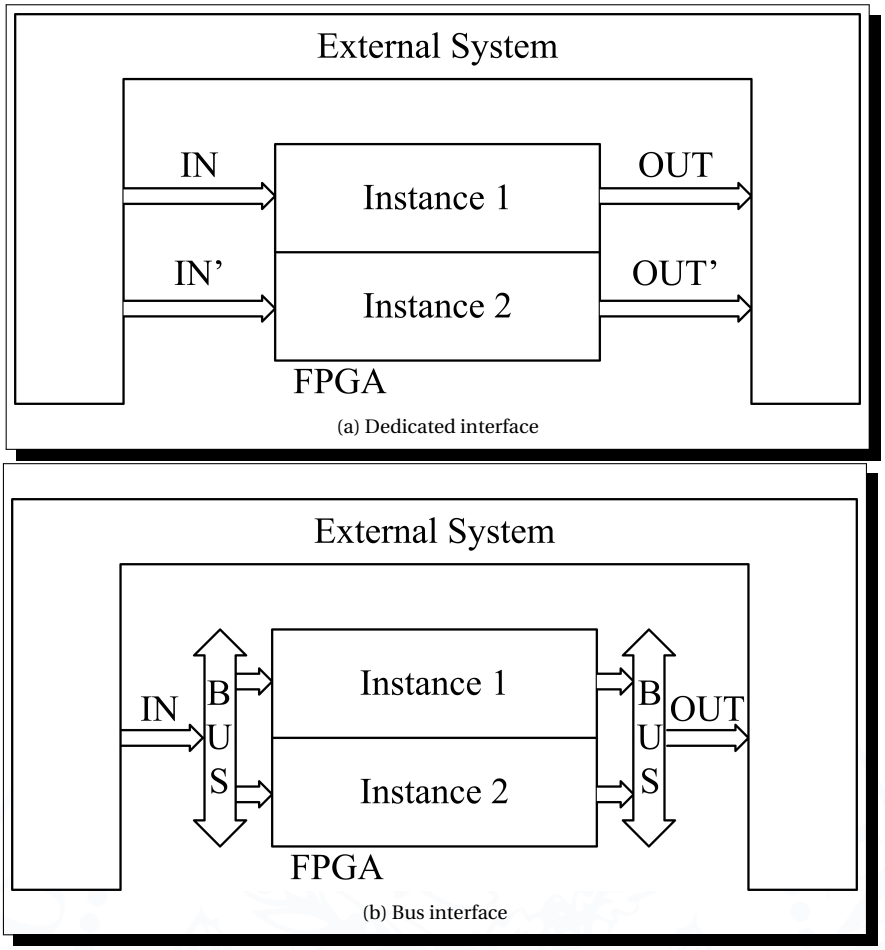
- the external user system (i.e., the system the FPGA is interfaced with) should provide the double of the minimum required number of ports and connect to each instance of a module in a FPGA with dedicated ports (Figure4.17(a));
- exploit an external bus (not laying in the configurable area) to connect the corresponding module ports and possibly, the ports of all the modules (Figure 4.17(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 two sets of I/O interfaces (e.g., *IN-OUT* and *IN'-OUT'* of Figure4.17(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. Further, the hypothesis to have an external bus connecting to many different ports of a programmable area is quite likely true, as it is an actual implementations where FPGA logic is used as hardware accelerators.

4.3.1.3 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).



(a) Dedicated interface

(b) Bus interface

Figure 4.17: Possible connection schemes

This controller, cannot be implemented in the reconfigurable logic of the FPGA as it would imply static connections passing across different partitions' boundaries which are potentially damaged during partial reconfiguration of a module. For this reason, an external controller should be used.

If a (micro)processor is involved in the design and enough computation power is available, the controller's functions can be allocated to the former: using an internal timer the processor could simply generate the correct times for *work* and *rest* and drive.

4.3.2 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 [64] EPP by Xilinx

to implement an embedded system having two IP-core modules.

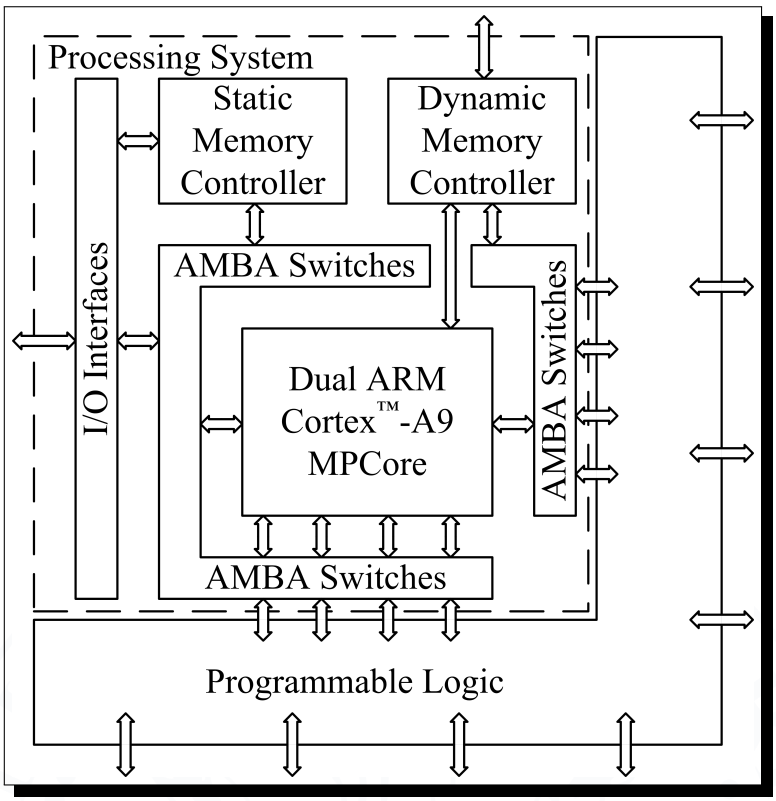


Figure 4.18: Zynq™Architecture

The Zynq™-7000's architecture perfectly applies to the conceived method. As a matter of fact, as shown in Figure 4.18, 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 Section 4.3.1.2, AMBA is used to deal with communication between the FPGA and the external system.

The function of the controller, described in Section 4.3.1.3, 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.

The processor, beside running the main application's program, by using multiprogramming, concurrently runs the code to implement the controller. This is made easier by using timer counters and interrupt controller to obtain the required time intervals in the states switching. Furthermore, DMA controller is used to transfer the bitstreams from the memories to ICAP controller.

Reconfiguration control procedures are inserted in the real-time interrupt (generated by timer counters) service routines, in such a way that they cannot be stopped by the application software and so preventing hazards derived from running the application while switching the states (i.e., while driving the partial reconfiguration).

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 Section 4.3.1.1. Let us consider that the system is initially configured in the state (*IPx*, *IPx'*)= (*work*, *rest*) for each IP-core, as shown in Figure 4.19(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 $BS2'$).

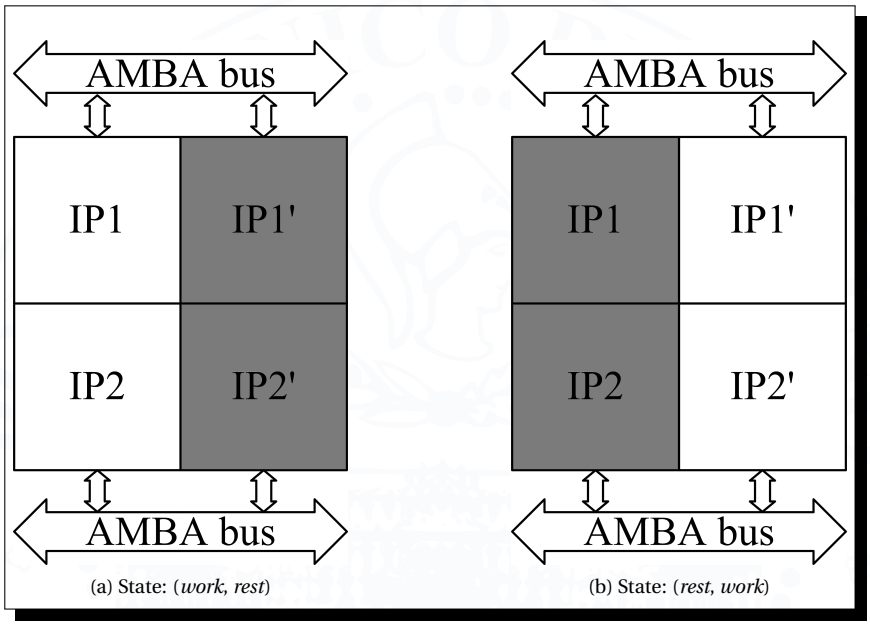


Figure 4.19: Graphic representation of IP-core states within the FPGA

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*) (Figure 4.19(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 con-

figuration memory's bits 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 Figure 4.19). The switching frequency of each module was chosen to be compatible with the foreseen idle times of the cores.

4.3.3 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 SNM 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 [70], 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, $V_{DD} = 1.1$V and channel width $W_{pMOS} = 0.21\mu$m.

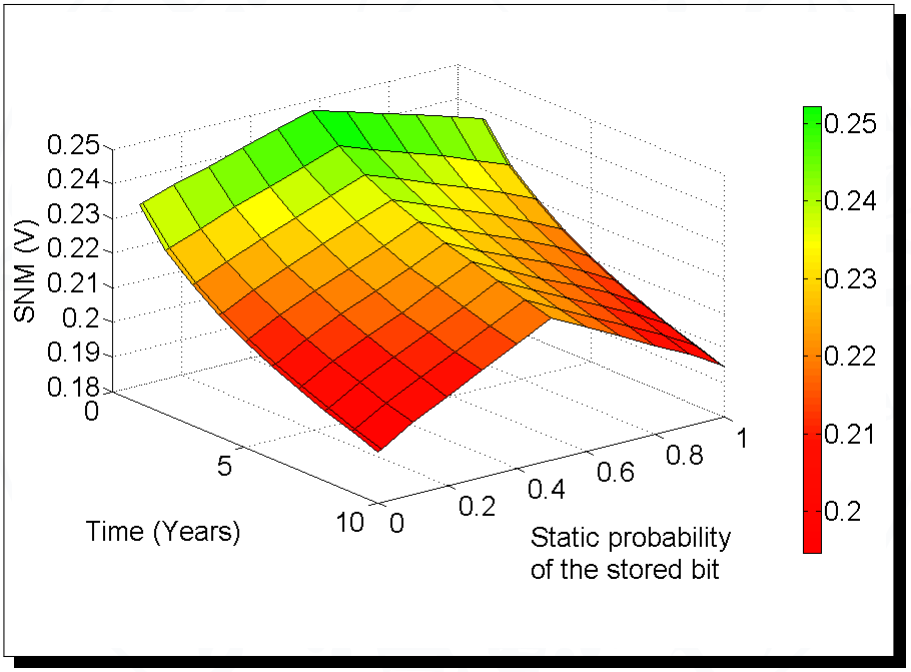


Figure 4.20: Signal Noise Margin degradation in function of time (measured in years) and static probability (probability to have '1' in a SRAM cell)

Figure 4.20 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.

4.4 SATTA: a Self-Adaptive Temperature-based TDF awareness methodology for dynamically reconfigurable FPGAs

In a complex FPGA-based SoPC, high clock frequency and precise timing optimization are crucial to guarantee high performance and throughput. Designers try to push systems' working frequency to its limit, which is mostly settled by the maximum path delay of the design. Proper *set-up* (T_{setup}) and *hold* (T_{hold}) times at the input of each flip-flop must be guaranteed in order to sample stable signals, thus avoiding metastable states and sampling of wrong values belonging to previous or following clock cycles.
Faults due to incorrect sampling caused by timing issues are in general referred to as *Transition Delay Faults* (TDFs). Figure 4.21 shows a typical TDF example in which the signal fed to a sampling register violates the set-up time.

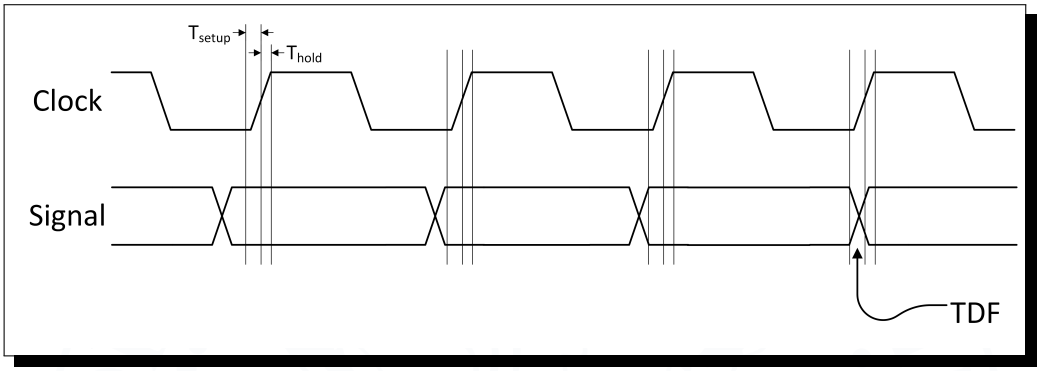


Figure 4.21: Example of Transition Delay Fault. A signal connected to the input of a register or flip-flop violates the set-up time changing its value in correspondence to the clock rising edge and thus increasing the probability of sampling a wrong value.

TDFs are often caused by non-functional parameters such as process variations, voltage fluctuation, high temperature and aging. Aging effects, such as NBTI and HCI may deteriorate the system's timing during its lifetime (see Section 2.2) High temperature is another significant TDF cause. It magnifies aging effects such as NBTI and HCI and, furthermore, it directly impacts the transistors' delay by increasing their threshold voltage. Park et al. [49] experimentally demonstrated that working frequency penalty triplicates when changing the operating temperature from 25 °C to 125 °C. Solutions targeting TDFs avoidance and detection have been deeply studied in ASICs. Dynamic Frequency Scaling (DFS) [22] consists in dynamically decreasing the system's working frequency when the path delay increases. TDFs are avoided by keeping a guard-band on the set-up time. On the other hand, Dynamic Voltage Scaling (DVS) [14, 15] relies on increasing the voltage source to increase the drain current, thus compensating the performances losses. Another approach to increase drain current capability consists in modifying the body bias voltage, as in [23]. DVS and DFS have also been merged to maximize their mitigation efficiency [53].

Adapting ASIC approaches to FPGA-based systems is non-trivial, due to the fixed FPGA internal architecture that restricts the set of available design solutions. A set of works propose the use of sensors based on shadow registers with clock skewed with respect to the primary one in order to detect delay variations [6, 43, 56]. [6] proposes a sensor for detecting late transitions due to aging effects in FPGAs. The sensor is designed in order to be inserted in parallel at the endpoint of the critical path and exploits two flip-flops wit a positive clock phase to detect late transitions. One of the main drawbacks of this sensor is that it enables to detect TDFs when they arise and therefore it requires complex recovery strategies that usually imply stalling the system. Differently from [6], [43] proposes to use negative clock skew to detect delay variations thus enabling to identify paths that are reaching a critical delay level before actual faults arise. The goal of the proposed technique is to accurately and precisely characterize the register-to-register delays of a large number of otherwise unobservable combinational paths at test-time. Even if not explicitly defined for FPGA designs, the method has been applied for the characterization of a FPGA based test case. A similar idea has been presented in [56], where a similar sensor is employed to detect the likelihood of TDFs occurrence in selected critical paths. Despite we recognize that the use of sensors based on shadow registers with clock negatively skewed with respect to the primary one is a very efficient method to detect delay variations, in these publications, no counteractions at system level are presented to exploit the gathered information at run-time to avoid TDFs occurrence in FPGA designs. Furthermore, these solutions are characterized by a constant monitoring of the observed paths, thus incurring in sensor self-aging and power waste due to clock duplication, if used for run-time monitoring of FPGA aging. Being aware of the TDF main causes and considering the models beyond their occurrences are crucial to efficiently tackle the problem of TDF avoidance. Nonetheless, static probability of signal values and temperature profile of a system depend on the actual workload and on the environmental conditions [74]. They are therefore hard to estimate at design time [5]. This forces designers to take worst-case decisions when selecting the system's working frequency.

This methodology tries to overcome these limitations presenting SATTA, a design solution able to exploit run-time device temperature information to predict delay variations and to smartly activate temperature sensors, thus reducing power overhead. Furthermore, SATTA manages the dynamic reconfiguration of the system's working frequency, taking into account the delay dependency on the temperature and the transitory nature of aging phenomena (due to signal probabilities and temperature profiles). The proposed methodology has the potential to efficiently avoid TDFs, guaranteeing a graceful degradation of system's performance when required, but restoring or boosting it in absence of critical conditions. A manager, implementing the proposed methodology, is introduced in this paper, in order to easily apply the methodology to FPGA-based SoPCs.

4.4.1 SATTA sensors organization and architecture

SATTA implements a TDFs avoidance strategy based on the monitoring of the delay of those signals routed through the critical paths of the FPGA design. Two main events may arise:

1. whenever, due to increased temperature and/or aging effects, the delay of a critical path approaches the set-up sampling time limit, the system's clock frequency is lowered;

2. whenever, due to decreased temperature and/or recovery from aging effects, the critical path delay decreases, the system's clock frequency is slowly increased again.

A set of hardware modules are required to efficiently implement this monitoring activity and the related reaction policies. These modules are designed exploiting a set of features commonly available in modern reconfigurable devices, thus minimizing area, power and performance overheads. Resorting to widely spread FPGA functionalities enables easy implementation on all modern dynamically reconfigurable FPGAs, such as the newest Xilinx Virtex [70] or Altera Stratix [3] FPGAs.

Figure 4.22 graphically summarizes the overall SATTA architecture. It includes:

- One or more *Temperature sensors*. They measure the temperature of the device and provide this information to the manager that exploits it to estimate the system's path delay status.

- One or more *TDF sensors*. A TDF sensor is able to monitor the delay of a given path asserting a warning signal whenever it increases to a level that may compromise the correct sampling of the information.

- A *CLK Generator*. It generates the set of clock signals (i.e., CLK1 and CLK2) required by the TDF sensors to properly operate. It enables to dynamically change the phase relationship between CLK1 and CLK2. This feature is exploited by the manager to measure the actual delay in the observed paths. Moreover, it can dynamically modify the frequency of the main system's clock (CLK1) at run-time, exploiting FPGA Dynamic Partial Reconfiguration (DPR). This enables to adapt the system to the temperature and aging conditions, avoiding TDFs while maximizing the performance.

- A *Manager*, that collects information acquired from the sensors (i.e., *TDF sensors* and *Temperature sensors*) and issues proper reconfiguration commands for the *CLK Generator* without interrupting the system's tasks.

Details about the implementation of each block are provided in the next sections.

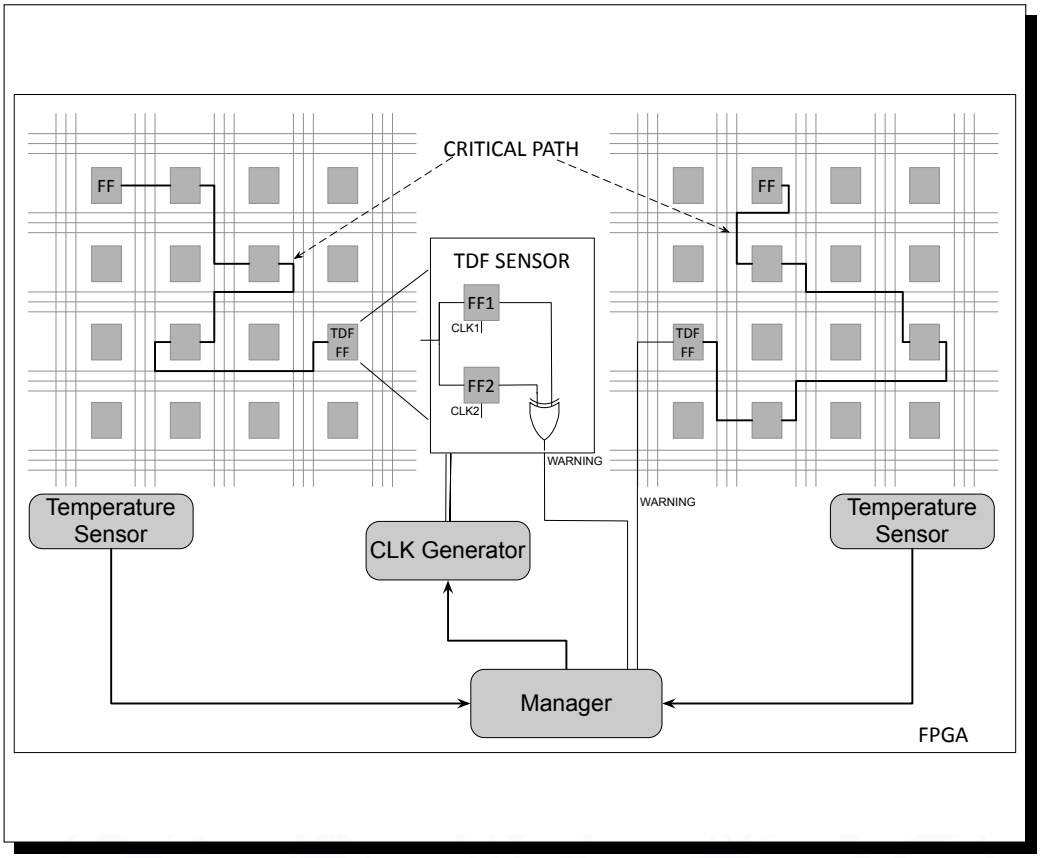


Figure 4.22: SATTA overall architecture. TDF detection and mitigation is implemented inserting in the design: (i) a set of temperature sensors, (ii) a set of TDF sensors, (iii) a clock generator, (iv) and a global manager.

4.4.1.1 Temperature Sensor

Temperature has a strong impact on aging effects and on TDFs in general. An increase of the FPGA temperature would, on the one hand, heighten the path delay and, on the other hand, raise the transistor's threshold voltage due to NBTI.
Figure 4.23 plots the trend of a path delay in a Virtex 4 device, depending on the die temperature, using the temperature model provided by Xilinx, extracted with the aid of the timing analysis tool. Figure 4.23 shows an increase of more than 300 ps when the operating temperature raises from 0°C to 85°C. Similar effects are also experimentally measured on different devices such as 45 nm Spartan 6 and 28 nm Artix 7 devices [50, 51]. Device temperature must therefore be monitored, and gathered information must be exploited to predict the delay behavior and to take proper countermeasures.

Both Altera and Xilinx FPGAs include a built-in temperature sensor diode. However, this

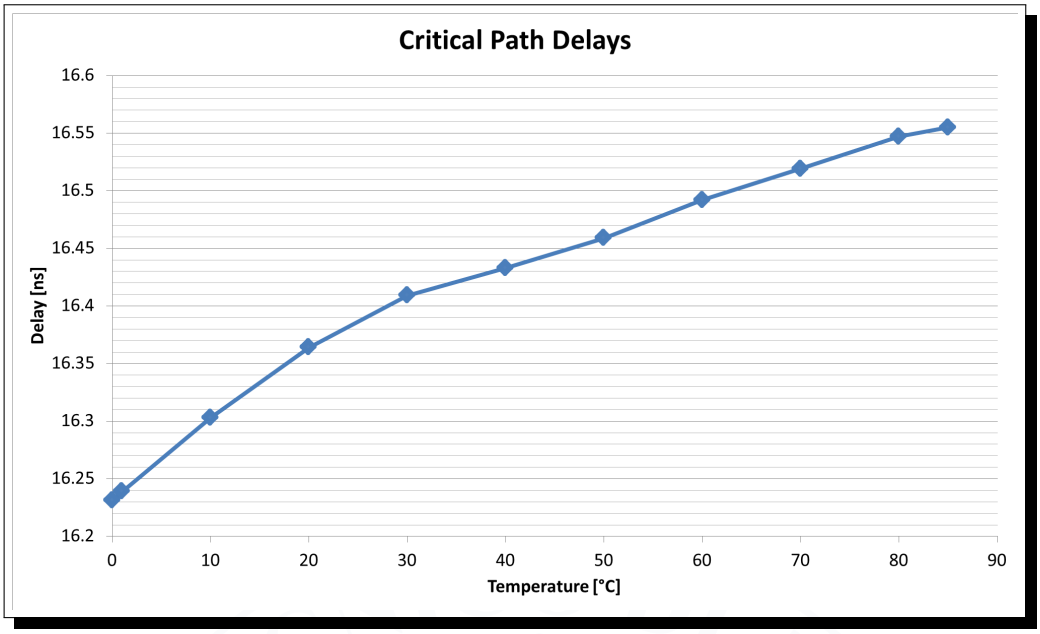


Figure 4.23: Example of a critical path delay variation w.r.t. temperature in Virtex 4 Devices

diode is located in a fixed position of the die and is unable to provide measures related to a specific path. Following [26], SATTA exploits the built-in temperature diode as a reference sensor and a ring oscillator with odd number of inverters (i.e., 11 inverters mapped on different FPGA slices) as an approximated temperature sensor. This structure has several advantages: (i) it does not require device dependent hardware features, (ii) it enables to measure temperature with a reasonable area occupation and accuracy, and (iii) it enables to instantiate several sensors in the device, monitoring the temperature in the area in which the controlled critical path is routed. Looking at Figure 4.23, it is worth to highlight here that the temperature can be measured with an uncertainty in the measure of some degrees without introducing significant errors in the corresponding delay estimation. This enables us to strongly reduce the number of cycles required to obtain a measurement from the ring oscillator. Therefore, differently from [26] that uses 2^{17} measurement cycles, SATTA can perform measurements with a drastically reduced number of cycles (i.e., few tens of cycles are sufficient to have a measure suitable for the SATTA purposes).
If the available hardware resources are very limited, the reference diode alone can still be used to evaluate the temperature in the device. In this case, temperature information provided by the diode refers to the die temperature. Even if it is less specific, this information can still be used to evaluate the overall temperature at which the device works. It is important to highlight here that sensors are subject to aging as well. Therefore, the temperature should not be constantly measured to avoid both sensor self-aging and to decrease power consumption.

If more built-in temperature sensors spread on the FPGA will be available in next generation

devices, they can be exploited to obtain fast and accurate temperature measurements and to save area dedicated to ring oscillators used as temperature sensors. Nevertheless, these sensors require the use of built-in analog-to-digital converters available on the FPGA. These converters represent limited resources often required to interface the FPGA with external devices. Ring oscillators remain a viable general solution whenever these resources must be allocated to different tasks related to the system's mission.

4.4.1.2 TDF Sensor

Once the most critical paths of the design have been identified, a TDF sensor for each critical path is inserted. The TDF sensor is a special block able to monitor the delay of a path. Each critical path includes several levels of combinational logic and is delimited by two memory elements (e.g., flip-flops) working at the system's clock frequency. The TDF sensor replaces the flip-flop placed at the end of the selected critical path.

Following previous works [43, 56] we exploit a sensor based on a shadow flip-flop clocked with a negative clock skew with respect to the main system's clock. The sensor is composed of two flip-flops (FF1 and FF2) and a 1-bit XOR gate (Figure 4.22). It is connected to the selected path and clocked with two different clock signals. Since the two flip-flops are clocked with two different clock signals, they sample the critical path output at two different times (see Figure 4.24). FF1 is clocked at the system's clock *CLK1* and its output is connected to the rest of the system. Instead, FF2, clocked at *CLK2*, acts as a warning sensor. *CLK2* is a replica of *CLK1* shifted with a negative phase, in order to sample the data before *CLK1*.

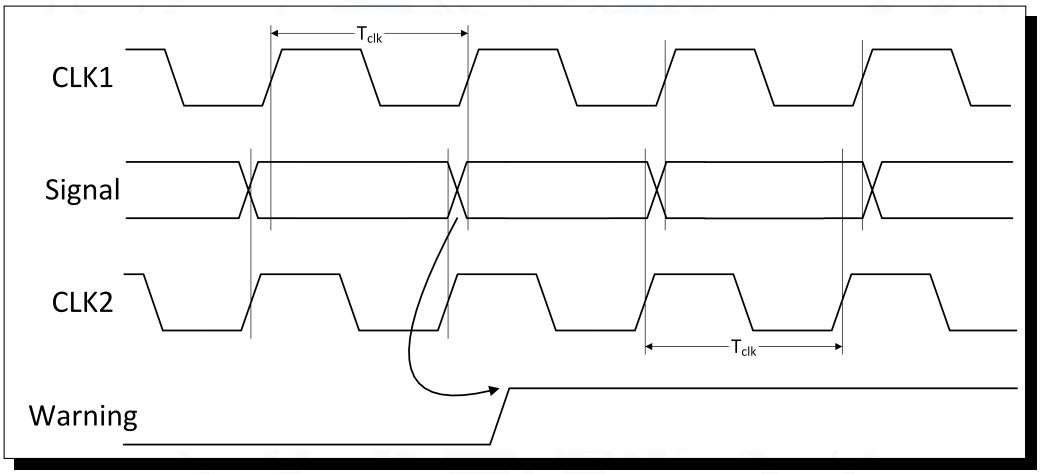


Figure 4.24: Behavior of the TDF sensor

The output of the two flip-flops is compared using the XOR gate and a warning signal is asserted if the two sampled values differ. This happens if the sampled signal stabilizes too close to

the rising edge of the system's clock, indicating that the path delay is increasing due to temperature variations, aging effects, or presence of glitches.

Setting a negative phase between *CLK1* and *CLK2* is pivotal to obtain fault avoidance. When the path delay increases, metastability conditions arise in FF2 first. If timely detected, the SATTA *Manager* can take the proper countermeasures to prevent that the path timing degrades at a level in which also FF1 is compromised generating an error in the system. The way this countermeasures are implemented will be better detailed in the next sections.
To increase the accuracy of the TDF sensor, FF1 and FF2 must be placed close to each other in the design, thus avoiding timing differences that may cause erroneous TDFs warnings. Differently from previous approaches [56], that rely on post-placement manipulation to add the desired register by hand, we designed a procedure to easily add TDF sensors at RTL level while respecting this constraint (see Section 4.4.2 for details on the sensors insertion methodology).

4.4.1.3 CLK Generator

The *CLK Generator* is responsible for generating both the main system's clock *CLK1*, and the negative shifted clock *CLK2*. These two signals are generated using reconfigurable *Phase Locked Loop* (PLL) hard-macros available in modern FPGAs [3, 70].

Reconfigurable PLLs have two main characteristics: (i) they can be configured at run-time, thus enabling to adapt the system's clock frequency and the clock phase difference according to the *Manager* requests, and (ii) they enable the generation of multiple clock signals. Nevertheless, the main problem of these PLL hard-macros is that, in general, they must be reset after each reconfiguration. To avoid any interruption of the system's operation, two PLLs are therefore used. Figure 4.25 shows the internal architecture of the *CLK Manager* implemented in a Xilinx 7 Series FPGA [70] (similar considerations and architectures also apply for Altera FPGAs). It includes two Xilinx Mixed Mode Clock Managers (*MMCM*) hard-macros and two clock multiplexers [72]. Each *MMCM* can synthesize one or more clock signals starting from a reference input clock. Each output clock signal can have independent frequency and phase relationship with respect to other outputs.
In the proposed *CLK Manager* architecture, each *MMCM* generates two clock signals (*CLK11*/*CLK12* for *MMCM1* and *CLK21*/*CLK22* for *MMCM2*).

At system's start-up, *MMCM1* is selected as main clock source and therefore *CLK11* is used as main system's clock while *CLK12* represents the negative shifted secondary clock, required by the TDF sensors.

Whenever the *Manager* detects a warning from one of the TDF sensors, it reconfigures *MMCM2* through its Dynamic Reconfiguration Port (*DRP*) [70]. The reconfiguration process sets new clock frequency parameters, while the first *MMCM* is still operating with the current frequency.

When the dynamic reconfiguration of *MMCM2* is completed, it is activated and the clock mul-

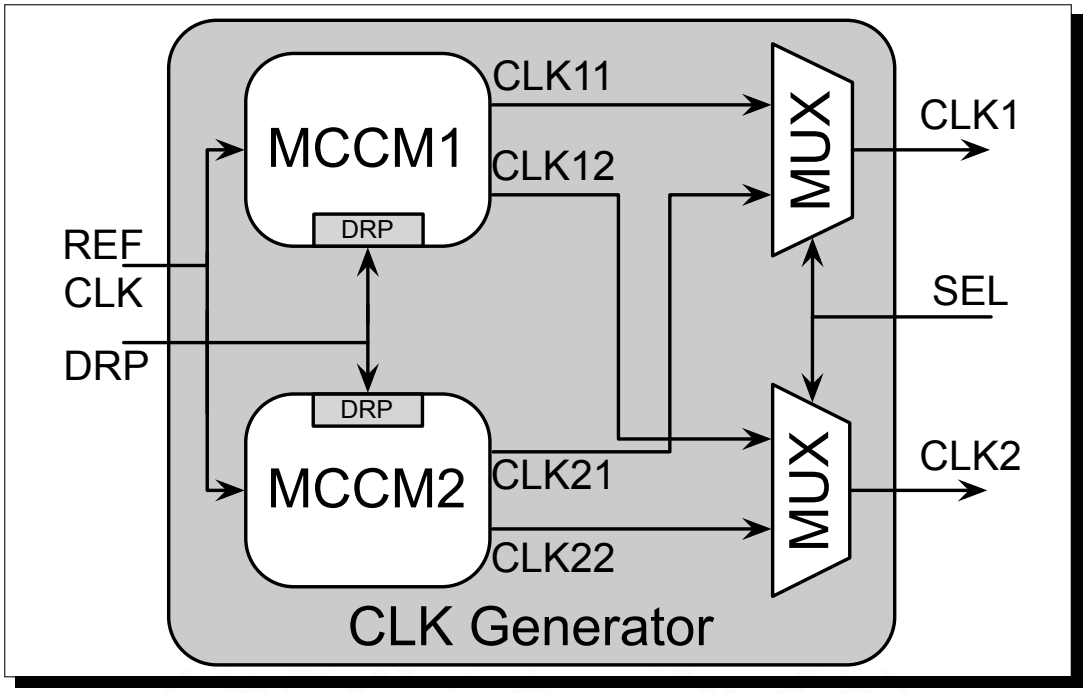


Figure 4.25: Clock Generator internal architecture

tiplexers are both switched to modify the system's clock frequency, therefore setting *MMCM2* as clock source. Similar operations are performed, alternatively switching the role of *MMCM1* and *MMCM2*, whenever a new reconfiguration is required. The inactive *MMCM* is powered down to reduce the power consumption of the clocking network.

It is important to highlight that the switching between the two clock sources exploits the hard macro *BUFGMUX* of Xilinx FPGAs [72], that is able to avoid metastability problems. It eliminates the need to stall the system clock, thus interrupting the system operations. The reader may refer to [72] for more information and detailed time diagrams of the *BUFGMUX* behavior.

The proposed *CLK Generator* architecture represents an improvement with respect to the scheme presented in [56], in which two clock generators are used, as well. The main difference between the two architectures is that in [56] the system clock and the auxiliary clock are generated by two different clock managers, thus introducing uncertainty in the clock phase difference. Since the two clock managers are placed in different portions of the FPGA, process variations and temperature gradients can cause different skew and jitter effects between the two clock signals. The *CLK Generator* architecture presented in this paper does not incur in power penalty with respect to [56]. In fact, only a single *MMCM* is active during the normal operation, while both *MMCMs* are turned-on only during a short transient phase (i.e., few tens of clock cycles for

reconfiguration).

4.4.1.4 Manager

The Manager coordinates the activities of all the described blocks (i.e., TDF sensors, temperature sensors, and CLK generator) according to the Algorithmic State Machine (ASM) reported in Figure 4.26.

At the beginning, in the RESET state the system is initialized. The most important parameters managed by this block are:

- CLK_FREQ: represents the actual working frequency of the system;
- CLK_PHASE: is the actual phase relationship between the two clocks in output from the *Clock Generator*, evaluated as the ratio between the skew of *CLK1* and *CLK2* and their period;
- TEMP: represents the actual measured device temperature;
- TIMER_VALUE: represents the timeout period of an internal timer used to switch on the monitoring procedure.

The value of the different constants and thresholds (i.e., INIT_FREQ, INIT_PHASE, ΔCLK_DWFREQ) depends on the specific design and user specifications, and must be customized taking into account synthesis parameters (see Section 4.4.3).

Starting from its IDLE state, the Manager is activated at regular intervals by an internal timer (TIMER_VALUE and TIMER_END represent the timeout period and the flag, which is set at the end of this period). Each time the manager is switched-on (i.e., each time the timer reaches its time-out value) all temperature sensors are activated and, based on their readings, a new value of TIMER_VALUE is computed (TEMP_S state). If multiple temperature sensors are instantiated, only the highest temperature value is taken into account for TIMER_VALUE computation.
Basically, the temperature and TDF sensors activation period depends on: (i) the actual operating frequency of the circuit, (ii) the operating temperature, and (iii) the temperature variation speed. The sampling period must always respect the condition that the maximum increment of the path delay between two consecutive readings must not exceed the phase relationship delay between the two clocks sourcing TDF sensors (i.e., *CLK1* and *CLK2*) according to the following equation:

$$TIMER\_VALUE : \Delta\tau = \tau(t + TIMER\_VALUE) - \tau(t) < \overbrace{CLK\_PHASE \cdot T_{CLK1}}^{\text{Clock skew}} \quad (4.5)$$

where $\Delta\tau$ is the maximum increment in the path delay between two sensors activations, T_{CLK1} is the actual clock period of the system, and *CLK_PHASE* is the actual phase relationship, evaluated as the ratio between the skew of *CLK1* and *CLK2* and their period. This condition ensures that,

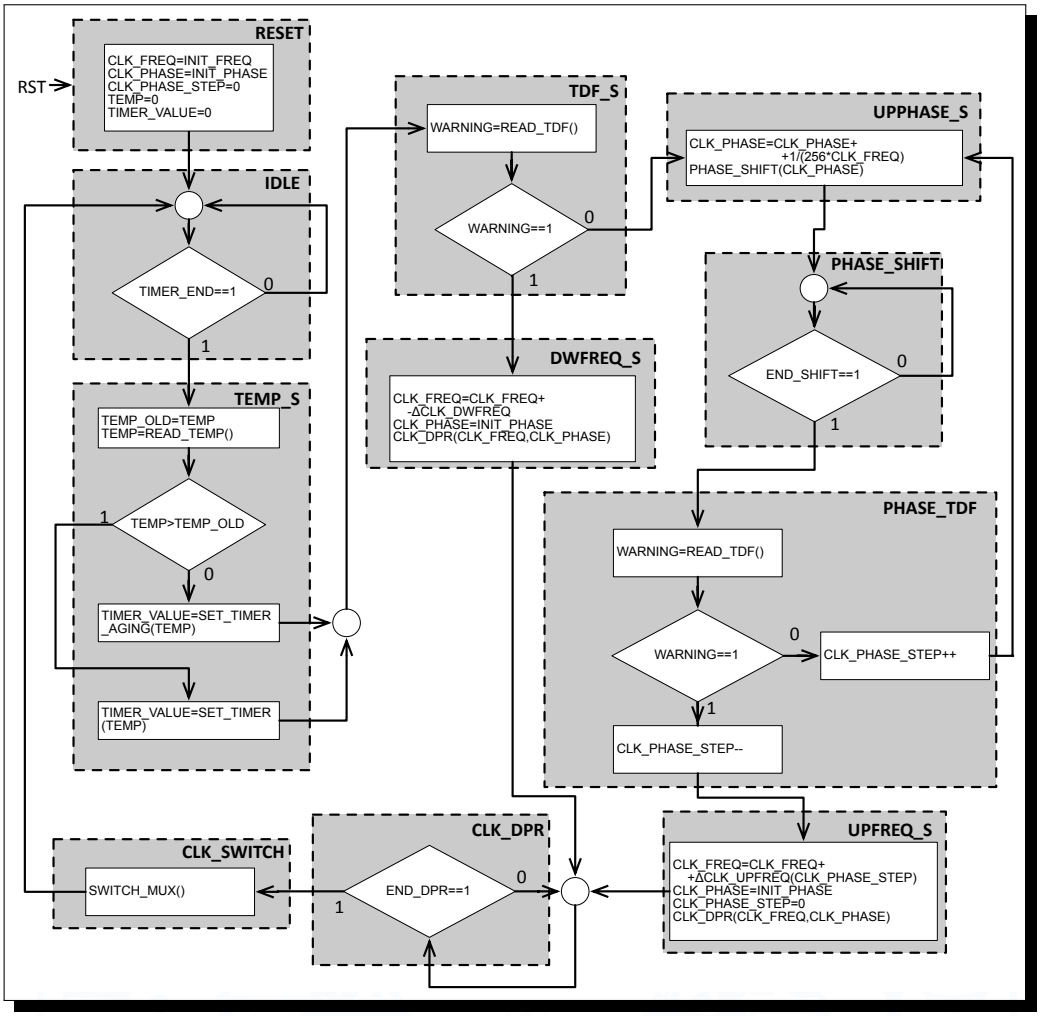


Figure 4.26: ASM describing the behavior of the Manager

between two sensor readings, the increment of path delay will not cause an actual TDF. However, before computing the new TIMER_VALUE (TEMP_S state), the new value of temperature is compared with the last measured temperature (TEMP_OLD). The case TEMP>TEMP_OLD represents the worst condition for the system, since an increase of temperature has immediate impact on the delay variation. The temperature sensor must therefore be activated at short intervals to guarantee proper adaptation of the system without incurring in transition delay faults. Differently, if TEMP<=TEMP_OLD, the delay increment can be only generated by aging effects, whose impact is in general much slower than temperature variation effects. Larger activation intervals for the temperature sensors can thus be setup. The SET_TIMER and SET_TIMER_AGING functions compute the new value of the timer based on the sign of ΔTemp=TEMP-TEMP_OLD

and on the actual temperature value (i.e., TEMP in Figure 4.26).

As aforementioned, the circuit temperature variation speed directly impacts on the timer sampling period. However, this parameter strongly depends on the device technology, on the circuit implemented in the FPGA device, and on the actual workload. To extract this parameter, the user should measure how the temperature profile of the circuit changes during a representative execution time interval [26]. If this information is not known, the designer can choose to adopt sampling periods that are orders of magnitude higher than the time constant with which the temperature changes (tenths of a second, or seconds, depending on the aforementioned parameters).

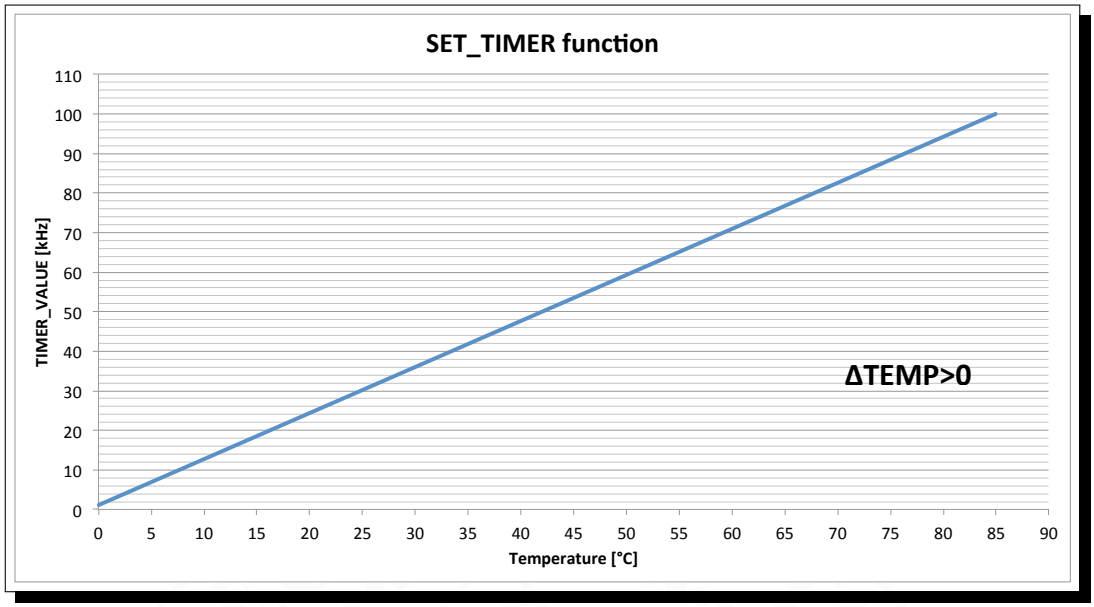
In particular, we adopt a linear function for both positive and negative ΔTemp (i.e., SET_TIMER and SET_TIMER_AGING functions), according to Figure 4.27(a) and Figure 4.27(b).

The difference between the two functions is the value of the sampling period (TIMER_VALUE), which spans, in the operating temperature range (0-85°C), from 1 kHz to 100 kHz and from 10 Hz to 1 kHz for positive and negative ΔTemp, respectively. In both cases, the higher is the temperature, the higher will be the sampling frequency (i.e., TIMER_VALUE). Moreover, the lower sampling frequency in the case ΔTemp<=0 leads to a saving in SATTA modules power consumption, with respect to the case ΔTemp>0.

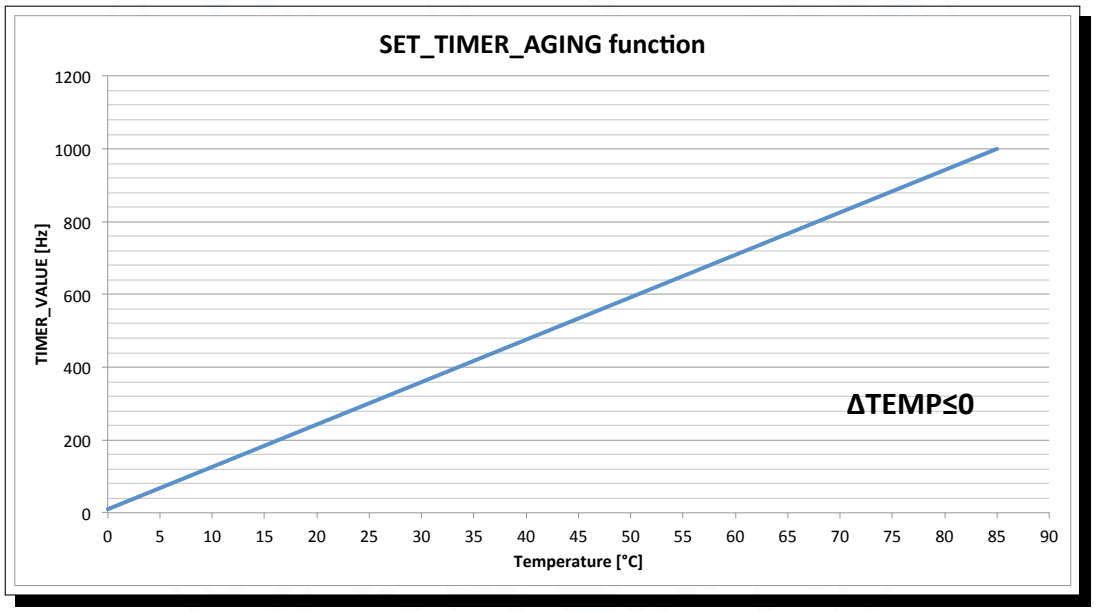
After TIMER_VALUE computation, the manager enters the TDF_S state in which all TDF sensors are enabled. The outputs of all TDF sensors are OR-ed, and if at least one sensor issues a warning condition (DWFREQ_S state) the system's clock frequency (CLK_FREQ) is lowered by a given step (ΔCLK_DWFREQ) and the phase difference between the two clocks of the CLK generator (CLK_PHASE) is set to a predefined value (INIT_PHASE). A DPR command is therefore issued to the CLK Generator (CLK_DPR state) to change the operating frequency. When the configuration is complete, the CLK Generator multiplexers are switched (CLK_SWITCH state) and the *Manager* returns to the IDLE state, waiting for a new sampling cycle.

If no warning signal is raised, the *Manager* investigates the possibility of increasing the phase difference between the two clocks (i.e., the TDF sensor one and the standard one) in order to understand which is the maximum frequency the system can actually support. In this procedure, composed of the UPPHASE_S, the PHASE_SHIFT, and the PHASE_TDF states, the *Manager* tries to increase the phase difference between the two clocks generated by the *CLK Manager* up to a limit in which a warning is issued. This is obtained by iteratively increasing CLK_PHASE. In our specific Xilinx implementation of the *Manager*, the phase is increased of 1/256 of the actual system's clock period at every iteration (it is the phase resolution limit of the *MMCM*). As soon as a warning signal is detected, in the PHASE_TDF state, the limit has been already exceeded, so the previous value of CLK_PHASE_STEP is used to modify the working frequency (UPFREQ_S state). The new boosted clock frequency is therefore computed according to the following equation:

$$CLK\_FREQ = \frac{CLK\_FREQ}{1 - \frac{CLK\_PHASE\_STEP}{256}} \tag{4.6}$$



(a) SET_TIMER function



(b) SET_TIMER_AGING function

Figure 4.27: Sampling periods with respect to temperature

The CLK_PHASE_STEP decrement (before leaving the PHASE_TDF state) is necessary in order to increase the working frequency in a dependable way, leaving a margin with respect to the actual maximum working frequency (Equation 4.6).

4.4.2 SATTA integration methodology

In order to gather information and manage reconfigurations, SATTA needs to define a set of design rules to properly automate both sensors insertion and system integration. Figure 4.28 highlights the overall SATTA integration flow.

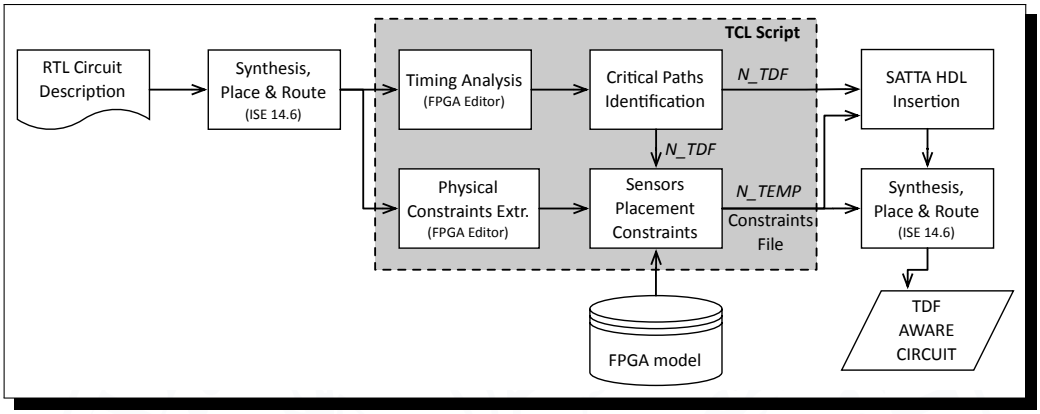


Figure 4.28: SATTA integration flow. It enables to automate the insertion of all sensors and structures required to implement the TDF detection and compensation.

SATTA integration starts from the synthesis and implementation of the RTL model of the target system given a target FPGA. After the placement and routing phase, each design path is analyzed in order to calculate the slack (i.e., the difference between the clock period and the actual path delay) using a timing analysis tool (e.g., *PlanAhead* or *FPGA Editor* tools for Xilinx FPGAs). For instance, the timing analyzer tool embedded in Xilinx PlanAhead can generate a text *timing report (.twr)* containing the slack information for each path with respect to the maximum allowable delay (i.e., the clock period). In particular, for each path are also listed the source and destination Flip-Flops (FFs). Figure 4.29 shows a snippet of the timing analysis output file extracted using the Xilinx Timing Analyzer tool, highlighting the lines containing the slack values for the analyzed paths. It is worth to note that each destination FF (e.g., S_30 and S_reg29 in Figure 4.29) can be associated to several design paths. In this case, paths with the same destination FF present different source-destination delays since they are sourced by different FFs (i.e., B_reg_4, B_reg_10 and A_reg_7 in Figure 4.29) spread on the FPGA physical area (thus leading to different logic and routing delays).

```
Slack (setup path):     0.493ns (requirement - (data path - clock
path skew + uncertainty))
  Source:               B_reg_4 (FF)
  Destination:          S_30 (FF)
  Requirement:          5.500ns
  Data Path Delay:      4.912ns (Levels of Logic = 4)
  Clock Path Skew:      -0.060ns (0.831 - 0.891)
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP rising at 5.500ns
  Clock Uncertainty:    0.035ns

Slack (setup path):     0.496ns (requirement - (data path - clock
path skew + uncertainty))
  Source:               B_reg_10 (FF)
  Destination:          S_30 (FF)
  Requirement:          5.500ns
  Data Path Delay:      4.909ns (Levels of Logic = 4)
  Clock Path Skew:      -0.060ns (0.831 - 0.891)
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP rising at 5.500ns
  Clock Uncertainty:    0.035ns

Slack (setup path):     0.497ns (requirement - (data path - clock
path skew + uncertainty))
  Source:               A_reg_7 (FF)
  Destination:          S_30 (FF)
  Requirement:          5.500ns
  Data Path Delay:      4.903ns (Levels of Logic = 3)
  Clock Path Skew:      -0.065ns (0.831 - 0.896)
  Source Clock:         clk_BUFGP rising at 0.000ns
  Destination Clock:    clk_BUFGP rising at 5.500ns
  Clock Uncertainty:    0.035ns

                        ......
                        ......
                        ......

Slack (hold path):      0.484ns (requirement - (clock path skew +
uncertainty - data path))
  Source:               B_reg_25 (FF)
  Destination:          S_reg29 (FF)
  Requirement:          0.100ns
  Data Path Delay:      0.915ns (Levels of Logic = 2)
  Clock Path Skew:      0.496ns (2.000 - 1.504)
  Source Clock:         clk_BUFGP rising at 5.500ns
  Destination Clock:    clk1_BUFGP rising at 5.400ns
  Clock Uncertainty:    0.035ns
```

Figure 4.29: Example of a timing analysis report generated using the Xilinx Timing Analyzer tool.

A script can automatically parse the .twr file to extract all values contained in the *Slack (setup path)* lines, annotating also the source and the destination points, to identify the related paths in the design. The lower is the slack, the more a path is critical. The selection of the most critical paths can be done by analyzing the distribution of the slack values with respect to the clock period (see Section 4.4.3 for some distribution examples and how critical paths can be selected). The number of identified critical paths (presenting different destination FFs) defines the number of TDF sensors needed to protect the circuit against transition delay faults (*N_TDF* value in Figure 4.28).

Afterwards, physical placement and routing information of the implemented original design are extracted and stored as a user physical constraint file (.ucf). This task can be performed exploiting, for instance, the *Xilinx FPGA Editor* tool. This tool is able to read the implemented design netlist and to extract the placement information for each LUT and FF used in the FPGA device. Figure 4.30 shows a snippet of the .ucf file, where each design resource is allocated to a

LUT or a FF in a slice.

```
INST "ADDER1_gen[13].FAI/Co1_SW8" LOC=SLICE_X66Y80;
INST "ADDER1_gen[13].FAI/Co1_SW8" BEL="A5LUT";
INST "ADDER1_gen[13].FAI/Co1_SW2" LOC=SLICE_X66Y80;
INST "ADDER1_gen[13].FAI/Co1_SW2" BEL="A6LUT";
INST "ADDER1_gen[13].FAI/Co1_SW1" LOC=SLICE_X66Y80;
INST "ADDER1_gen[13].FAI/Co1_SW1" BEL="B6LUT";
INST "ADDER1_gen[13].FAI/Co1_SW7" LOC=SLICE_X67Y80;
INST "ADDER1_gen[13].FAI/Co1_SW7" BEL="B6LUT";
INST "A_reg_13" LOC=SLICE_X80Y91;
INST "A_reg_13" BEL="DFF";
INST "ADDER1_gen[13].FAI/Co1_SW5" LOC=SLICE_X81Y79;
INST "ADDER1_gen[13].FAI/Co1_SW5" BEL="B5LUT";
INST "ADDER1_gen[13].FAI/Co1_SW3" LOC=SLICE_X81Y79;
INST "ADDER1_gen[13].FAI/Co1_SW3" BEL="B6LUT";
INST "ADDER1_gen[13].FAI/Co1_SW6" LOC=SLICE_X81Y80;
INST "ADDER1_gen[13].FAI/Co1_SW6" BEL="B5LUT";
INST "ADDER1_gen[13].FAI/Co1_SW4" LOC=SLICE_X81Y80;
INST "ADDER1_gen[13].FAI/Co1_SW4" BEL="B6LUT";
INST "ADDER1_gen[13].FAI/Co1_SW0" LOC=SLICE_X81Y80;
INST "ADDER1_gen[13].FAI/Co1_SW0" BEL="C6LUT";

                              ......
                              ......
                              ......
```

Figure 4.30: Example of User Constraints File (.ucf) extracted running Xilinx FPGA Editor tool after design implementation.

Xilinx FPGAs are arranged in columns and rows of Configurable Logic Blocks (CLBs), in which several slices are available. Each slice is numbered and uniquely identified by a couple of coordinates (X-Y) [69], and includes several LUTs and FFs. As an example, in Figure 4.30, the A_reg_13 register is allocated in a FF inside the Slice X80Y91.

This placement information is used in the last implementation phase to force the synthesis and implementation tools to keep the circuit implementation constant, thus avoiding changes in the path delays. The physical constraint file is also used, along with the *FPGA model* file, to extract (i) the number of temperature sensors (*N_TEMP* in Figure 4.28), and (ii) placement constraints for both TDF and temperature sensors.

The *FPGA model* file contains information about the physical organization of Configurable Logic Blocks (CLBs) and slices inside a specific FPGA device. By merging the information of the FPGA slices organization and the one contained in the physical constraint file, it is possible to automatically search for free slices in which sensors can be placed. More specifically, to ensure good path delay monitoring capabilities, the *FF2* of a TDF sensor must be placed in a slice as near as possible to the original circuit FF (i.e., *FF1)* associated with the selected path.

Although temperature sensors should be placed as near as possible to those FFs, some considerations and optimizations can be done. In fact, the number of temperature sensors to insert in the circuit depends on relative distances between the TDF sensors inserted in the design, and on the target measure accuracy. However, as already mentioned in Section 4.4.1.1, SATTA does not need a highly accurate temperature measure. Thus, if two or more TDF sensors are physically located close to each other, they can be grouped together. A single temperature sensor can

be employed to measure their temperature value, thus reducing the total number of required temperature sensors. This leads to a significant decrease of both power consumption and hardware overhead, wasted by the ring oscillators. As in [26], the maximum zone that a temperature sensor is charged to monitor is 10x10 FPGA CLBs. This means that, if one or more critical paths are enclosed in a 10x10 CLBs square, a single temperature sensor can be associated to them.

Once the number of required sensors and their associated placement constraints are computed, the HDL description of the circuit is modified to include SATTA modules and sensors. The different thresholds exploited by the *Manager* to perform its activities are selected by resorting to the knowledge of the circuit functionality (see Section 4.4.3).
Finally, the modified circuit is re-synthesized, placed, and routed using the *Constraints file*, in order to obtain the final implementation of the TDF aware circuit. Obviously, the *Constraints file*, includes both sensors and original circuit implementation constraints.
In each step of Figure 4.28, tools used when dealing with Xilinx FPGAs are listed in brackets. It is worth to note that all steps enclosed by the dashed line can be automated using a *tcl* script.

4.4.3 Experimental results

In order to evaluate the effectiveness of the proposed methodology, in both highly and lowly pipelined FPGA-based digital systems, SATTA has been applied on several circuit designs: (i) a LEON3-based SoC, including the LEON3 processor [24] and several peripherals, such as memory controllers, UARTs, and debug units, (ii) an AES decoder core [47], and (iii) an AMBER a25 ARM-compatible processor [48].
The LEON3-based SoC has been implemented on a Xilinx Virtex 6 device (*XC6VLX240T-1FF1156*), while the other circuits on a Virtex 7 FPGA device (*XC7VX485T-2FFG1761C*). Table 4.2 shows synthesis results of the stand-alone original designs, obtained using Xilinx Vivado™ Design Suite v2013.3.

Table 4.2: Synthesis Results

	Look-Up Tables	*Registers*	*Crit. path delay*	*Max. frequency*
Leon3-based SoC	12,225	10,834	12.54 ns	79.74 MHz
AES	1,335	429	4.99 ns	200.40 MHz
AMBER	8,230	3,002	17.39 ns	57.50 MHz

Following the SATTA insertion methodology presented in Section 4.4.2, for each implemented circuit path delays and slack distributions have been extracted. Figure 4.31 shows the slack distribution for each circuit. The figure reports, for each circuit, the number of paths (y axis) with a delay comprised within a given range with respect to the most critical path that sets the clock period (x axis). One can notice that few paths are very close to the clock period (range of 2% or 5%). Looking to these distributions, the user can choose how many paths can be considered as

"critical", thus requiring the TDF sensor insertion. As an example, if a 5% range is considered, the designer assumes that, during the whole lifetime of the device, the maximum increase of delay, caused by both aging effects and temperature variations, on the unmonitored paths (i.e., paths with slack higher than 5%), never exceeds the delay of the most critical path of the circuit.

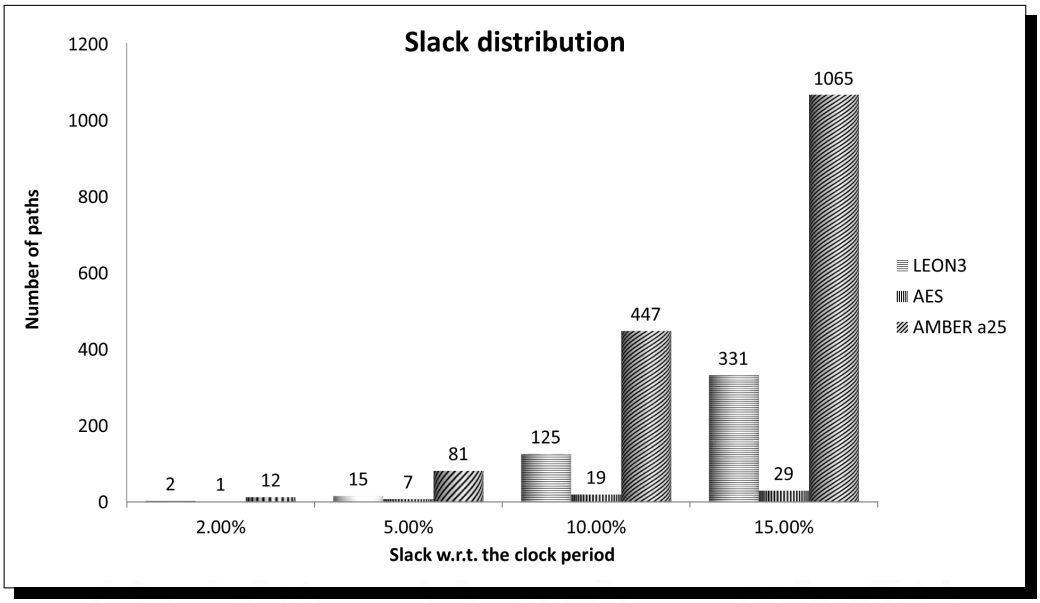


Figure 4.31: Paths slack distribution. The y axis represents how may paths have a delay included in a range of the nominal clock period identified on the x axis.

In the sequel, for the sake of brevity, experimental setup and simulation results will be given for the Leon3-based SoC design, only, while area overhead results, after applying the proposed methodology, will be provided for all the considered test cases.

By analyzing the slack distribution of the Leon3-based SoC design (Figure 4.31), assuming that the circuit will operate for one year, aging phenomena will cause an increment of paths delay of no more than 2% [55], and the device temperature variations will cause a maximum delay increment of 3% (e.g., from 25 °C to 75 °C following the model reported in [51]). All paths whose slack is less or equal than 5% of the clock period are therefore good candidates to be monitored and protected against TDFs. In this case study, the number of chosen paths is equal to 15, while all the other paths are not monitored since TDFs are not likely to appear in such paths.
Afterwards, the original design is modified and the 15 TDF sensors are automatically inserted, replacing the final FFs of the selected critical paths. Figure 4.32 shows that the optimal number of temperature sensors that should be inserted is equal to 3, since the 15 critical paths can be grouped into three proximity sub-sets (i.e., 10x10 CLBs square, as explained in Section 4.4.2). In Figure 4.32, white arrows show critical paths logical connections, from the starting point to the endpoint, while yellow circles represent the area in which each temperature sensor must be

placed. In this case study the maximum group dimension (i.e., maximum distance between endpoints in terms of FPGA CLBs) is equal to 9 CLBs.

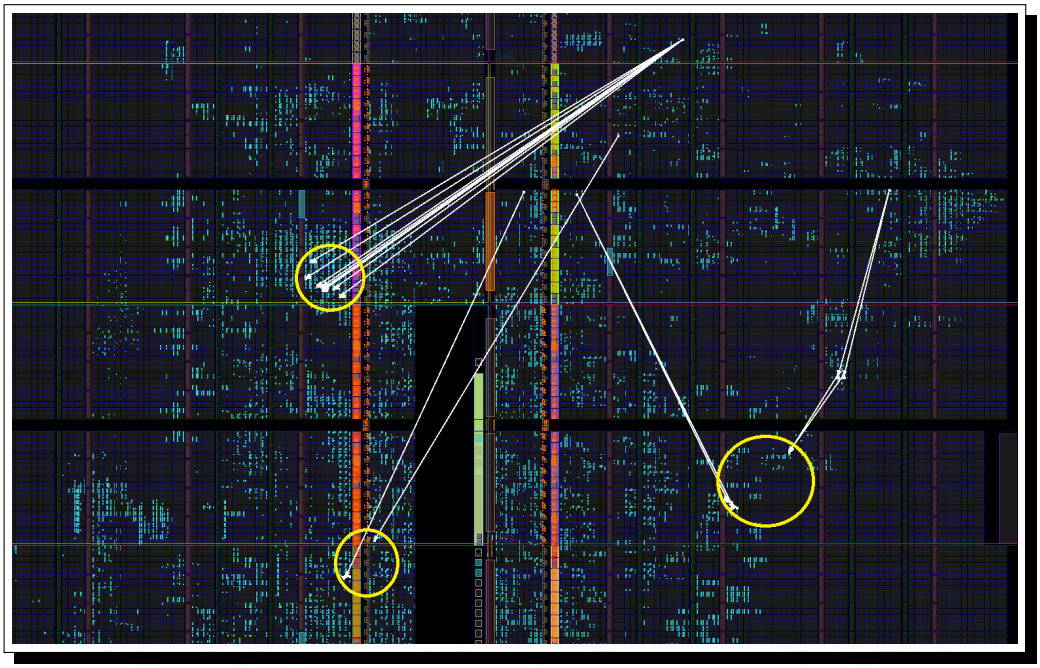


Figure 4.32: Leon3-based SoC critical paths and temperature sensors floorplan

The initial system's frequency (INIT_FREQ) has been set at the reference value provided by the synthesis tool considering worst-case temperature and voltage conditions (i.e., 79.74 MHz). The negative phase relationship between CLK1 and CLK2 (i.e., INIT_PHASE), has been chosen as the minimum possible (i.e., 1/256) of the initial system clock period (i.e., 311 ns), while CLK_DWFREQ has been set equal to 1/256 of the synthesis frequency (i.e., 0.3 MHz), minimizing the performance loss when scaling down frequency.

Afterwards, the modified system has been synthesized and implemented, including all sensors and SATTA modules. Table 4.3 shows the area occupancy of the single blocks involved in the SATTA methodology.

Table 4.3: SATTA synthesis results

	LUTs	*Registers*	*Block-RAMs*	*XADC*	*MMCM*	*BUFGMUX*
TDF sensor	2	2	-	-	-	-
CLK Generator	-	-	-	-	2	2
Temperature sensor	53	35	-	1	-	-
Manager	488	63	1	-	-	-

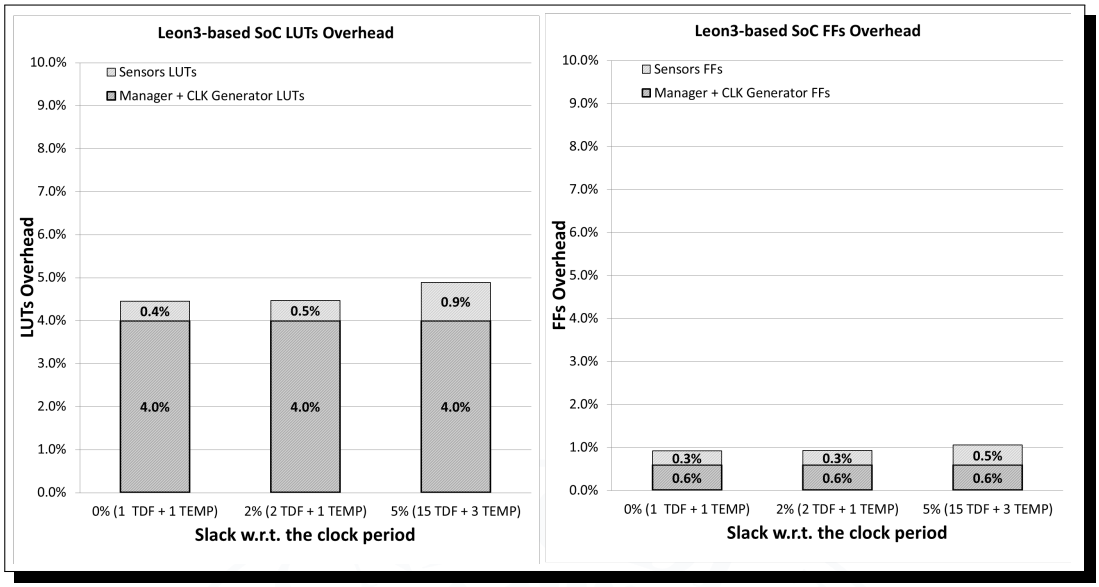


Figure 4.33: SATTA LUTs and FFs overhead when applied to the Leon3-based SoC case study

Figure 4.33 to Figure 4.35 show the resources overhead (in terms of LUTs and FFs) when applying SATTA to the five presented case studies, varying the selected slack threshold (i.e., 0%, 2%, and 5%). The x-axis also reports the number of required TDFs and temperature sensors. The number of TDF sensors is extracted by looking at Figure 4.31, while te number of temperature sensors is extracted looking to the physical layout of the implemented circuit (as done in the Leon3-based SoC case).

Looking at the results, one can notice that the area overhead introduced by the SATTA is in the range of 4-7% for large designs such as the Leon3-based SoC, while it becomes very large when applied to small modules such as the AES design. SATTA is therefore better suited for complex designs, like SoCs or processors.

The area overhead introduce by SATTA is due to two contributions highlighted with different textures in Figure 4.33 to Figure 4.35: (i) a constant part represented by the *Manager* and *CLK Generator* and (ii) a design-dependent part, which strongly depends on the number of sensors to be inserted. As depicted in Figure 4.33 to Figure 4.35, the variable hardware overhead can be reduced by selecting a lower threshold when looking to the paths slack distribution. It is worth noting that selecting a 0% slack threshold means that only the most critical path of the design is monitored.

Looking to the Leon3-based SoC case study, the insertion of SATTA modules causes an increase of resources usage of 4.9%, in terms of LUTs, and 1.2%, in terms of FFs, with a slack thresh-

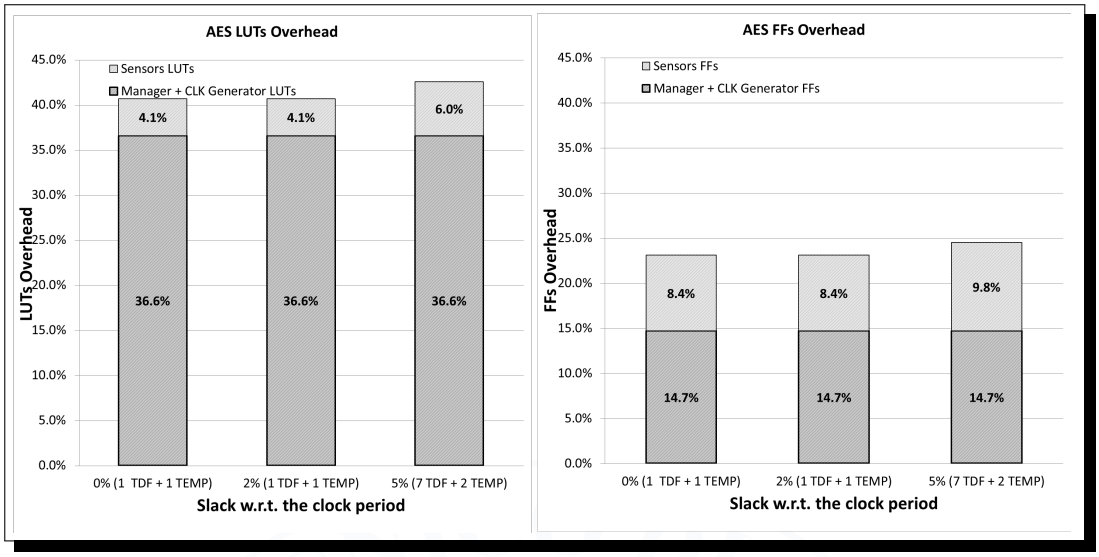


Figure 4.34: SATTA LUTs and FFs overhead when applied to the AES case study

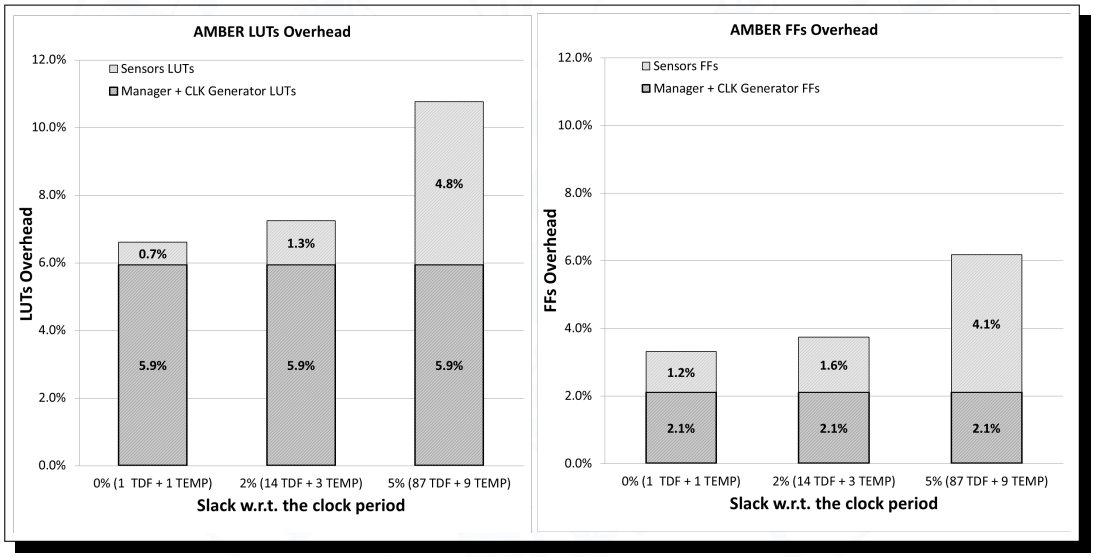


Figure 4.35: SATTA LUTs and FFs overhead when applied to the AMBER case study

old equal to 5%. A single FPGA internal Block-RAM memory is used to store the functions implemented by the *Manager* as look-up tables. An Analog-to-Digital Converter (i.e., *XADC* FPGA internal hard macro) is also required to measure the temperature from the internal reference diode. Nonetheless, just one input channel is employed, leaving available other channels to be used for other purposes. The CLK Generator employs 2 MMCMs and 2 clock multiplexers (i.e., *BUFGMUX*), but the actual overhead is of 1 MMCM, since one is already present in the original design in order to provide the system clock signal. A small timing overhead on the monitored path delay is present as well. This overhead is due to the insertion of the TDF sensor (composed of two FFs, instead of one FF present in the original design) in the path. In this case-study, the insertion of the TDF sensor leads to a 45 ps increase of the path delay, resulting in a minimal decrease of the maximum working frequency. However, this increase on the critical path delay is design and technology-dependent. Looking to the other case-studies, this increase attests in the range of 32-65 ps. Moreover, it is important to highlight that in all considered case studies, it was possible to insert the *FF2* of each TDF sensor into a CLB adjacent to the one containing the related *FF1* (as mentioned in Section 4.4.2).

Regarding power consumption, the initial unmodified design consumes 5.47 W. After the insertion of the SATTA modules, there is a power increase of 124 mW, mainly due to the additional *MMCM* hard macro and to the additional clock net, while the additional logic's power waste is negligible. This leads to a power overhead of 2.2% with respect to the unmodified design. It is worth noting that a power comparison with [56], which implemented a TDF sensor along with the clock generation network, is not possible since the used technologies are quite different. The Xilinx Virtex 6 and Virtex 7 FPGA devices used in this work are built on a 40 nm and 28 nm process technology, respectively, while the Xilinx Spartan-3AN device used in [56] is built on a 90 nm process technology.

To verify the proper working of the aging manager in the system, the post place and route design (generated by the synthesis tool) has been simulated using Modelsim® SE 10.0c. The effects of temperature and aging on path delays have been emulated inserting parametric delays in the post place and route HDL simulation model. Delays are automatically read from a text file at different times. These delays increase during the execution time, thus emulating the effects of temperature variations. The higher the temperature is, the greater is the delay, following the models reported in [50, 51], in which the path delay increases linearly of 5% when spanning the full operating temperature range (0-85 °C). Figure 4.36 plots the operating frequency profile when the temperature profile changes over time.

In Figure 4.36 the temperature increases linearly from 25 °C to 85 °C in 45 seconds. Then, after a constant period, it decreases linearly and, finally, it remains stable around 50 °C. At power-up, the working frequency quickly grows compared to its initial value (*Synthesis Frequency* in Figure 4.36) since the real temperature profile decreases the path delay, that was calculated in worst-case conditions during the synthesis process. This therefore improves the system's performance. When

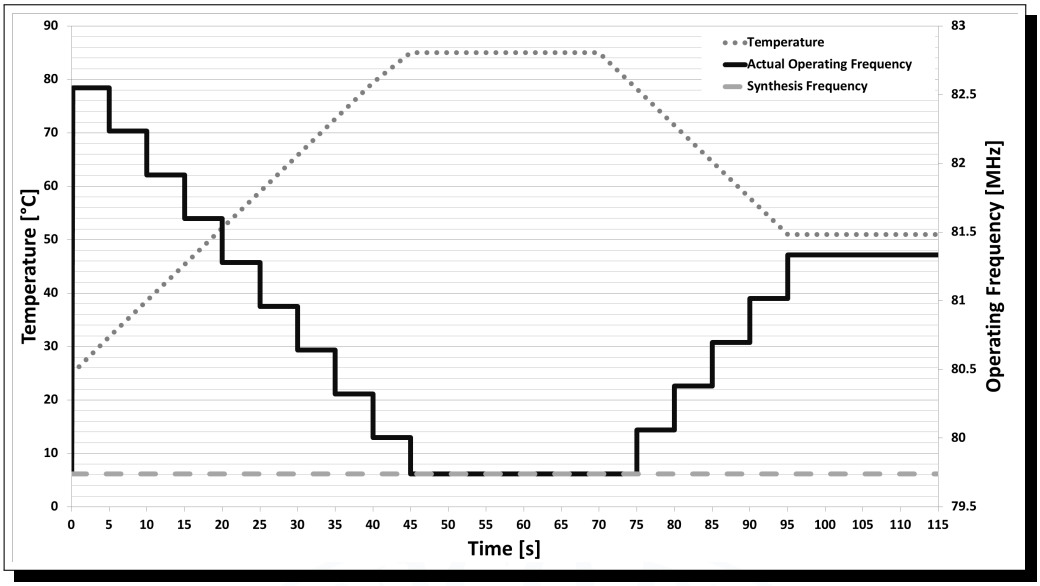


Figure 4.36: Simulated operating frequency trend with temperature variations

the temperature starts to increase, the working frequency is adapted by the *Manager*, decreasing its value by CLK_DWFREQ when a warning signal is raised. Obviously, when the temperature reaches the worst case condition (i.e., 85 °C), the operating frequency approaches the synthesis frequency. A soon as the worst case is reached, the working frequency remains stable since, even if no TDF warnings are detected, the *Manager* is not able to increase the frequency. This is due to the fact that the *Manager* recognizes a warning signal after a single CLK_PHASE increment (see Figure 4.26).

Whenever the temperature decreases (from 70 seconds in Figure 4.36), the working frequency is then adapted following the behaviour explained in Section 4.4.1.4.

E quindi uscimmo a riveder le stelle

Dante Alighieri

CHAPTER 5

CONCLUSIONS

The research activity carried out during the PhD program aimed at the development and the assessment of innovative methodologies to increase dependability of FPGA-based systems.
FPGAs flexibility and low non-recurrent engineering cost has risen the demand of such devices. This pushed FPGA vendors to adopt latest technology nodes, historically used mainly for high-end processors, to meet user needs. New issues in terms of devices dependability risen, as pointed out in Chapter 2.
During the initial phases of the research, reconfigurable devices have been deeply studied and analysed. As reported in Chapter 3, SRAM-based FPGAs have been chosen as most promising platform for future technology trends. Dynamic Partial Reconfiguration has been identified as a powerful instrument that, if properly exploited, could enhance system dependability.
The first outcome of the research has been the development of an innovative methodology for a dependable partial reconfiguration process, assuring a good level of dependability with marginal impact on performances. The proposed methodology featured low impact on reconfiguration time, and does not affect the static portion of the FPGA, even in presence of faulty bitstream files due to SEEs, as evaluated in Section 4.1.
An alternative and orthogonal methodology, presented in Section 4.2 enhances the dependability of the system by storing a compressed bitstream inside the device as backup. This partial bitstream guarantees the proper behaviour of the whole system by reconfiguring the partition with a blank module, that perhaps correctly routes the static connections through the reconfigurable partition, recovering from potential SEUs in the configuration memory.
Section 4.3 present a methodology to mitigate the NBTI effect in SRAM based FPGA. Exploiting consecutive DPRs 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 whole device lifetime.
The main contribution of the research activity has been the development of SATTA, a Self-Adaptive

Temperature-based TDF Awareness methodology to compensate for path delay variations due to both variation of temperature and aging effects in complex SoPCs, which features are explained in Section 4.4. The SATTA manager, supported by a reconfigurable and efficient clock manager module, is able at run-time to optimize the system frequency avoiding TDFs while optimizing the performance. Moreover, the manager optimizes the use of all SATTA sensors in order to minimize their self-aging. The application of SATTA on a set of real designs clearly proved the effectiveness of this approach especially when applied to large real designs that incur in very low overheads. Eventually, the full SATTA design pipeline has been easily automated by means of a set of TCL scripts. This represents an additional and valuable result for the easy exploitation of this methodology in real design flows.

LIST OF SYMBOLS AND ACRONYMS

Due to the large number of symbols used in this thesis to support the description of covered material, we provide the following list of symbols and abbreviations. This list is intended to help the reader identify the meaning of a given symbol or acronym in a fast and easy way.

ADC *Analog to Digital Converter*

ASET *Analog SET*

ASIC *Application Specific Integrated Circuit*

ALM *Arithmetic Logic Modules*

COTS *Commercial Off The Shelf*

CMOS *Complementary MOS*

CLB *Configurable Logic Block*

CPU *Central Processing Unit*

CRC *Cycle Redundancy Code*

DDR *Double Data Rate*

DFD *Design for Dependability*

DSET *Digital SET*

DPR *Dynamic Partial Reconfiguration*

DRAM *Dynamic RAM*

EM *Electro Migration*

EDA *Electronic Design Automation*

FIT *Failure In Time*

FET *Field Effect Transistor*

FF *Flip Flop*

FPGA *Field Programmable Gate Array*

HCE *Hot Carrier Effects*

IC *Integrated Circuit*

ICAP *Internal Configuration Access Port*

IP *Intellectual Property*

ITRS *International Technology Roadmap for Semiconductor*

JTAG *Joint Test Action Group*

LET *Linear Energy Transfer*

LUT *Look-Up Table*

MOS *Metal Oxide Semiconductor*

MAP *Microwave Anisotropy Probe*

MBU *Multiple Bit Upset*

NBTI *Negative Bias Temperature Instability*

PAR *Place and Route*

RAM *Random Access Memory*

RLE *Run Length Encoding*

SoC *System on Chip*

SoPC *System on Programmable Chip*

SOI *Silicon Over Insulator*

SBU *Single Bit Upset*

SEB *Single Event Burnout*

SEGR *Single Event Gate Rupture*

SEL *Single Event Latch-up*

SET *Single Event Transient*

SEEs *Single Event Upsets*

SEFI *Single Event function interrupts*

SER *Soft Error Rate*

SNM *Static Noise Margin*

SRAM *Static RAM*

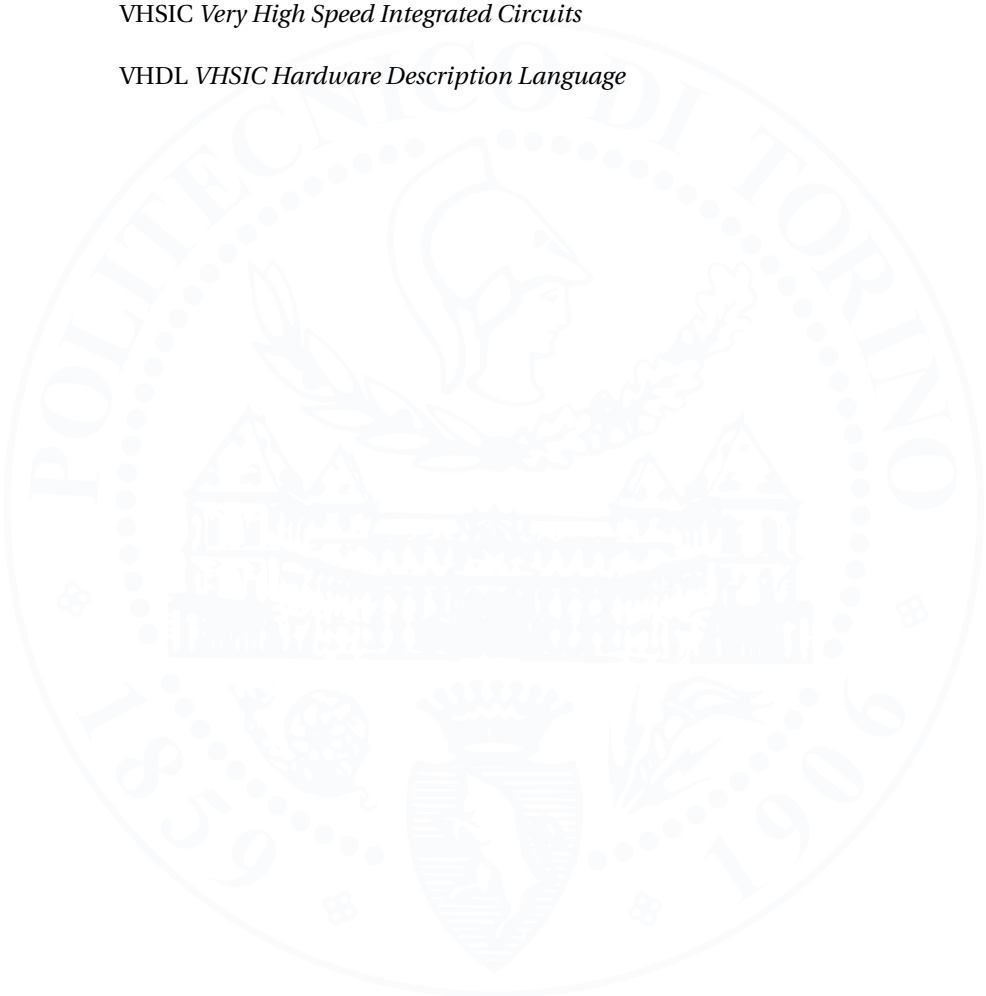
TDDB *Time Dependent Dielectric Breakdown*

TDF *Transition Delay Fault*

TTM *Time To Market*

VHSIC *Very High Speed Integrated Circuits*

VHDL *VHSIC Hardware Description Language*



POLITECNICO DI TORINO
1859
1906

BIBLIOGRAPHY

[1] Actel Corporation. *Effects of Neutrons on Programmable Logic Ű̈a white paper*, December 2002.

[2] Aeroflex Gaisler. *GRLIB IP Core User Manual*, 1.1.0 edition, January 2012.

[3] Altera Corporation. *Stratix V Device Overview*, sv51001 edition, 2012.

[4] Altera Corporation. *Cyclone V Device Overview*, cv-51001 (ver 2013.05.06) edition, May 2013.

[5] A. Amouri and M. Tahoori. High-level aging estimation for fpga-mapped designs. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 284–291, 2012.

[6] Abdulazim Amouri and Mehdi Tahoori. A low-cost sensor for aging and late transitions detection in modern FPGAs. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 329–335, September 2011.

[7] J. Autran, S. Semikh, D. Munteanu, S. Serre, G. Gasiot, and P. Roche. *Soft-Error Rate of Advanced SRAM Memories: Modeling and Monte Carlo Simulation*. Numerical Simulation - From Theory to Industry, 2012.

[8] R. Baumann. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In *Electron Devices Meeting, 2002. IEDM '02. International*, pages 329–332, Dec 2002.

[9] R. C. BAUMANN. Soft errors in commercial integrated circuits. *International Journal of High Speed Electronics and Systems*, 14(02):299–309, 2004.

[10] Hephaestus Books. *Articles on Coding Theory, Including: Huffman Coding, Run-Length Encoding, Bch Code, Hamming Code, Hamming Distance, Reed "Solomon Error Correction, Prefix Code, Binary Symmetric Channel, Unary Coding, Low-Density Parity-Check Code*. Hephaestus Books, 2011.

[11] Stephen Buchner and Dale McMorrow. Single event transients in linear integrated circuits. In *IEEE Nuclear and Space Radiation Effects Conferenc*, 2005.

[12] 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.

[13] Yuh-Jue Chuang and Ja-Ling Wu. An SGH-tree based efficient huffman decoding. In *Proc. of the Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing, and Fourth Pacific Rim Conference on Multimedia*, volume 3, pages 1483–1487, 2003.

[14] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. *Solid-State Circuits, IEEE Journal of*, 41(4):792–804, 2006.

[15] Shidhartha Das, Carlos Tokunaga, Sanjay Pant, Wei-Hsiang Ma, Sudherssen Kalaiselvan, Kevin Lai, David M. Bull, and David T. Blaauw. RazorII: In situ error detection and correction for PVT and SER tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):32–48, January 2009.

[16] S. Di Carlo, G. Gambardella, T. Huynh Bao, M. Indaco, P. Prinetto, D. Rolfo, and P. Trotta. Zipstream, improving dependability in dynamic partial reconfiguration. In *Design and Test Symposium (IDT), 2013 8th International*, pages 1–6, Dec 2013.

[17] S. Di Carlo, G. Gambardella, M. Indaco, P. Prinetto, D. Rolfo, and P. Trotta. Dependable dynamic partial reconfiguration with minimal area amp; time overheads on xilinx fpgas. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4, Sept 2013.

[18] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, and P. Trotta. Satta: a self-adaptive temperature-based tdf awareness methodology for dynamically reconfigurable fpgas. In *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 2015.

[19] Stefano Di Carlo, Salvatore Galfano, Giulio Gambardella, Daniele Rolfo, Paolo Prinetto, and Pascal Trotta. NBTI mitigation by dynamic partial reconfiguration. In *Electronics Conference (BEC), 2012 13th Biennial Baltic*, pages 93–96, October 2012.

[20] P.E. Dodd and L.W. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *Nuclear Science, IEEE Transactions on*, 50(3):583–602, June 2003.

[21] P.E. Dodd, M.R. Shaneyfelt, J.A. Felix, and J.R. Schwank. Production and propagation of single-event transients in high-speed digital logic ics. *Nuclear Science, IEEE Transactions on*, 51(6):3278–3284, Dec 2004.

[22] M. Fukazawa, M. Kurimoto, R. Akiyama, H. Takata, and Makoto Nagata. Experimental evaluation of digital-circuit susceptibility to voltage variation in dynamic frequency scaling. In *VLSI Circuits, 2008 IEEE Symposium on*, pages 150–151, 2008.

[23] H. Fuketa, M. Hashimoto, Y. Mitsuyama, and T. Onoye. Adaptive performance compensation with in-situ timing error predictive sensors for subthreshold circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(2):333–343, 2012.

[24] J. Gaisler. A portable and fault-tolerant microprocessor based on the SPARC v8 architecture. In *Dependable Systems and Networks (DSN), 2002. International Conference on*, pages 409–415, June 23–26, 2002.

[25] G Groeseneken, R. Degraeve, B. Kaczer, and P. Roussel. Challenges in reliability assessment of advanced cmos technologies. In *Physical and Failure Analysis fo Integrated Circuit, 2007. IPFA 2007, 14th International Symposium on the*, 2007.

[26] Markus Happe, Andreas Agne, and Christian Plessl. Measuring and predicting temperature distributions on FPGAs at run-time. In *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, pages 55–60, November 2011.

[27] R. Hashemian. Memory efficient and high-speed search huffman coding. *Transactions on Communications*, 43:2576–2581, 1995.

[28] S. Hauck and A. DeHon. *Reconfigurable Computing: The Theory and Practice of FPGA-based Computation*. Systems on Silicon Series. Morgan Kaufmann, 2008.

[29] D.F. Heidel, P.W. Marshall, J.A. Pellish, K.P. Rodbell, K.A. LaBel, J.R. Schwank, S.E. Rauch, M.C. Hakey, M.D. Berg, C.M. Castaneda, P.E. Dodd, M.R. Friendlich, A.D. Phan, C.M. Seidleck, M.R. Shaneyfelt, and M.A. Xapsos. Single-event upsets and multiple-bit upsets on a 45 nm soi sram. *Nuclear Science, IEEE Transactions on*, 56(6):3499–3504, Dec 2009.

[30] C.M. Hsieh, P.C. Murley, and R.R. O'Brien. Dynamics of charge collection from alpha-particle tracks in integrated circuits. In *Reliability Physics Symposium, 1981. 19th Annual*, pages 38–42, April 1981.

[31] V. Huard, M. Denais, and C. Parthasarathy. {NBTI} degradation: From physical mechanisms to modelling. *Microelectronics Reliability*, 46(1):1 – 23, 2006.

[32] International Electrotechnical Commission. *IEC 61508 Second Edition: Functional Safety of Electrical/Electronic/Programmable Electronic Systems*, 2010.

[33] International Organization for Standardization. *ISO 26262 First Edition: Road vehicles Ů Functional safety*, 2011.

[34] ITRS. *International Technology Roadmap for Semiconductors - Process Integration, Devices, and Structures (PIDS)*, 2013.

[35] Pradip Bose Jude A. Rivers Jayanth Srinivasan, Sarita V. Adve. The impact of technology scaling on lifetime reliability. In *Conference on Dependable Systems and Networks*, 2004.

[36] Tejas Jhaveri, Andrzej Strojwas, Larry Pileggi, and Vyachelav Rovner. Enabling technology scaling with "in production" lithography processes. volume 6924, pages 69240K–69240K–10, 2008.

[37] Kunhyuk Kang, M.A. Alam, and K. Roy. Characterization of nbti induced temporal performance degradation in nano-scale sram array using iddq. In *Test Conference, 2007. ITC 2007. IEEE International*, pages 1–10, Oct 2007.

[38] P. Koopman. 32-bit Cyclic Redundancy Codes for internet applications. In *Dependable Systems and Networks (DSN), 2002. International Conference on*, pages 459–468, December 2002.

[39] P. Koopman and T. Chakravarty. Cyclic Redundancy Code (CRC) polynomial selection for embedded networks. In *Dependable Systems and Networks (DNS), 2004. International Conference on*, pages 145–154, July 2004.

[40] Ian Kuon, Russel Tessier, and Jonathan Rose. Fpga architecture: Survey and challenges. *Foundamental Trends in Electronic Design Automation*, 2:135–253, 2008.

[41] Glen G. Langdon. An introduction to arithmetic coding. *IBM J. Res. Dev.*, 28:135–149, 1984.

[42] J.-C. Laprie. Dependable computing and fault tolerance : Concepts and terminology. In *Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on*, pages 2–, Jun 1995.

[43] Jie Li and J. Lach. Negative-skewed shadow registers for at-speed delay variation characterization. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*, pages 354–359, Oct 2007.

[44] C. Maxfield. *FPGAs: World Class Designs*. World Class Designs. Elsevier Science, 2009.

[45] Kaizad Mistry, C Allen, C Auth, B Beattie, D Bergstrom, M Bost, M Brazier, M Buehler, A Cappellani, R Chau, et al. A 45nm logic technology with high-k+ metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and 100% pb-free packaging. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 247–250. IEEE, 2007.

[46] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. Legall, editors. *MPEG Video Compression Standard*. Chapman & Hall, Ltd., London, UK, UK, 1996.

[47] OpenCores. Aes (rijndael), 2009.

[48] OpenCores. Amber arm-compatible core, 2013.

[49] Sang Phill Park, Kunhyuk Kang, and Kaushik Roy. Reliability implications of bias-temperature instability in digital ICs. *Design & Test of Computers, IEEE*, 26(6):8–17, November 2009.

[50] P. Pfeifer and Z. Pliva. On measurement of impact of the metallization and fpga design to the changes of slice parameters and generation of delay faults. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 743–746, 2012.

[51] Petr Pfeifer and Zdenek Pliva. On measurement of parameters of programmable microelectronic nanostructures under accelerating extreme conditions (xilinx 28nm xc7z020 zynq fpga). In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4, 2013.

[52] S.K. Saha. Modeling process variability in scaled cmos technology. *Design Test of Computers, IEEE*, 27(2):8–16, March 2010.

[53] T. Sato and Y. Kunitake. A simple flip-flop circuit for typical-case designs for dfm. In *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, pages 539–544, 2007.

[54] Minal Sawant. Single event effects complicate military avionics system design. *COTS Journal*, 2012.

[55] Suresh Srinivasan, Ramakrishnan Krishnan, Prasanth Mangalagiri, Yuan Xie, Vijaykrishnan Narayanan, Mary Jane Irwin, and Karthik Sarpatwari. Toward increasing FPGA lifetime. *Dependable and Secure Computing, IEEE Transactions on*, 5(2):115–127, April 2008.

[56] M.D. Valdes-Pena, J. Fernandez Freijedo, M.J.M. Rodriguez, J.J. Rodriguez-Andina, J. Semiao, I.M. Cacho Teixeira, J.P. Cacho Teixeira, and F. Vargas. Design and validation of configurable online aging sensors in nanometer-scale fpgas. *Nanotechnology, IEEE Transactions on*, 12(4):508–517, 2013.

[57] R. Vattikonda, Wenping Wang, and Yu Cao. Modeling and minimization of pmos nbti effect for robust nanometer design. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 1047–1052, 2006.

[58] Fan Wang and V.D. Agrawal. Single event upset: An embedded tutorial. In *VLSI Design, 2008. VLSID 2008. 21st International Conference on*, pages 429–434, Jan 2008.

[59] Wenping Wang, Shengqi Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Yu 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, Feb 2010.

[60] H. Watanabe, K. Matsuzawa, and S. Takagi. Scaling effects on gate leakage current. *Electron Devices, IEEE Transactions on*, 50(8):1779–1784, Aug 2003.

[61] F. Wrobel, J.-M. Palau, M.-C. Calvet, O. Bersillon, and H. Duarte. Simulation of nucleon-induced nuclear reactions in a simplified sram structure: scaling effects on seu and mbu cross sections. *Nuclear Science, IEEE Transactions on*, 48(6):1946–1952, Dec 2001.

[62] Xilinx. *MicroBlaze Processor Reference Guide*, 2004.

[63] Xilinx. *ML403 Evaluation Platform*, v2.5 edition, May 2006.

[64] Xilinx. *Zynq-7000 All Programmable SoC Overview*, ds190 (v12.3) edition, October 2014.

[65] Dagan White Xilinx. *Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors*, v1.0.1 edition, March 2012.

[66] Xilinx Corporation. *Virtex-4 FPGA User Guide*, December 2008.

[67] Xilinx Corporation. *PlanAhead User Guide*, ug632 (v11.4) edition, 2009.

[68] Xilinx Corporation. *PowerPC Processor Reference Guide*, January 2010.

[69] Xilinx Corporation. *7 Series FPGAs Configurable Logic Block*, ug474 (v1.4) edition, November 2012.

[70] Xilinx Corporation. *7 Series FPGAs Overview*, ds180 (v1.14) edition, 2012.

[71] Xilinx Corporation. *Partial Reconfiguration User Guide*, ug702 (v12.3) edition, October 2012.

[72] Xilinx Corporation. *7 Series FPGAs Clocking Resources*, ug472 (v1.8) edition, 2013.

[73] Xilinx Corporation. *7 Series FPGAs Configurable Logic Block*, ug474 (v1.5) edition, 2013.

[74] Kenneth M. Zick and John P. Hayes. On-line sensing for healthier fpga systems. In *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '10, pages 239–248, New York, NY, USA, 2010. ACM.

ACKNOWLEDGEMENTS

This thesis represents the culmination of three years of research. During this period many persons gave me suggestions and encouraged me. From all these persons I received a kind support, and for this reason I would like to thank all of them.

First of all, I wish to thank my advisor Prof. Paolo Prinetto, from Politecnico di Torino, that gave me the opportunity to start my research activity in his active and dynamic group. This work would not be possible without his precious suggestions. I wish to thank Prof. Stefano di Carlo, from Politecnico di Torino. His scientific driving guide helped me to find the right direction for my research activity. I wish to thank him also for gave me a real helpful hand during my whole research career.

It has been a pleasure, during the past years, to cooperate with the guys working in Lab.6. Daniele, Pascal, Salvatore and Alessandro, thanks to you for all the good time spent together. It has been an honour sharing with you this experience. I cannot forget to thank Davide and Diego, the revival lunches always inspired and motivated me so hard.

With the working weeks going on, Fridays evening have been always the major release of stress. That's why I have to thank my football team, U.S. Frazioni, for all the matches played together, both in the football field and outside.

This thesis is also the result of the experience I had during my time spent in ABB Corporate Research in Billingstad. In this sense I would like to thank two persons that have spent more than six months working with me and sharing their vision about scientific research: Frank Reichenbach and Trond Løkstad. With them and all the other persons of the Wireless and Embedded system group, I shared my experience, not only the work.

I would like also to thank people from the Technical University of Cluj-Napoca: many thanks, I passed beautiful moments with you.

A special thought is dedicated to Francesca, she has always been there for me, helping, sustaining, listening and caring about me. Many thanks my Love.

And last, but not least, I would like to really thank my mum Lidia, my dad Rosario, my brother Davide with Samantah and my grandma Iuccia, for they unconditional love and patience. Even if they can not understand all the parts of my work, they always share all my problems, giving an unlimited and invaluable support. They always encouraged me during this adventure.