ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μεταπτυχιακή Διατριβή

«TOWARDS EFFICIENT TMR SUPPORT FOR SRAM - BASED FPGAs»

ΚΥΡΙΑΚΟΥΛΑΚΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

Πνευματικάτος Διονύσιος, Αναπληρωτής Καθηγητής Π.Κ. (Επιβλέπων)

Δόλλας Απόστολος, Καθηγητής Π.Κ.

Παπαευσταθίου Ιωάννης, Επίκουρος Καθηγητής Π.Κ.

Χανιά – Μάιος 2009

ABSTRACT

Traditionally, TMR has been successfully applied in FPGAs to mitigate transient faults, which are likely to occur in harsh environments such as for space applications. In addition fault tolerance techniques in ICs gain importance as technology evolution and in particular the transistors' dimension shrinkage makes the modern circuits less reliable. However, TMR comes at high area penalty, which increases as the TMR grain becomes finer. The minimum area cost for coarse grain TMR is 3x, where x is the area of the simple design. This thesis proposes a novel SRAM-based FPGA architecture that is suitable for mapping designs when fault tolerance is desirable. We propose a slight modification to existing SRAM-based FPGA architectures that can support fine grain redundancy, at an area cost even less than 3x (1.8x in average for our benchmark circuits). Our approach also provides accurate fault location and allows smaller and more infrequent reconfigurations saving both reconfiguration time and power.

Contents

ΕΥΧΑΡΙΣΤΙΕΣ

Πρώτα απ'όλους θα ήθελα να ευχαριστήσω τον καθηγητή Δ. Πνευματικάτο, για την καθοδήγηση και τη βοήθειά του κατά τη διάρκεια υλοποίησης αυτής της μεταπτυχιακής εργασίας, αλλά και για την γενικότερη συνεργασία που είχαμε.

Στην συνέχεια, θέλω να ευχαριστήσω :

Τον κ. Μάρκο Κιμιωνή, μέλος ΕΕΔΙΠ και υπεύθυνος του εργαστηρίου Μικροεπεξεργαστών και Υλικού, για την καθοδήγηση και την αμέριστη συμπαράστασή του, καθόλη την παραμονή μου στο εργαστήριο.

Τον καθηγητή Δ. Σούντρη και τον Δρ. Κώστα Σιώζιο για την συνεργασία που είχαμε και την βοήθεια που μου προσέφεραν, για ένα σημαντικό κομμάτι της εργασίας.

Τους κ. Κυπριανό Παπαδημητρίου και Ευρυπίδη Σωτηριάδη, διδακτορικοί φοιτητές, για τη βοήθεια καθώς και τις συμβουλές τους σε διάφορα σημεία της εργασίας.

Τους κ. Παναγιώτη Χρήστου και Γιώργο Χατζηπαρασκευά, μεταπτυχιακοί φοιτητές, αλλά και όλους τους προπτυχιακούς και μεταπτυχιακούς φοιτητές του εργαστηρίου Μικροεπεξεργαστών και Υλικού για την βοήθεια και την στήριξη τους.

Τους φίλους και συναδέλφους μου, Γιώργο Επιτρόπου και Δημήτρη Μεϊντάνη, για την πολύ μεγάλη υποστήριξή τους κατά τη διάρκεια εκπόνησης της εργασίας.

Chapter 1

Introduction

Fault tolerance on semiconductor devices has been an important ever since upsets were first experienced in space applications. Space applications must consider the effect energetic particles (radiation) can have on electronic components. Single Event Effects (SEE) occur when charged particles hit the silicon, transferring enough energy in order to provoke a state change and consequently a fault in the system, with potentially serious consequences for the application, including loss of information and functional failure. This phenomenon is a concern in space, but also in other harsh operating environments.

When SEE have a transient effect we refer to their consequences as Single Event Upset (SEU). The main SEU are bit flips in the memory elements. For FPGAs in particular, SEUs may alter the logic-state of any static memory element (latch, flip flop, or RAM cell) or cause transient pulses in combinatorial logic paths. Since the user-programmed functionality of an FPGA depends on the data stored in millions of configuration latches within the device, an SEU in the configuration memory array might have adverse effects on the expected functionality of the user implemented design. Similarly, Single Event Transients (SETs) have a high probability for recognition at flip flop inputs where, if registered, causes a soft-error in the user data.

Static upsets in the configuration memory are not necessarily synonymous with a functional error; however, soft-errors are by definition functional errors, and upsets may or may not have an effect on functionality depending on whether the particular LUT is in use, the entry of this LUT is activated by the LUT inputs, the fault is masked by some other logical condition, etc. However, an accumulation of upsets in the configuration memory is eventually certain to lead to an error and then to functional failure.

In addition to space applications area, fault tolerance techniques increasingly gain importance as the evolution of the fabrication technology process leads to more sensitive and thus less reliable circuits. Charged particles that once posed a negligible threat have now higher probability to cause errors in current and future ICs.

Several SEU mitigation techniques have been proposed to avoid the effects of faults in digital circuits, including FPGAs. In particular, for SRAM-based FPGAs, SEU mitigation solutions can be classified in architectural and high-level solutions. Architectural level solutions are in fact suggestions of new architectural technologies and topologies such as using hardened memory cells for SRAM-based FPGAs, or using innovative routing structures. High-level techniques do not require changes in the FPGA's architectural structure, suggest user-level circuit design techniques that achieve SEU mitigation and can be easily applied by the user or CAD tools. High-level techniques are more attractive than architectural ones because they can be readily applied to all implementation technologies. They basis for high-level techniques is the use of redundancy, and the most well-known such technique is triple modular redundancy (TMR).

Redundancy can be either spatial or temporal. Temporal redundancy uses the same circuit to compute the same result multiple times. In the case of transient faults, a faulty computation will be followed by a correct one, and then the error will be discovered by the inequality of two results that are supposed to be identical. A subsequent re-computation will most likely give the correct result assuming that faults do not appear that frequently so as to affect the same circuit two times in a short time window.

Spatial redundancy uses multiple copies of the circuit to compute the multiple results in parallel. This is clearly better in terms of speed, but the cost is significantly increased. Typical approach for this type of redundancy is the Triple Modular Redundancy (TMR). In short, the cost of TMR is 3 times the circuit cost plus the cost of the comparator/voter module that checks the results and determines the correct value of the result.

The overall TMR cost is dependent on the *granularity* in which we apply the TMR method. One can apply TMR at the system level, taking the entire circuit, triplicating it, connecting it to the same inputs and adding a TMR-Voter on its outputs. However, using a large granularity leaves the system more vulnerable to multiple faults, whereas fine-grain TMR can tolerate multiple faults as long as they occur in different TMR subcircuits.

In this work we address the problem of faults in the LUT configuration memories cause by SEUs, usually dealt with using TMR and periodic scrubbing

(reprogramming of the entire device). We propose a new architectural-level solution aimed to reduce the cost of TMR. Our approach extends with minor changes the LUT and CLB structure of current FPGAs in order to reduce the cost of mapping fine grain TMR. We opt for fine granularity as this gives the best SEU mitigation results. Our approach exploits structures already available in commercial FPGAs, and with minor additions is able to reduce the space overhead of fine-grain TMR to about 1.76x for our benchmark circuits. Additionally, our proposed approach incurs minor performance overhead compared to a doubling of the latency for traditional fine-grain TMR. In this way, our proposed architecture addresses the main disadvantages of the fine-grain TMR. The proposed architecture comes at a small overhead that can enable the support of TMR a flexible mapping option instead of having to adopt a different radiation-hardened device.

Chapter 2

Faults and Fault Tolerance

First we shall analyze what the term "fault" means. It is difficult to find a unique definition for this word, because it has been used by many different fields of science. In our case we use the term for hardware circuits. So we give a general and a more specific definition.

fault = 1.an abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function 2.a defective point or region in a circuit or a device.

The fault has a physical significance. Its presence in a circuit gives another one that does not have the initial and desirable form or operation. The fault can have as consequence the creation of error.

The manifestation of the fault as an error is the discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition.

It should becomes understandable the difference between fault and error (fault and error). Let's study three cases.

Example 1:

We give to an AND gate as inputs 0 and 1, but the output's value is 1. This is an unexpected output because the theoretically correct value is 0, so there is an error. This error could be cause by anyone of the following faults:

(a).Initial wrong designing of the gate.

(b).A change of the values of the gate's transistor attributes. That change can be caused by various phenomena like increased temperature, pressure or deformity due to percussion.

In every case a fault produced a circuit with different layout than the desirable one. Even if we cannot discover easily this fault we observed its consequence: the error. The fault degraded the desirable operation of the circuit and caused a mistaken output.

Example 2:

Consider a 32-bits adder. The signal that notifies for overflow has for some reason "stuck" at zero. We have the existence of a fault. However as long as none of our calculations does not cause overflow, we will have never a wrong output. Only when we make an addition that causes overflow, an error will appear.

Example 3:

In an FPGA one specific lut (look up table) has been defected and does not operate properly. This is obviously a fault of the circuit. However it is possible this lut does not been used. If the circuit that is mapped doesn't need all the resources of the FPGA, this lut may be always an unused one. Thus the circuit will always work correctly.

In conclusion an error comes always from a fault, but a fault does not cause always an error. We would ideally like to have zero faults, but such a case is not possible in our physical world. If we can however eliminate the errors, despite the existence of faults, we come up with an acceptable functionality. After all, what really interests us is the correctness of our results. That is the meaning of fault tolerance, so:

Fault tolerance is the ability of a functional unit to continue to perform a required function in the presence of faults.

This tolerance of faults is referred to the methods that allow a system to by-pass in some way the faults and to avoid the errors. The system of our concern is an SRAM-based FPGA. Before we study the techniques that have been developed for fault tolerance in FPGAs, we will present first a categorization of errors in computer systems.

2.1 Fault-Tolerant Theory

2.1.1 Error detection vs. error correction

It is desirable, and in some cases vital, that data in a system remain correct when written into memory, stored, read from memory, communicated, or manipulated. The complexity of modern computers makes it impractical to depen d solely on reliable components and devices for reliable operation. Some redundancy is deeded for the detection and/or correction of errors which will invariably occur as information is being stored, transferred, or manipulated.

An important approach is the use of error detection and error correction. Error detection normally results in an interruption in the computation, followed by possibly a retry of some or all past computations, possibly a switching from use of a presumed faulty part of the computer to a presumed reliable part, or possibly some maintenance procedure either with or without performance penalty. On the other hand, error correction permits the computation process to continue uninterrupted. This would seem to favor error correction over error detection, but there are some mitigating factors. For a given amount of redundancy, if the number of error patterns which the decoder attempts to correct is increased, the probability of an undetected error increases; and the computational decoding complexity increases rapidly as the amount of error-correction capability is increased.

In communication networks, error detection combined with acknowledgment and retransmission protocols often provides a satisfactory method of obtaining extremely reliable communication in the presence of communication channel symbol errors.

2.1.2 Error Models

Most errors in computer systems are caused by faults, which are faulty or failed components of the system. In the absence of faults error s can occur due to random disturbances or noise. Such errors are rare except in communication over a long distance.

The types of error statistics which occur in memory, logic, and arithmetic circuits are many and varied. We can attempt to categorize them by reference to the following qualities depicted on scale with opposing properties on the two ends.

A 1. Symmetric: 0 to 1 and 1 to 0 errors are equally likely	2a. Asymmetric: a given word or operational unit has only 1 to 0 or only 0 to 1 errors. 2b. Unidirectional: a given word has only 1 to 0 or 0 to 1 errors but the decoder does not know a priori the type of error.
B 1. Independent bit errors	2. Physically clustered bit errors.
C 1. Transient faults or intermittently occurring bit errors	2. Permanent faults.

With respect to each quality, statistics could fall in some middle range of the scale rather than the extreme. For example, in A, the errors could be predominately, but not entirely, 1 to 0; in B, errors could be only lightly dependent, or perhaps clustered in a byte, while independent between bytes; in C, some bit faults could be of a semi permanent nature, with errors in that bit occurring more frequently than normal, but at random. In our work our concentration is given in A1, B1 and C (both 1 and 2) error categories, because these types of errors characterize SRAM -based FPGAs.

Chapter 3

Fault Tolerance in Integrated Circuits

This section is an introduction in fault tolerance for integrated circuits. Fault tolerance on semiconductor devices is an important issue since many years ago. Studies on radiations effects in integrated circuits prove that circuits are sensitive to charged particles that can harm their functionality. Upsets are classified in first, second and third order. These issues, as well as relevant research, are presented in the next sections. We use [1] as our main source for chapter 3 and 4.

3.1 The growing importance of fault tolerance

Fault tolerance on semiconductor devices has been an important issue since upsets were first experienced in space applications several years ago. Since then, the interest in studying fault-tolerant techniques in order to keep integrated circuits (ICs) operational in such hostile environment has increased, driven by all possible applications of radiation tolerant circuits, such as space missions, satellites, high-energy physics experiments and others [2].

Spacecraft systems include a large variety of analog and digital components that are potentially sensitive to radiation and must be protected or at least qualified for space operation. Designers for space applications currently use radiation-hardened devices to cope with radiation effects. However, there is a strong drive to utilize standard commercial-off-the-shelf (COTS) and military devices in spaceflight systems to minimize cost and development time as compared to radiation-hardened devices [3], [4].

The space radiation environment can have serious effects on spacecraft electronics. Single Event Effect (SEE) is the main concern in space [5], with potentially serious consequences for the application, including loss of information and functional failure. SEE occurs when charged particles hit the silicon transferring enough energy in order to provoke a fault in the system.

SEE can have a destructive or transient effect, according to the amount of energy deposited by the charged particles and the location of the strike in the device. The main consequences of the transient effect, also called Single Event Upset (SEU), are bit flips in the memory elements. SEU has been constantly magnified in the past years, caused by the continuous technology evolution that has led to more and more

complex architectures, with a large amount of embedded memories, followed by an amazing scaling down process of transistor dimensions following Moore's Law [6].

The fabrication technology process of semiconductor components is in continuous evolution in terms of transistor geometry shrinking, power supply, speed, and logic density [7]. As stated in [8], [4], [9] and [10], drastic device shrinking, power supply reduction, and increasing operating speeds significantly reduce the noise margins, and thus the reliability that very deep submicron (VDSM) ICs face from the various internal sources of noise. Reliability is the probability of no failure in a given operating period. It is used to measure how good a system is and how frequently it goes down.

The fabrication process is now approaching a point where it will be unfeasible to produce ICs that are free from upset effects. A more significant problem is related to SEU. It is predicted that neutrons produced by sun activity will dramatically affect the operation of future ICs. At the sea level, the energy of these particles is not strong enough to drastically affect the operation of current ICs. But as one approaches 0.1µm, or very low supply voltages, the rate of random errors induced by cosmic neutrons will be unacceptable. The situation is worse at flight altitudes. Alpha particles produced by packaging material are becoming another cause of increasing soft error rates in these technologies [11].

The necessity to protect integrated circuits against upsets has become more and more eminent [12], [13]. Experiments presented in [11], [14], [15] indicate that neutron particles present in the atmosphere are capable of producing SEE in avionics. Recent studies also show that memory cells composed of transistors with channel length smaller than 0.25 µm and combinational logic composed of transistors with length smaller than 0.13 µm may be subject to upsets while operating in the space environment, or inside the atmosphere [16], [17]. Terrestrial applications that are determined as critical such as bank servers, telecommunication servers and avionics are more and more considering the use of fault-tolerant techniques to ensure reliability.

Both discussed factors, the space market interest of using COTS/military devices in space applications and the constant increase in the radiation sensitivity of integrated circuits driven by the process scaling, have brought the necessity of researching fault-tolerant techniques for ICs able to cope with the radiation effects at

sea level, and also qualifying the design for space applications. Based on the definition of fault-tolerance, the goal is to maintain the IC operating correctly despite the existence of upsets. Although many techniques have been developed in the last few years attempting to avoid SEU, efficient fault-tolerant solutions are still a challenge for the next generation semiconductor industry, especially because of the complexity of the new architectures.

The development of fault-tolerant techniques is strongly associated with the target device, and it requires a detailed analysis of the effects of an upset on the related architecture. For each type of circuit, there is a set of most suitable solutions to be applied. In the past years, the integrated circuit industry has designed more and more complex architectures in order to improve performance, to increase logic density and to reduce cost. Examples of this development include Application Specific Integrated Circuits (ASICs), microprocessors composed of millions of transistors, high density Field Programmable Gate Array (FPGA) components and, more recently, System-on-a-Chip (SOC) composed of embedded microprocessors, memories and analog blocks. These architectures have made a dramatic impact on the way systems are designed, providing a large amount of information processing on a single chip. They cover a wide range of applications, from portable systems to dedicated embedded control units and computers. In particular, FPGAs have made a major improvement in systems design by adding the reconfigurability feature, which reduces the time to market and increases the design flexibility.

The first step of a fault-tolerant scheme is fault detection. Fault detection has two purposes: first to alert a supervising process that action needs to be taken for the system to remain operational and, second to identify which components of the device are defective so that a solution can be determined. These two functions may be addressed simultaneously or by a multi-stage process comprising of different strategies. The main approach for on-line fault detection methods uses redundancy (off-line approaches can reuse the test circuitry to detect permanent faults).

Time and hardware redundancy techniques are largely used in ASIC [18], [19], [20]. They range from concurrent error detection (CED) to correction mechanisms. The use of full time or full hardware redundancy permits voting the correct value in the presence of single upsets

3.2 Radiation Effects in Integrated Circuits

A single particle can hit either the combinational logic or the sequential logic in the silicon [21], [22]. Figure 3.1 illustrates a typical circuit topology found in nearly all sequential circuits. The data from the first latch is typically released to the combinatorial logic on a falling or rising clock edge, at which time logic operations are performed. The output of the combinatorial logic reaches the second latch sometime before the next falling or rising clock edge. At this clock edge, whatever data happens to be present at its input (and meeting the setup and hold times) is stored within the latch.

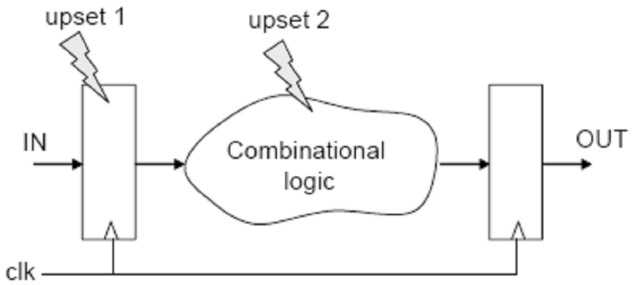


Figure 3.1: Upsets hitting combination and sequential logic

When a charged particle strikes one of the sensitive nodes of a memory cell, such as a drain in an off state transistor, it generates a transient current pulse that can turn on the gate of the opposite transistor. The effect can produce an inversion in the stored value, in other words, a bit flip in the memory cell. Memory cells have two stable states, one that represents a stored '0' and one that represents a stored '1'. In each state, two transistors are turned on and two are turned off (SEU target drains). A bit-flip in the memory element occurs when an energetic particle causes the state of the transistors in the circuit to reverse, as illustrated in figure 3.2. This effect is called Single Event Upset (SEU), and it is one the major concerns in digital circuits.

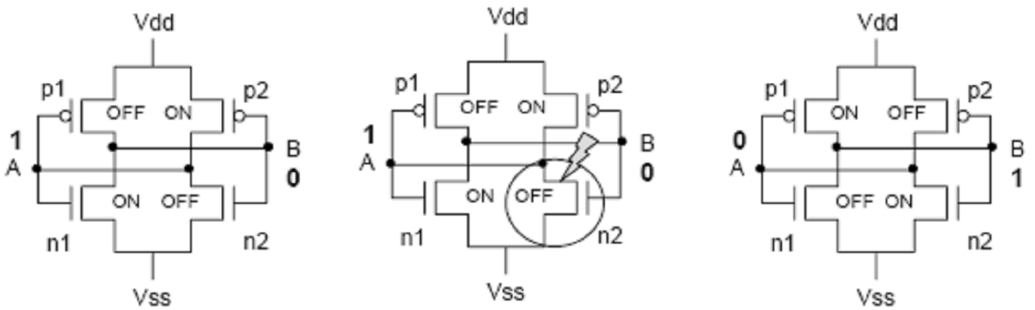


Figure 3.2: Single Event Upset (SEU) effect in a SRAM Memory cell

When a charged particle hits the combinational logic block, it also generates a transient current pulse. This phenomenon is called single transient effect (SET) [23]. If the logic is fast enough to propagate the induced transient pulse, then the SET will eventually appear at the input of the second latch in figure 3.1, where it may be interpreted as a valid signal. Whether or not the SET gets stored as real data depends on the temporal relationship between its arrival time and the falling or rising edge of the clock.

Figure 3.3 exemplifies the signal paths in a combinational logic. In [24], [25] the probability of a SET becoming a SEU is discussed. The analysis of SET is very complex in large circuits composed of many paths. Techniques such as timing analysis could be applied to analyze the probability of a SEU in the combinational logic being stored by a memory cell or resulting in an error in the design operation. Additional invalid transients pulses can occur at the combinatorial logic outputs as a result of SETs generated within global signal lines that control the function of the logic. An example of this would be SETs generated in the instruction lines to an ALU (Arithmetic Logic Unit). In [26], the widths of some induced transient pulses are measured to obtain more precise models for fault-tolerant analysis.

Please note that according to the logic fan-out, a single SET can produce multiple transient current pulses at the output. Consequently, SETs in the logic can also provoke multiple bit upsets (MBU) in the registers once the SETs are captured by the flip-flops.

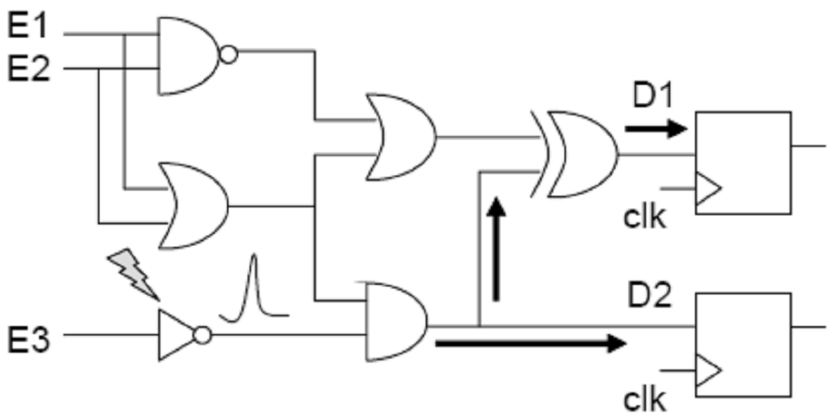


Figure 3.3: Single Event Transient (SET) Effect in Combinational Logic based on (ANGHEL et al., 2000)

Performing a more detailed analysis, the sensitive regions of an integrated circuit are the surroundings of the reverse-biased drain junctions of a transistor biased in the off state [27], as for instance the drain of the off p-channel transistor, see figure 3.4. As current flows through the struck transistor, the transistor in the on-state (n-channel transistor in figure 3.4) conducts a current that attempts to balance the current induced by the particle strike. Actually, there are three current components at the struck node. The current induced by the particle strike IP, the current ION that flows through the transistor in the on-state, and the current IC that charges the parasitic capacitances at the node. The current IC (t) is the current that will charge the node equivalent capacitance and cause the bit flip, and is given by:

$IC(t) = IP(t) - ION(t)$

If the current induced by the particle strike is high enough, the on-transistor cannot balance the current and a voltage change at the node will occur. This voltage change can be propagated to the opposite inverter and lead to the flipping of the bit stored in the memory cell. If the voltage transient is feedback through the opposite inverter a SEU occurs. If the voltage on the struck node is recovered by the current feed through the on-transistor no SEU will be observed.

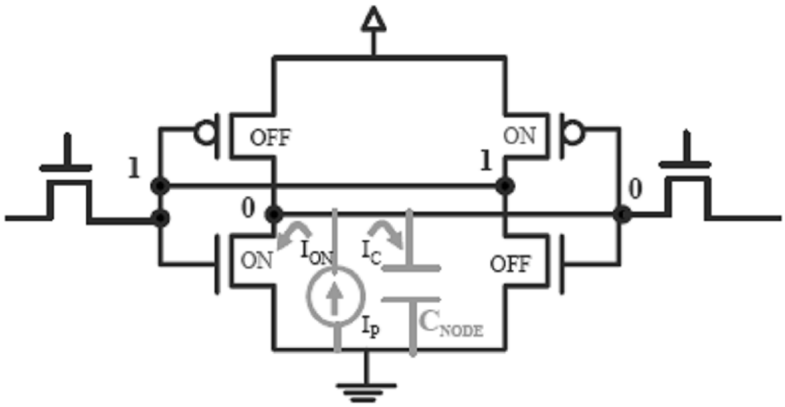


Figure 3.4: Single Event Upset effect analysis in a SRAM Memory cell

The critical charge has been reduced in new process technologies because of scaling. For constant field scaling, for example, as all physical device dimensions such as gate length L, gate width W, and gate oxide thickness TOX, are reduced, the supply voltage VDD and the threshold voltage VTH are also reduced proportionately. This fact results in proportionately lower drain current (ION), proportionately lower

load capacitance (C), and proportionately lower circuit gate delay (C*VDD/ION). This means that less charge or current is required to store information. Consequently, devices are becoming more vulnerable to radiation and this means that particles with small charge, which were once negligible, are now much more likely to produce upset.

3.3 SEU Classification

SEUs can be classified in first, second and third order effects, according to the number of upsets that occur at the same time in the circuit. A single bit upset (SEU) is classified as a first order effect, while multiple bit upsets (MBU) are classified as second or third order effects. MBU can occur when a single charged particle traveling through the IC at a shallow angle, nearly parallel the surface of the die, simultaneously strikes two sensitive junctions by direct ionization or nuclear recoil [28].

In [29], experiments in memories under proton and heavy ions fluxes have shown multiple upsets provoked by a single ion. MBUs were observed for all angles of incidence for LET greater than 25 MeV/(mg/cm2). There are three types of MBU. The first one occurs when a single particle hits two adjacent sensitive nodes, located in two distinct memory cells. This event is classified as a second-order effect. This type of MBU can be avoided by specific placement, for instance, memory cells of a same register or memory data can be placed far away from each other to same data structure.

The second type of MBU occurs when a single particle strikes two adjacent sensitive nodes located in the same memory cell. This event is classified as a third-order effect. The probability of such a multiple node strike can be minimized in a circuit design by taking care in the physical layout to separate critical node junctions by large distances, and by aligning such junctions so that the area of each junction, as viewed from the other, is minimized.

The third type of MBU occurs when multiple particle strike multiple sensitive nodes in the silicon provoking upsets in multiple memory cells. This event can be analyzed like a group of SEU and it will represent the same immunity characteristics. Based on [29], the majority of multiple upsets located in adjacent cells are provoked

by a single particle. There is a very low probability of more than one charged particle interacting in adjacent cells, provoking upsets in a period smaller than one second. This can be observed in [30], where it is shown some SEU flight results of two SRAM memories (Hitachi and MHS). A total of 691 upsets were detected for the analyzed period of time, 333 of them arising on the Hitachi SRAM and 358 occurring in the MHS SRAM memory. From this amount only few were multiple upsets, 8 double upsets in the Hitachi and 3 in the MHS memory. The distribution of bit flips within the memory word bits was uniform, and transitions 1 to 0 seem to be slightly more frequent than 0 to 1 for all the tested memories too.

Chapter 4

Fault Tolerance in SRAM-Based FPGAs

This chapter refers to fault tolerance techniques for FPGAs and especially for SRAM-based category. There are some effects that separate ASIC from FPGAs and must be taken of great concern when designing a successful fault tolerant technique. TMR is the most known and frequently used technique. In association with TMR, another technique called scrubbing can successfully mitigate SEUs in FPGAs. These and other techniques are discussed in the sections.

4.1 SRAM-based FPGAs and fault tolerant techniques

There are many types of customization in the FPGAs. One of the most popular ones uses SRAM memory cells to customize the FPGA, which makes possible in-the-field customization as many times as necessary in a very short period of time. Examples are the families Virtex-II, Virtex-4 and Virtex-5 fabricated by Xilinx. As a result, SRAM-based FPGAs are even more valuable for remote missions by offering the additional benefits of allowing in-orbit design changes, with the aim of reducing the mission cost by correcting errors or improving system performance after launch.

The advantages of using SRAM-based FPGAs for space applications and the increase of logic complexity of the programmable logic with more and more embedded memories and specific architectures such as microprocessors brings us the necessity of researching new SEU mitigation techniques specific for programmable architecture. This book presents the study and development of SEU mitigation techniques for programmable logic architectures, more specifically for SRAM-based FPGAs. The consideration of using FPGA for space applications is fairly recent, and there is a lot of work to be done in this area. Presently, there is no efficient solution for SRAM based FPGAs that can ensure 100% reliability in all conditions against SEU.

Several fault-tolerant techniques have been studied in the past years to protect ASICs against transient faults, and because FPGAs are composed of combinational and sequential logic, and more recently embedded processors, previous work dealing with standard integrated circuits can be adapted to the programmable logic architecture by finding the best tradeoff among area overhead, performance penalties,

single and multiple upset correction, process technology and implementation cost. However, the SEU mitigation techniques previously used for ASICs cannot simply be applied to programmable circuits because of the distinct effect of a SEU in the FPGA architecture compared to an ASIC, as will be further discussed in the next chapter. Consequently, the effect of SEUs in the SRAM-based FPGA architecture must be investigated to identify the limitations of the already used fault-tolerant techniques and to guide the investigation to new solutions.

There are two ways to implement fault-tolerant circuits in SRAM-based FPGAs, as exemplified in the flowchart in figure 3.2. The first possibility is to design a new FPGA matrix composed of fault-tolerant elements. These new elements can replace the old ones in the same architecture topology or a new architecture can be developed in order to improve robustness. The cost of these two approaches is high and it can differ according to the development time, number of engineers required to perform the task and the foundry technology used. Another possibility is to protect the high-level description by using some sort of redundancy, targeting the FPGA architecture. In this way, it is possible to use a commercial FPGA part to implement the design and the SEU mitigation technique is applied to the design description before the description is synthesized in the FPGA. The cost of this approach is inferior to the previous one because, in this case, the user is responsible for protecting his/her own design, and the solution does not require new chip development and fabrication. In this way, the user has the flexibility of choosing the fault-tolerant technique and consequently, the overheads in terms of area, performance and power dissipation.

In summary of figure 4.1, the four different implementations of a fault tolerant FPGA, respectively, A, B, C and D have different costs. Cost B is higher than cost A, which is much higher than cost C, which is also higher than cost D. All of them have their own space in the market, as each application requires different constraints. But since the semiconductor industry tends to emphasize time-to-market and low-cost production, the implementations C and D look more interesting. In this work, both architectural and the high-level methods are presented and discussed, but because of the high cost of the implementations A and B, only implementations C and D are designed and tested in details. Next following chapters present some works that have been developed in these four alternatives solutions to protect SRAM-based FPGAs against SEU.

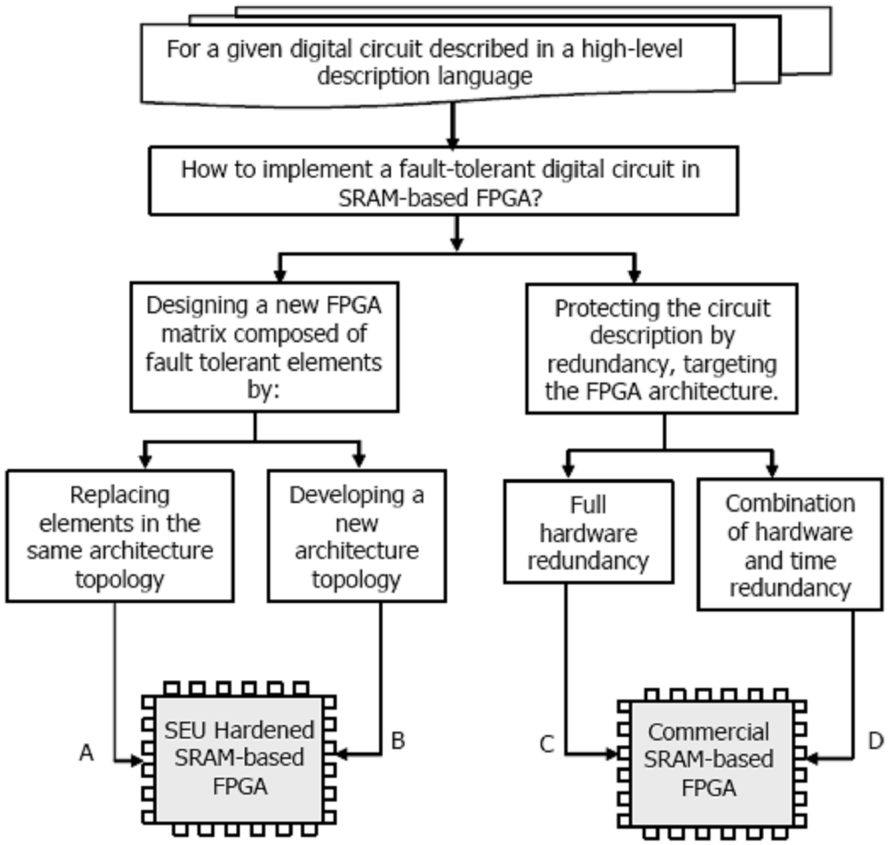


Figure 4.1: Design flow of how to protect a digital circuit implemented in a SRAM-based FPGA, where the cost of solution B is higher than th e cost of solution A, which is much higher than cost of solutions C and D

In the case of SRAM based FPGAs, the problem of finding an efficient technique in terms of area, performance and power is very challenging, because of the high complexity of the architecture. Redundancy is widely used as a method of fault detection in FPGAs. The most well-known high-level fault tolerance technique is the TMR. Many methodologies have been proposed in order to make any kind of circuit more reliable through the TMR [31], [32], [33], [34].

Redundancy provides a very fast means of error detection, as a fault is uncovered as soon as a discrepancy occurs. The identification resolution of the faulty component depends on the granularity of the technique; for example in TMR the fault can be pinned down to a particular functional block. In TMR, fault resolution increases as the granularity becomes finer.

Redundancy needs not be restricted to only spatial. It is also possible to detect errors with temporal redundancy, trading-off latency/data throughput for reliability. For example, operations are carried out twice by the same circuit, and the two results compared to detect discrepancies [35]. In the second operation, operands are encoded in such a way that they exercise the logic in a different way, and the output is then passed through a suitable decoder and compared to the original.

Although most of the work on redundancy has been aimed at detecting and correcting SEUs, there have been publications that apply the techniques to f ault detection. Dual Modular Redundancy (DMR) is used in [36] to grade the 'fitness' of competing configurations in an evolutionary approach. Parity checking is used in [37] as part of a fault tolerant scheme which is structured so that detection is applie d to small regular networks, rather than being bespoke to the function that is implemented. DMR is also presented as a suggestion in our architecture.

Redundant and data-checking detection systems are generally designed into an FPGA configuration, as they fit around the specific data and control functions that are implemented. In [38], a FPGA structure was considered which has built-in redundancy, so that it is transparent to the user who is designing the configuration.

4.2 Peculiar Effects in SRAM-Based FPGAs

The most common FPGA architecture consists of an array of configurable logic blocks (CLBs), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array. A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop, as shown below in figure 4.2. In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance.

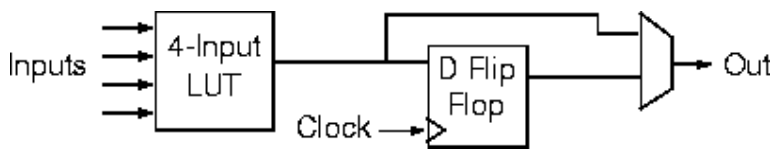


Figure 4.2: Typical logic block

There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. Since clock signals (and often other high fan-out signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed. Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block. Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it. Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it. Generally, the FPGA routing is unsegmented. That is, each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The figure below illustrates the connections in a switch box.

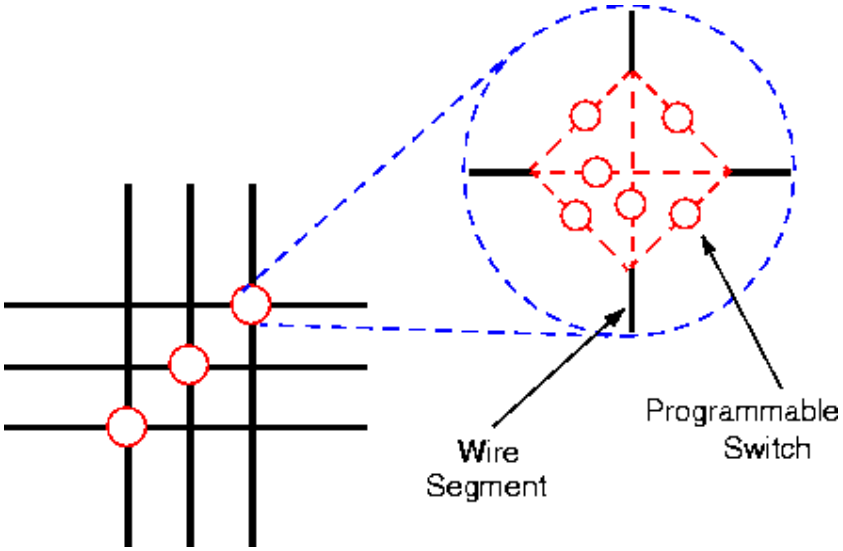


Figure 4.3: Switch box topology

SEU has a peculiar effect in FPGAs when a particle hits the user's combinational logic. In an ASIC, the effect of a particle hitting either the combinational or the sequential logic is transient; the only variation is the time duration of the fault. A fault in the combinational logic is a transient logic pulse in a node that can disappear according to the logic delay and topology. In other words, this means that a transient fault in the combinational logic may or may not be latched by a storage cell. Faults in the sequential logic manifest themselves as bit flips, which will remain in the storage cell until the next load.

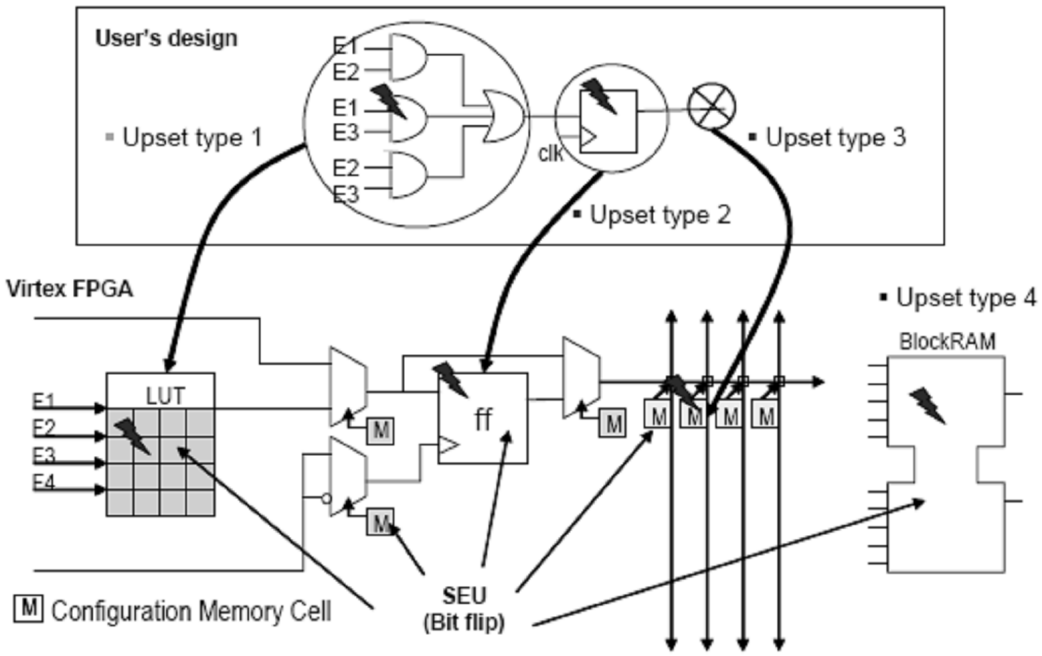


Figure 4.4: The comparison of the effects of a SEU in ASIC and FPGA architecture

On the other hand, in a SRAM-based FPGA, both the user's combinational and sequential logic are implemented by customizable logic memory cells, in other words, SRAM cells. In figure 4.4, the FPGA represented is a typical Virtex family architecture from Xilinx. When an upset occurs in the combinational logic synthesized in the FPGA, it corresponds to a bit flip in one of the LUTs cells or in the cells that control the routing. An upset in the LUT memory cell modifies the implemented combinational logic, see figure 4.5. It has a permanent effect and it can only be corrected at the next load of the configuration bitstream. The effect of this upset is related to a stuck-at fault (one or zero) in the combinational logic defined by that LUT (figure 4.4, upset type 1). This means that an upset in the combinational logic of a FPGA will be latched by a storage cell, unless some detection technique is used. An upset in the routing can connect or disconnect a wire in the matrix, see figure 4.6. It has also a permanent effect and its effect can be mapped to an open or a short circuit in the combinational logic implemented by the FPGA (figure 4.4, upset type 3). The fault can also be corrected at the next load of the configuration bitstream.

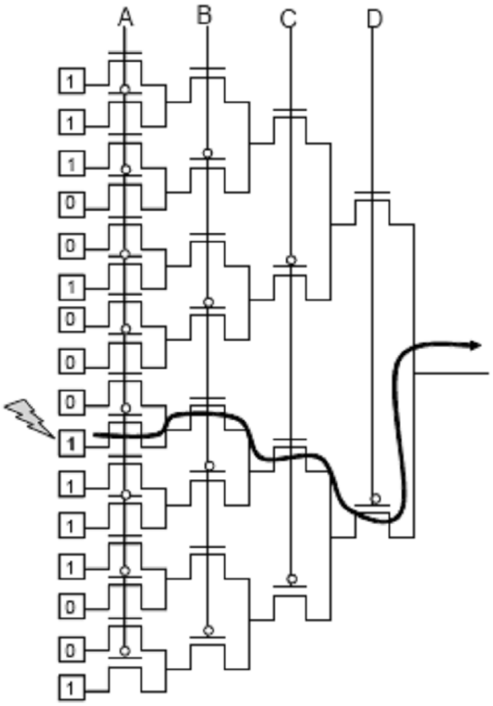


Figure 4.5: Upset in the LUT (logic change)

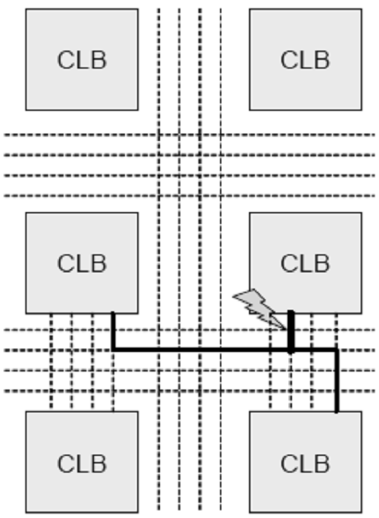


Figure 4.6: Upset in the routing (undesirable connection)

When an upset occurs in the user sequential logic synthesized in the FPGA, it has a transient effect, because an upset in the flip-flop of the CLB is corrected by the next load of the flip-flop (figure 4.4, upset type 2). An upset in the embedded memory (BRAM) has a permanent effect, and it must be corrected by fault tolerant techniques applied in the architectural or in the high-level description, as the load of the bitstream cannot change the memory state without interrupting the normal operation of the application (figure 4.4, upset type 4). In [39], [40], [41], the effects of upsets in the

FPGA architecture are also discussed. Note that there is also the possibility of having single event transient (SET) in the combinational logic used to build the CLB such as input and output multiplexors used to control part of the routing.

Radiation tests performed in Xilinx FPGAs [3], [42]-[46] show the effects of SEU in the design application, and prove the necessity of using fault-tolerant techniques for space applications. In [47] the effect of neutrons was also analyzed in a SRAM-based FPGA from Xilinx. In that time, the FPGA presented very low susceptibility to neutrons, but the vulnerability is increasing as the technology is reaching smaller transistor size and consequently higher logic density. Experiments with hundreds of latest generation FPGAs operating in tandem on the same board located at high altitude have shown one upset each 2 or 3 months due to neutrons. This number increases with the advance of technology.

A fault-tolerant system designed into SRAM-based FPGAs must be able to cope with the peculiarities mentioned in this section such as transient and permanent effects of a SEU in the combinational logic, short and open circuit in the design connections and bit flips in the flip-flops and memory cells.

4.3 Architectural-level vs. high-level mitigation techniques

In the case of SRAM-based FPGAs we can separate mitigation techniques into two categories: a) Architectural-level and b) High-level techniques. Architectural – level solutions are in fact suggestions of new topologies where structures constructed especially for SEU avoidance are used like: hardened memory cells and innovator routing structures.

Although these solutions can achieve a high reliability, they also present a high cost, because since they change the matrix, they need investment in development, test and fabrication. So far, there are very few FPGA companies that are investing in designing fault-tolerant FPGAs as this market is still focused in only military and space application, which is a very small market compared to the commercial one. However, because of the technology evolution, applications at the atmosphere and at ground level have been starting to face the effect of neutrons. As a result, fault-tolerant techniques begin to be necessary in many commercial applications that need some level of reliability.

A less expensive solution is a high-level SEU tolerant technique. that can be easily implemented by the user or by the company designers in commercial FPGAs or in parts manufactured by a technology that can avoid latch up and reduce the total ionization dose, as the Virtex QPRO family. The high-level SEU mitigation technique used nowadays to protect designs synthesized in the Virtex architecture is mostly based on TMR combined with.

4.4 Error correcting codes

A common way to protect memory structures, like SRAM blocks, is to use error correcting codes (ECC). An error-correcting code (ECC) or forward error correction (FEC) code is a code in which each data signal conforms to specific rules of construction such that departures from that construction in the received signal can be automatically detected and corrected. Some codes can correct a certain number of bit errors and only detect further numbers of bit errors. Codes which can correct one error are termed single error correcting (SEC), and those which detect two are termed double error detecting (DED). In general, these methods put redundant information into the data stream following certain algebraic or geometric relations so that the decoded stream can be corrected if damaged in transmission.

Examples of the ECC are Hamming code, BCH code, Reed-Solomon code, Reed-Muller code, Binary Golay code, and low-density parity-check codes. Hamming codes can correct single-bit errors and detect double-bit errors (SEC-DED) – more sophisticated codes can correct and detect more errors. An error-correcting code which corrects all errors of up to n bits correctly is also an error-detecting code which can detect at least all errors of up to 2n bits.

Computer memories that are sensitive to soft errors can use the Hamming code. More specialized codes are: single-error-correcting and byte-error-detecting codes or Byte-error-correcting codes. But these codes need more redundant bits than traditional Hamming. The first category needs as many bits as the original information, while the second that gives the best correction need check bits to be twice the original bits.

In case of codes with small overhead, like Hamming, one limitation of ECC arises in the case of two bit errors within a word, where the error can only be detected but not corrected. Another penalty is that every ECC needs special circuits like

syndrome calculator, decoding circuit and correcting circuit. These extra circuits cost area. Finally, even ECCs work well with memories they cannot efficiently implemented on combinational circuits, like adders.

4.5 TMR Description

The basic concept of triple redundancy is that a sensitive circuit can be hardened to SEUs by implementing three copies of the same circuit and performing a bit-wise "majority vote" on the output of the triplicate circuit (Figure 4.7). TMR works under the assumption that at most a single fault will be present in the replicated functions, and hence at most one of the voter's inputs can be incorrect. The cost of TMR is twofold: (i) area cost is increased due to the triplication of the functions, plus the voter cost, and (ii) the latency of the circuit is increased by the introduction of the voter in the circuit's critical paths.

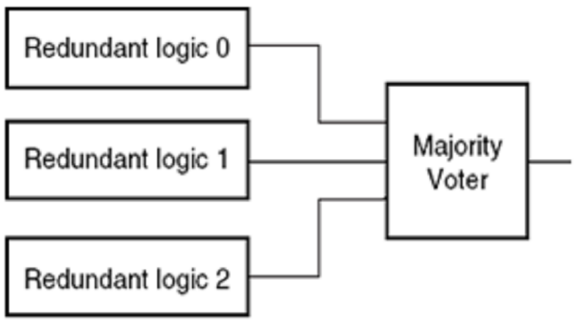


Figure 4.7: Upset Basic Triple Modular Redundancy (TMR).

The circuit in question can be a mere flip flop or an entire logic design. The function of the majority voter is to output the logic value ("1" or "0") that corresponds to at least two of its inputs. For example, if two or more of the voter's three inputs are a "1," then the output of the voter is a "1." If the inputs of the voter are labeled A, B, and C, and the output V, respectively, then the boolean equation for the voter is: V= AB + AC + BC. The Truth-Table is shown in Table 1.

A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 1: Majority Vote Truth-Table

Observing the truth table of the majority voter we can easily draw the corresponding circuit. Three different implement ations are presented in the next three figures. First and second use gates “and” and “or”, while the third 3-state buffers.

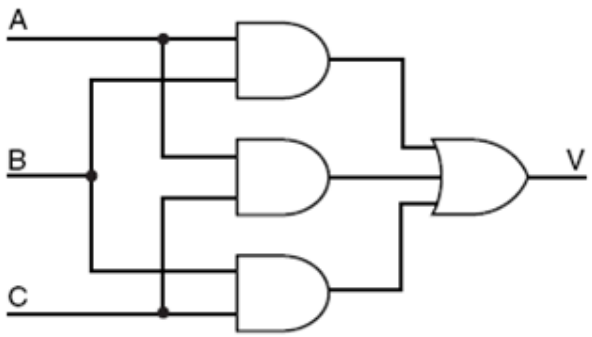


Figure 4.8: Majority Voter Circuit.

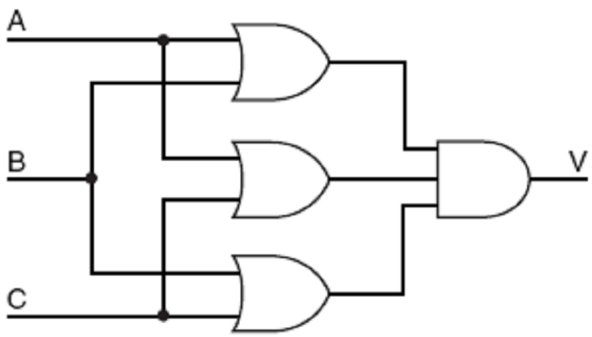


Figure 4.9: Majority Voter Circuit alternative.

For designs constrained by available logic resources, the majority voters can be implemented using the Virtex internal 3-state buffers instead of Look-Up Tables

(LUTs), which are used to implement all boolean functions in the user's design. The BUFT library primitive functions as an active low enabled 3 -state buffer.

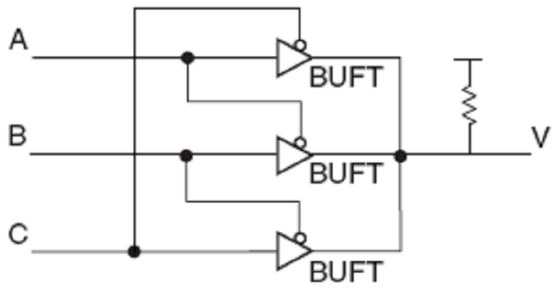


Figure 4.10: BUFT Style Majority Vote Circuit

4.5.1 TMR analysis specifically for Virtex FPGAs

The correct implementation of TMR circuitry within the Virtex architecture depends on the type of data structure to be mitigated. These data structures can be grouped into four different types:

1. Throughput logic
2. State-machine logic
3. I/O logic
4. Special features (block RAM, DLLs etc.)

1) Throughput logic

Throughput logic is a logic module of any size or functionality, synchronous or asynchronous, where all the logic paths within travel from the inputs to the outputs of the module without ever forming a logic loop. In other words, the logic states within a throughput logic structure are never dependent on their previous states. For example, an ADDER, of any size, is a throughput logic structure. Regardless of how many clock stages may, or may not, lie between the inputs and outputs of the adder, the output is always a function of the inputs only. An accumulator, however, is not a throughput logic structure because the output is fed back into the inputs of the embedded adder. An accumulator is an example of a state -machine logic structure.

2) State-machine logic

State-machine logic is any structure where a registered output (at any register stage within the module) is fed back into any prior stage within the module forming a registered logic loop. This structure is used in accumulators, counters, or any custom state-machines or state-sequencers, where the given state of the internal registers is

dependent on their own previous state. In this case it is essential, except triplicate the circuit, to put also a voter at the exit of each copy. Sate register should not stuck at an erroneous price, the exit of each voter is sent back, as an entry, to correct any possible fault. This loop appears in the following figure:

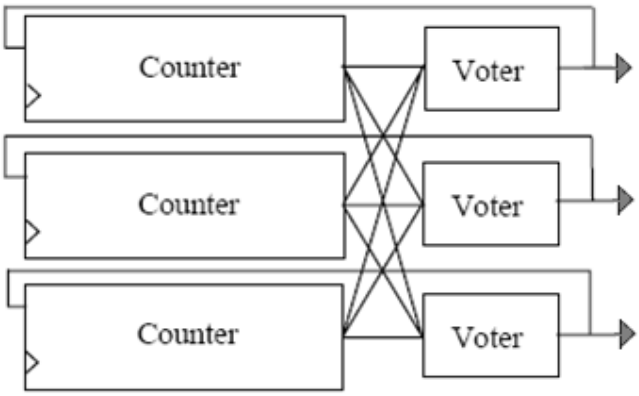


Figure 4.11: TMR counter schematic view

3) I/O logic

This logic refers to the inputs and the outputs of the FPGA design. We separately study inputs from outputs because the current techniques do not incorporate the use of bidirectional or differential Input Output Block (IOB) circuits.

The primary purpose for using a TMR design methodology is to remove all single points of failure from the design. This begins with the FPGA inputs. If a single input was connected to all three redundant logic legs within the FPGA, then a failure at that input would cause these errors to propagate through all the redundancies, and thus the error would not be mitigated. Therefore, each redundant leg of the design that uses FPGA inputs should have its own set of inputs (Figure 4.12). Thus, if one input suffers a failure, it will only affect one redundancy.

The outputs are the key to the overall TMR strategy. Since the full triple module redundancy generates every logic path in triplicate, there must ultimately be a method for bringing these triple logic paths back to a single path that does not create a single point of failure. This can be accomplished with TMR outputs.

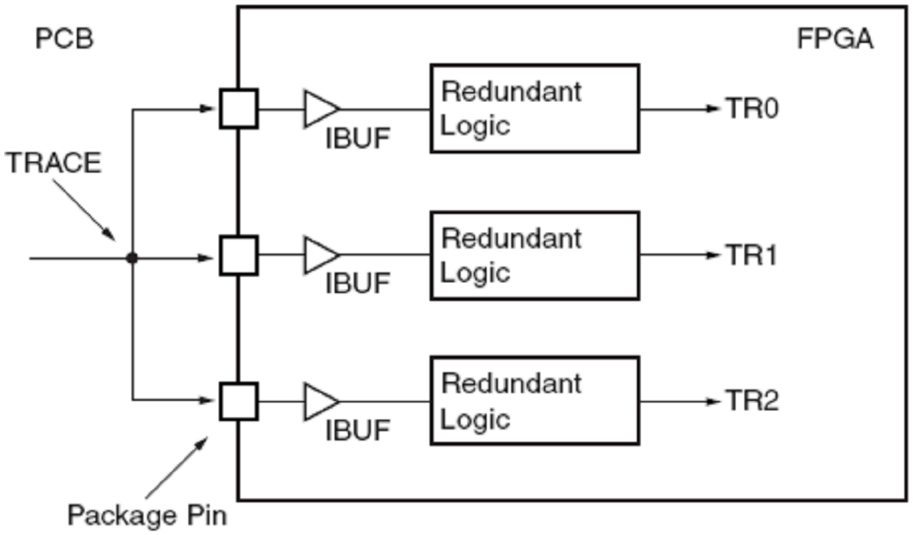


Figure 4.12: Triple Redundant FPGA Inputs

A TMR output is constructed using the OBUFT library primitives as shown in Figure 4.13. Each redundant logic path exiting the FPGA on an output does so through an OBUFT. The "enable" (T pin) of each OBUFT is controlled by a "minority voter" circuit. The minority voter indicates whether the path in question (primary path) agrees with either of the two redundant paths. If the primary path agrees with at least one of the redundant paths, then the primary path is considered to be part of the majority. If the primary path disagrees with both redundant pat hs, then the primary path is in the minority.

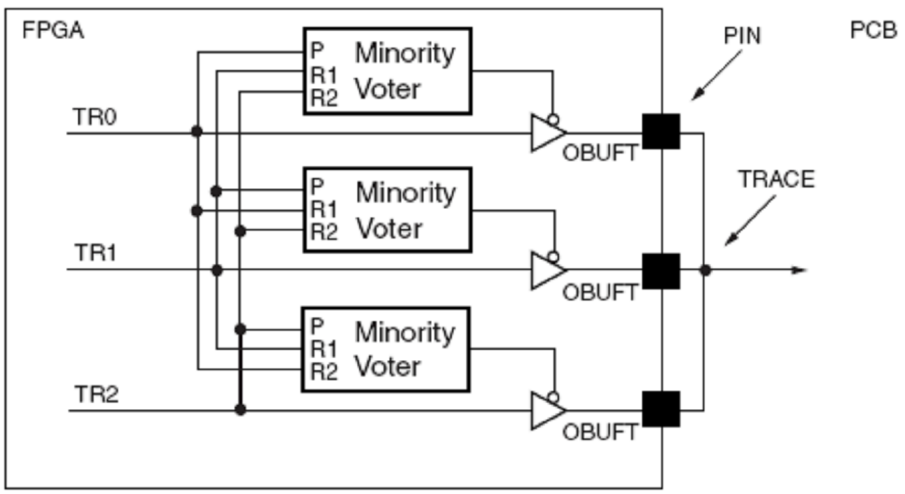
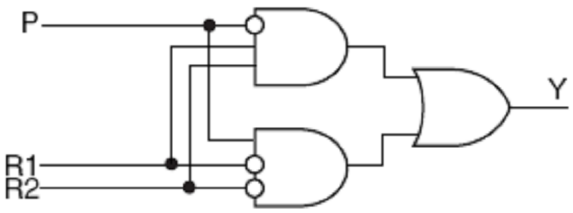


Figure 4.13: Minority Voted TMR FPGA Outputs

The minority voter is shown in Figure 4.14. If the primary path is part of the majority, then the minority voter will enable the corresponding (active Low) OBUFT allowing the data on its primary path to be driven out through the OBUFT and onto the Pad-Pin. If the primary path is not a part of the majority, then the OBUFT is disabled placing its output in a high-impedance state allowing the redundant outputs to drive the correct data.



P	R1	R2	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 4.14: Minority Voter Circuit

4) Block RAM

Architectural Virtex provides a number of special sub systems such as block RAM (BRAM), DLL and other, which need special concern when implementing TMR. A reliable method for BRAM TMR is to constantly refresh the block RAM contents. Since these are dual port memories, one of the ports can be dedicated to error detection and correction. But this also means that the block RAMs can only be used as single port memories by the rest of the user logic. To refresh the memory contents, a counter can be used to cycle through the memory addresses incrementing the address once every four clock cycles. In the following example (figure 4.15), the data width of port B is set to its maximum va lue of 16. This reduces the address width, and thus the counter size as well, to its minimum value of 8. However, the data width of Port A can be set independently of Port B and used in the application design.

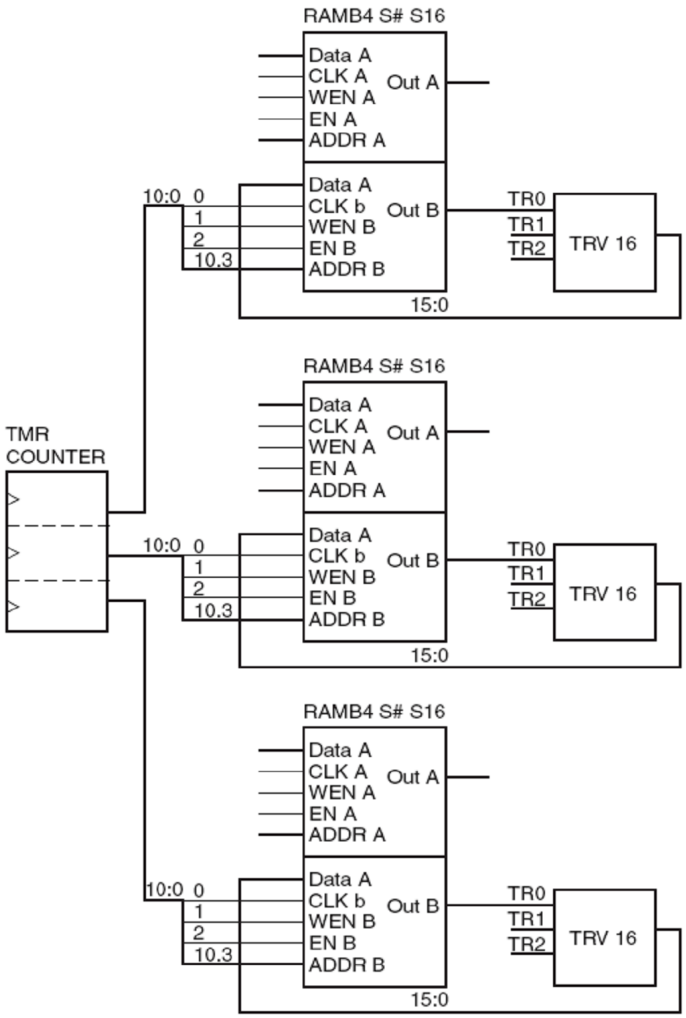


Figure 4.15: Minority Voter Circuit

4.5.2 TMR Granularity

The TMR technique can be applied at different granularity levels. Fine grain means that the application of the TMR method in small-sized modules. This option can tolerate more faults. Assuming that faults appear in random locations, when we consider smaller logic modules, the probability of two faults occurring within the 3 copies of the single function is reduced. Coarse grain TMR on the other hand is used to minimize the area and timing penalty: applying TMR in larger modules we minimize the voter area and timing penalty, at the expense of reduced tolerance to multiple faults. Figure 4.16 shows three different solutions of the application of the TMR at different levels of granularity:

a) The three replicas are grouped on the same reconfigurable portion, the voter occupies the adjacent frame and based on the detection one of the two portions is reconfigured;

b) The three replicas are placed on different frames and the corrupted one is reconfigured;

c) Each frame hosts three replicas and their voter and the corrupted stage is reconfigured.

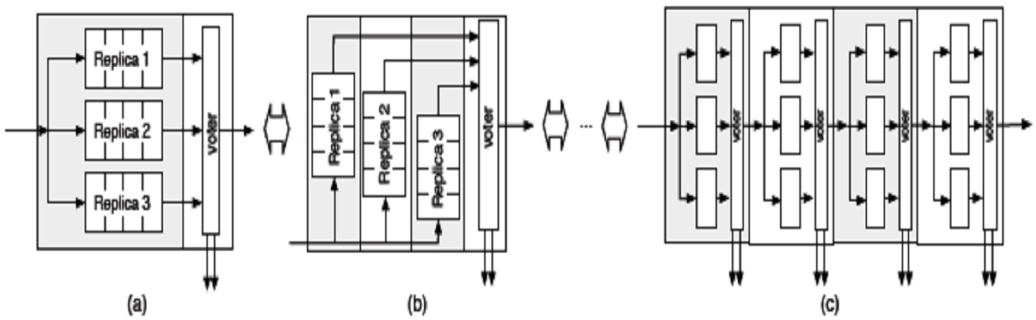


Figure 4.16: TMR applied with different levels of granularity (separately reconfigurable adjacent frames have different background color).

4.6 Scrubbing

The use of TMR in the design is not sufficient to ensure reliability for a long period of time, as upsets can accumulate in the matrix, provoking an error in the TMR. Note, as explained in previous section, that the upsets located at LUTs and in the routing configuration cells will not be removed until the next configuration of the device. Consequently, it is necessary to clean up all the upsets in such a frequency as to guarantee the correct functionality of the TMR methodology. The first technique proposed to clean the upsets inside the matrix was based on read-back of the bitstream, detecting an upset and problem of this technique is that it is too time consuming.

A simpler method of SEU correction is to omit read-back and detection of SEUs and simply reload the entire CLB Frame segment at a chosen interval (Xilinx, 2000c). This is called “scrubbing”. Scrubbing requires substantially fewer overheads in the system, but does mean that the configuration logic is likely to be in “write mode” for a greater percentage of time. However, the cycle time for a complete scrub can be made relatively short. The scrubbing allows a system to repair SEUs in the configuration memory without disrupting its operations.

In more detail: Figure 4.17 presents the configuration memory array that is divided into three separate segments: The "CLB Frames", "BRAM0 Frames" and "BRAM1 Frames. The two BRAM segments contain only the RAM content cells for the Block RAM elements. The BRAM segments are addressed separately from the CLB Array. Therefore, accessing the Block RAM content data requires a separate read or write operation. Read/Write operations to the BRAM segments should be avoided during post-configuration operations, as this may disrupt user operation.

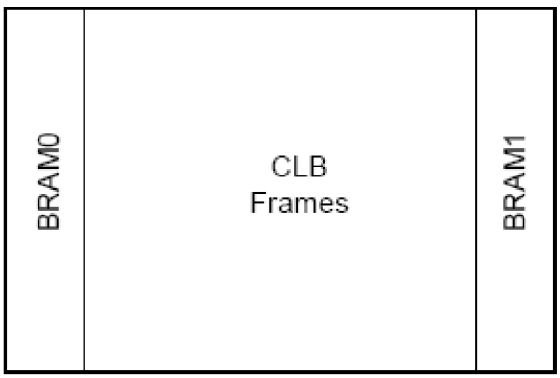


Figure 4.17: Virtex Frame Segments

The CLB Frames contain all configuration da ta for all programmable elements within the FPGA. This includes all Lookup Table (LUT) values, CLB, IOB, and BRAM control elements, and all interconnect control. Therefore, every programmable element within the FPGA can be addressed with a single read or w rite operation. All these configuration latches can be accessed without any disruption to the functioning user design, as long as LUTs are not used as distributed RAM components.

While CLB flip-flops do have programmable features that are selected by configuration latches, the flip-flop registers themselves are separate from configuration latches and cannot be accessed through configuration. Therefore, read-back and partial configuration will not affect the data stored in these registers.

The scrubbing cycle time depends on the configuration clock frequency and on the read-back bitstream size. The scrubbing rate describes how often a scrubbing cycle must occur. It is determined by the expected upset rate of the device for the given application. Upset rates are calculated from the static bit cross-section of the device and the charged particle flux the application or mission is expected to endure. The scrubbing rate should be set such that any SEU on the configuration memory will be fixed before the next upset will occur. In reality the scrubbing rate is minimized to be equal to the scrubbing cycle. In this way, configuration logic is always being refreshed. The implemented design can also have influence in the selection of the scrubbing rate. A good “rule of thumb” is to place the scrubbing rate one order of magnitude or more above the expected upset rate. In other words, the system should scrub, on the average, at least ten times between upsets.

Chapter 5

Proposed Architecture

This chapter describes our LUT structure that has the ability to support fault-tolerance. Two different implementations are presented, one for DMR and one for TMR. First, we present the structure of Virtex-5 LUT as it is our primary model where we made some changes in our architecture. Finally we present advantages and limitations of our implementation.

5.1 Considerations on TMR implementation

TMR implementation analysis, as a high-level fault tolerance technique on FPGAs led us to make some remarks and conclusions.

- Although in ASIC the topology of the circuit is similar to figure 4.1, when implementation is done by an FPGA topology is unknown. Place and route algorithms are applied to the design, so each of the redundant logic can be close or far from each other or the majority voter. When the wires are long they cause time and resource penalties. The ideal placement for TMR and voter logic would be to put the three redundant logic copies as close as possible and the voter next to them.
- Besides placement and routing inefficiencies, the logic copies and the voter are mapped to additional LUTs, and then the cost of fine-grain TMR is 3 extra LUTs for each function LUT or an overhead of 300%.
- For fine grain DMR the spatial overhead would be 200%, since each logic LUT is doubled, plus another LUT for the comparison. For DMR, the result of the comparison should be used somehow to trigger the recovery action, but this functionality is user/technology specific.
- Using coarser grain in TMR is a common technique for mitigating the area and speed cost of TMR. With larger blocks, the number of voters is reduced, and TMR cost is asymptotically 3 times the original cost (i.e. just the triplicated logic, with negligible voter cost). However, the TMR grain is actually a cost-reliability trade-off, as discussed earlier. Finer granularity offers improved fault detection and tolerance capabilities in the presence of multiple faults. This property is becoming increasingly important with new, deep-submicron technologies that are much more prone to SEU.

- Implementing TMR at LUT level, the finest granularity possible in FPGAs, while minimizing not only place and route penalties but also the 300% overhead is a challenge. But these goals cannot be succeeded through a typical high-level technique.

5.2 Vision and Goal

In this work we present an FPGA architecture that we suggest for fault tolerance support. The basic idea is to augment the LUT with TMR functionality reusing the existing structures as much as possible. Providing incremental changes to the LUT and slice structures will allow us to achieve low implementation cost at the hardware level, and lower TMR cost, both for area and latency. Our proposal involves the implementation of the voter circuit in full-custom logic within the LUT/slice structures. Also with every LUT having this functionality we achieve fine grain redundancy, so better fault tolerance. Furthermore our architectural changes are made, in a way that TMR mapping is given to user as an option.

We will present this work using the Xilinx Virtex 5 family architecture. This family, and any other similar to it, has an important property that makes our idea takes advantage of, to minimize the area costs of TMR. The property is that each LUT is constructed from two others, of half size each, and a multiplexer that chooses from the two outputs. Next section describes more clearly the Virtex 5 family's architecture.

5.3 Virtex 5 family architecture

Figure 5.1 depicts the Virtex-5 LUT structure. Each LUT can implement any arbitrary six-input Boolean function. Internally, a 6-input LUT is structured as two 5-input LUTs that can be used to implement two independent arbitrary five-input Boolean functions as long as these two functions share common inputs. In this case both O5 and O6 outputs are used. The two 5-input LUTs can be combined using the A6 input and an internal multiplexor, the two 5-input LUTs implement an arbitrary six-input function and output the result to O6.

Under certain conditions (for example there is no other function that uses the same 5 inputs with this one), the entire LUT is used to implement a single five-input function. In that case only the O5 output is used and one of the inner 5-LUTS has no

useful information (data)-it is redundant. Our approach is to exploit the redundancy and force the design to be mapped and placed only in simple 5-input LUTS so that we "reserve" with other half of each 6-input LUT. Then we will use the reserved redundancy to design our architecture.

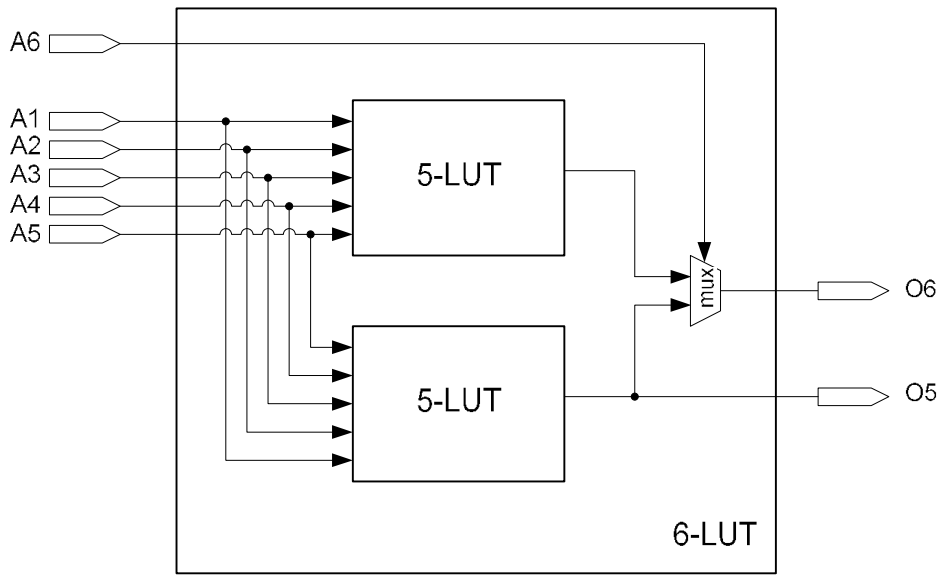


Figure 5.1: LUT structure for the Virtex-5 family. Two 5-input LUTs can implement independent functions that share the same inputs. Combined, they implement a 6-input function.

5.4 DMR Architecture

We explain our technique first for Dual Modular Redundancy (DMR) support. DMR is the first step to fault tolerance, since we can detect the fault's presence. First of all we make an important restriction: Every design will be mapped only to 5-input LUTs. Our architecture will be rested on this convention. We shall discuss the cost of this logical stride, later.

Having only one of the two 5-input LUTs active to implement the desired function, we can program the other one with the same function. Figure 5.2 demonstrates this LUT duplication for 5-iput function f. Two copies of the function are mapped in the two halves of the 6-input LUT. Comparison between the two outputs that would require a separate LUT, is done with an additional XOR gate inside the 6-input LUT.

Without SEU presence gate's output has value 0, and LUT's output can be either O5 or O6 since are identical. In DMR mode multiplexer selection signal is not useful. When a fault is present, the XOR gate will drive a high signal to the global column fault cable that will alert the system that a fault has been detected somewhere in this column. So after this alert no output should be accepted as correct and system must first reconfigure this column. Using this technique for fault detection has the advantage of on-line recovery from faults. Thus the additional DMR hardware implementation cost is a single xor gate for each 6-input LUT and the global fault presence wire(s).

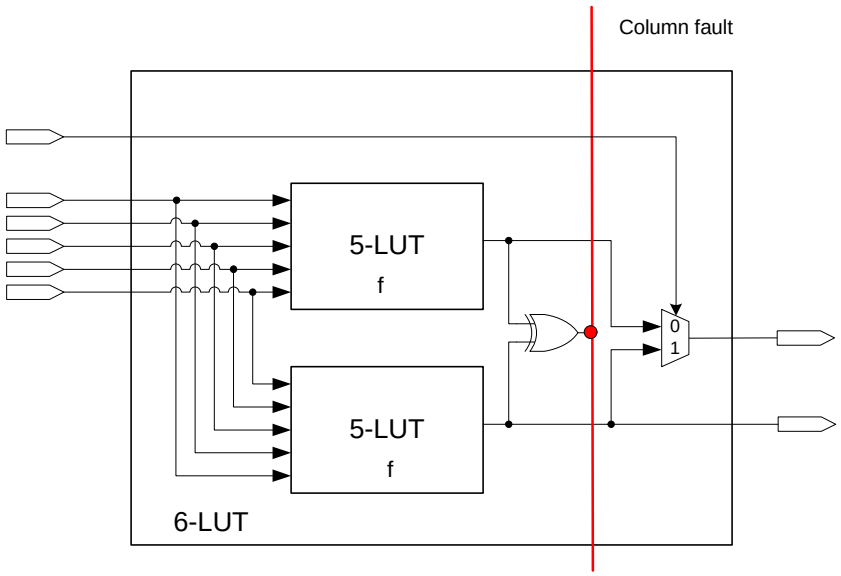


Figure 5.2: Dual-Modular Redundancy in our proposed LUT architecture. A single XOR gate compares the two results and signals fault when there is a mismatch.

However, an important detail is that there is another form of incurred cost: if DMR mapping is chosen by the user, then the mapped circuit cannot use 6-input LUTs. In cases where a full 6-input LUT would have been used (6-input functions or two 5-input functions with the same inputs), mapping tool must split them and map them to additional LUTs. We will address this cost in detail in the evaluation chapter 6.

5.5 TMR Architecture

Following the previous strategy we can get the TMR architecture by adding the necessary logic. We pose the same restriction, i.e. in TMR mode we allow only 5 -input functions and make an important observation: if two of the three copies needed are placed only in a single LUT as in the case of DMMR, there is no need to spend an entire 6-input LUT for the third copy of the function. Since the 6 -input LUT is actually two 5-input LUTs, they can be used to implement the third copy of two distinct functions. This idea is presented in figure 5.3. The three LUTs are combined to make the main cell of our architecture. This cell is referred as slice. Middle LUT should be able to implement two different functions, so for this reason, the two inner 5-input LUTs have separated inputs. This costs of course in additional cables. Thus for 2 functions we need 3 and not 4 LUTS. As for the DMR, besides the LUT implementation cost there is the cost incurred from restricting the mapping to 5 -input LUT only. If this cost is small (as we will show in the next section), then we have significant spatial reduction compared to classic TMR method.

The majority voter implementation follows this approach. We take advantage of the multiplexer that exists in the LUT structure to choose one of the two 5 -input outputs, and overload its functionality to select the correct TMR output. For function f, assume that we always take the first output (top one in figure) as correct (select 0 in multiplexer). We need to change this choice if a SEU is present in this copy. We determine this condition with the xor gate between 1st and 3rd copy. If the output is one then one of the two copies is wrong. However, we do not need to identify which one has the fault since under the single fault assumption the 2nd copy will be correct. So we select the 2nd output from the multiplexer. Consequently, we use this xor gate as the selector of the multiplexer, as figure 5.4 presents. As we can notice, the sixth input of each LUT is unnecessary in this TMR structure.

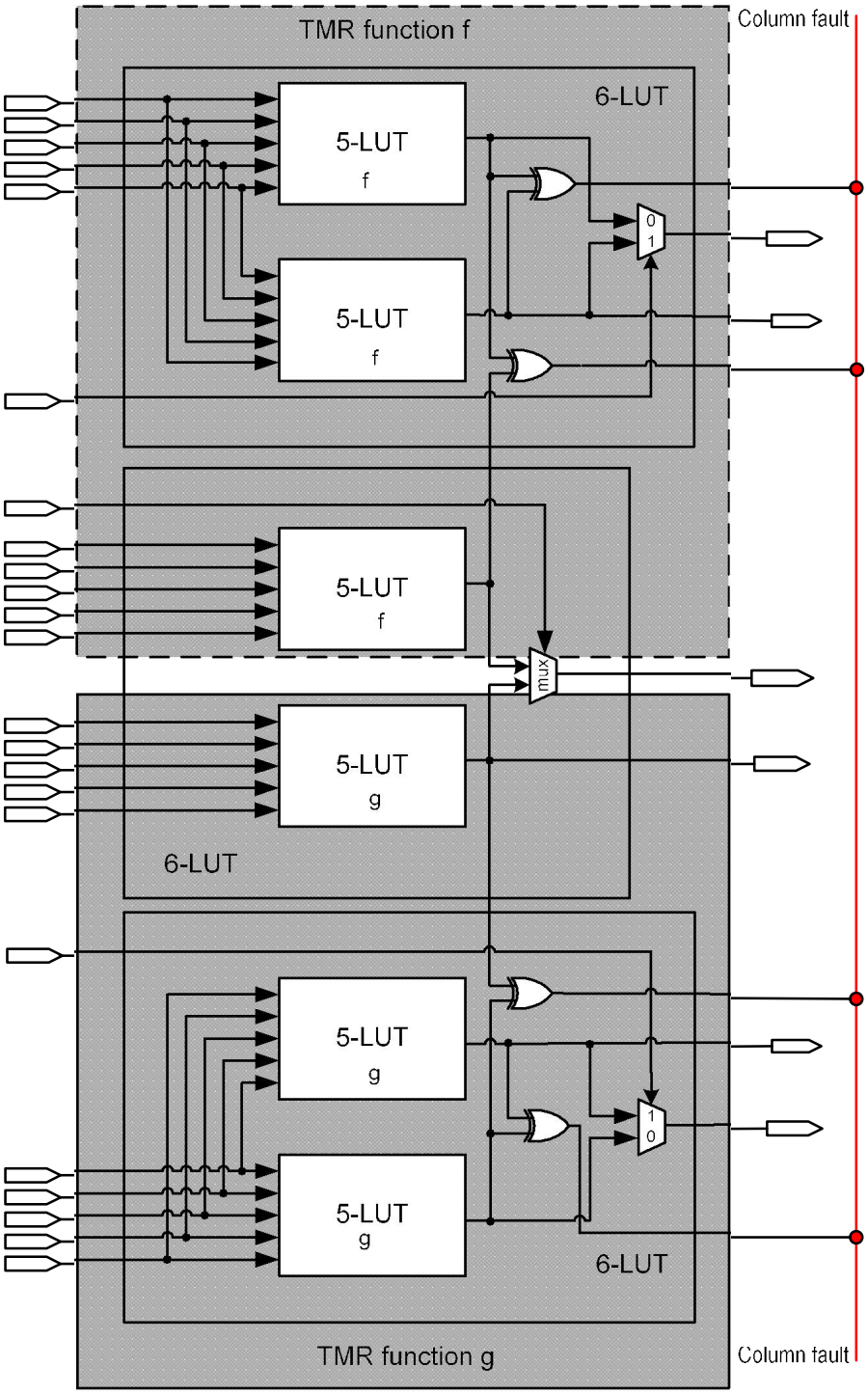


Figure 5.3: TMR mapping slice. For functions f and g we need 3 LUTs totally.

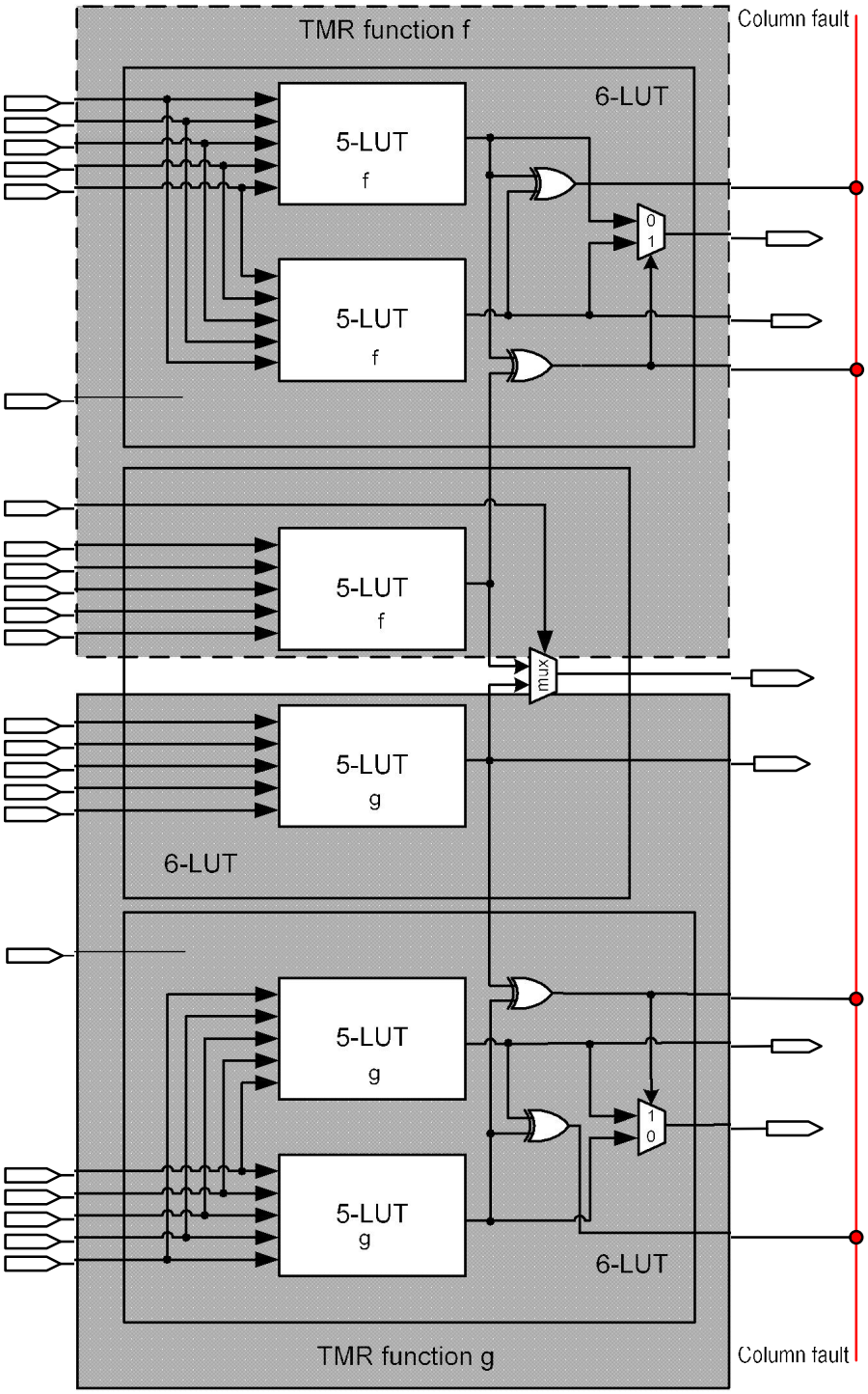


Figure 5.4: TMR's majority voter implementation. Multiplexer selects always a clear to fault copy of the function according to the xor gate of the 1 st and 3rd copy of each function.

In our system's architecture we want to provide optional TMR support. If TMR is not selected, then the multiplexer is controlled directly by the 6th input as it is in the original design (figure 5.3). In order to support TMR as an option, we add an FT signal (FT stands for fault tolerance) to select regular or TMR operation for this LUT. We envision this to correspond to a configuration bit. So now the multiplexer is controlled by a more complex function, rather than just a gate. The added gates implement the voter and control the multiplexer, as described above. The three LUTs together behave as a slice that is capable to map: a) three 6 -input functions (or up to 6 5-input functions as the original architecture supports), or b) two 5 -input functions that use TMR technique. We present the truth table as well as the equation for the right selection of the multiplexer.

INPUTS			OUTPUT
FT	A6	1xor3	Mux sel
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1

Table 1: Truth table of the Multiplexer's selector. Signal FT specifies if we have TMR mapping or not.

Mux_sel= (A6 AND FT') OR (1xor3 AND FT)

A6: is the sixth input of the LUT

1xor3: is the output of the xor gate between 1st and 3rd copy

FT: the configuration bit that decides the support of fault tolerance

Figure 5.5 presents the final architecture which provides optional TMR support. As in the DMR case, we use gates to discover the existence of faults and signal the fault's presence at a column granularity. Since the column fault is a high capacity wire, latching buffers should be used to decouple the LUT operation from the (slow) fault flag.

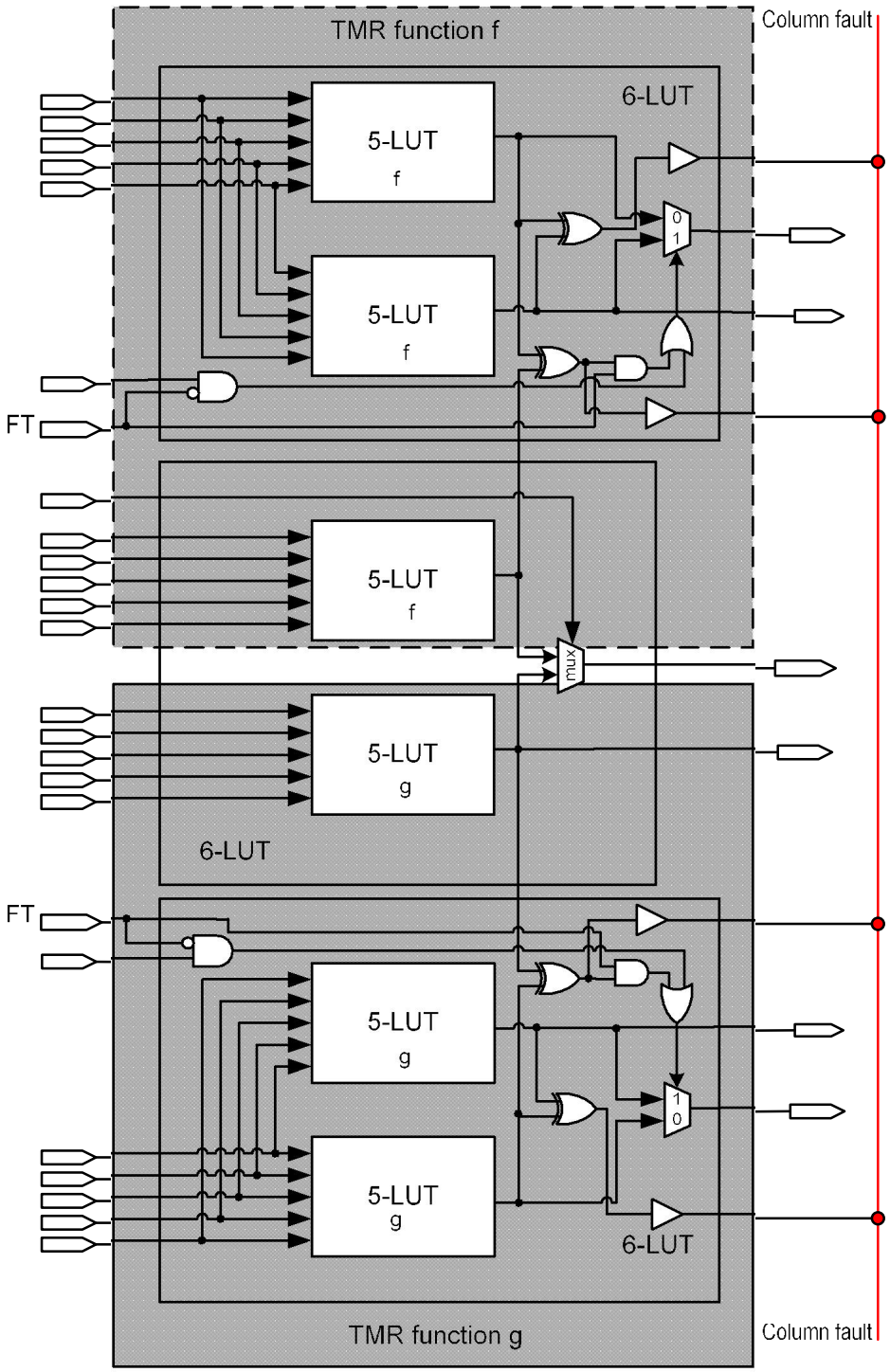


Figure 5.5: Triple-Modular Redundancy optional support slice. FT signal is responsible for standard or TMR function of the circuit.

The column fault signal can be used in to determine the recovery action. For FPGAs the commonly used technique is a preventive complete reprogramming of the FPGA in regular intervals (scrubbing). The fault detection signal can be used to initiate on demand the reconfiguration, avoiding in this way unnecessary reconfigurations. Even better, the column resolution can be exploited to perform a partial reconfiguration (supported in recent devices), saving additional time and energy.

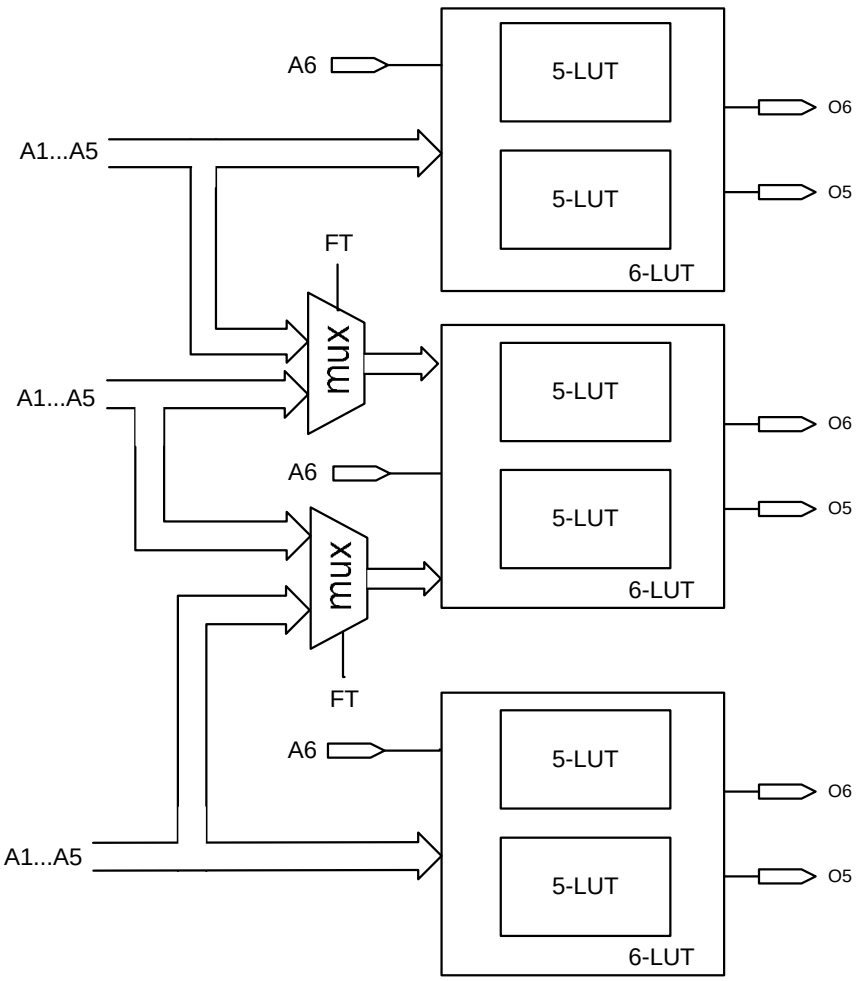


Figure 5.6: Slice block diagram. Under TMR the middle LUT implements two copies of independent functions. The MUXes select the independent inputs for regular and copies of the top and bottom inputs for TMR operation.

In addition to the voter gates, an important change in the requirement for independent input signals for the two halves of the middle LUT. This is because they implement copies of different functions. There are two implementation alternatives for this change: (i) extend the interface to add an additional set of (five) inputs, and use the routing logic to connect them to the appropriate signals, and (ii) keep the outer interface unchanged and use two multiplexers to decide if the middle LUT will be used as in TMR or not (figure 5.6). Option (i) results in smaller LUT cost but increased wiring and option (ii) leaves interconnect unchanged but increases the area cost and the LUT latency.

5.6 Discussion

This work implements TMR, atypical high-level technique, by an architectural approach. In this way designing cost of TMR is removed of user, who was responsible to redesign his circuit according the TMR rules. Instead of user, TMR implementation responsibility is passed to map tools. These tools have to take concern of our architecture and map properly the design in the new slices.

Supporting TMR as an option gives our architecture the advantage of flexibility. The same device can be used for ordinary or TMR mapping and so this type of FPGA has quite large target group of applications.

In case of TMR implementation our approach degrades the area cost compared to typical TMR implementation who has 200% overhead as the minimum area penalty. We calculate this decrease in the next chapter.

Providing signals that alert fault's presence has two benefits. First, fault detection and recovery occur on-line and system doesn't need to stop its operation. Secondly, fault alerting can reduce scrubbing rate because when we have information of zero fault, we can avoid unnecessary scrubbing. As a consequence system's power dissipation is reduced as well. Besides that, if we take advantage of partial reconfiguration we can refresh only the region of the fault. Of course we need to add a decoder component that checks the fault signal of each column and transmits information about fault's location. Further time and power is gained in this way.

TMR is supported in a way that succeeds very high upset tolerance, due to the granularity of the implementation. FPGA can mitigate as many faults as its slices, with the constraint that every slice has only one fault.

As every architectural approach, our suggestion has high cost of investment in development, test and fabrication. Adding the components described will slow down LUT's response and increase slice's area cost. Note that these costs ar e eliminated in DMR case. Finally this architecture cannot counter faults that have a permanent presence, thus destructive LUT faults. In this case LUT must be bypassed through another procedure.

Chapter 6

Evaluation

This chapter describes procedures and results in order to evaluate our architecture. First we present the high-level TMR scheme that we implemented. After TMR cost measurements, we calculate the area cost that occurs when using only half of 6-input LUTs. Two different tool flows were used. An academic, named AMDREL, and one using Xilinx tools. Comparisons and estimations according to results give the total cost of our TMR architecture.

6.1 Benchmarks

To evaluate the benefits and cost of our proposed architecture we applied th e TMR technique to a set of the ITC (International Test Conference)99 benchmarks. The ITC'99 benchmarks developed in the CAD Group at Politecnico di Torino (I99T) are a set of circuits whose characteristics are typical of synthesized circuits. For each bench both the RT-level VHDL description and the synthesized Gate-Level netlist are available. We used the VHDL description, where made the appropriate transforms, in order to derive the equivalent TMR circuit. Table 2 presents a small description of the circuits taken under concern. We used the following subset of ITC'99 benchmarks:

NAME	ORIGINAL FUNCTIONALITY
b01	FSM that compares serial flows
b02	FSM that recognizes BCD numbers
b03	Resource arbiter
b04	Compute min and max
b05	Elaborate the contents of a memory
b06	Interrupt handler
b07	Count points on a straight line
b08	Find inclusions in sequences of numbers
b09	Serial to serial converter
b10	Voting system
b11	Scramble string with variable cipher
b12	1-player game (guess a sequence)
b14	Viper processor (subset)
b15	80386 processor (subset)

Table 1: Circuit description

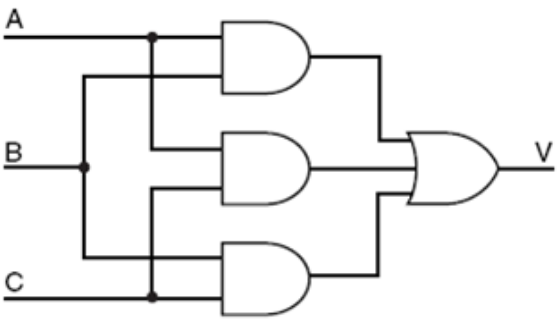
Our proposed architecture is motivated by the Vitrex-5 LUT structure, so we used this FPGA family in our experiments. To implement TMR we followed Xilinx's TMR design guide specific for Virtex FPGAs, that outlines the recommended design methodology for constructing and implementing TMR logic within the Virtex architecture. More specifically, we followed the TMR State-Machines section and that because all our test circuits were written as FSMs.

6.2 TMR Design Technique

6.2.1 TMR State-Machines

Finite State-Machines (FSMs) are an inevitable aspect of digital design. FSM circuitry implements sequential processing, sequencing control, and decision-making algorithms. Development methods for FSM circuits range from explicit Case statements to FSM compilers.

State-machines create registered feedback logic loops that must be replicated and voted in order to assure reliability against SEUs. The TMR implementation of such circuits might seem a bit tedious, but its effectiveness is well worth the effort.
An appropriate encoding scheme must be selected for the FSM. Symbolic encoding will not work for implementing TMR state-machines, because the majority voters must be inserted into each register feedback path. Therefore, state-machine encoding must be explicit. The registered State signal forms a registered logic loop. Therefore, this is the insertion point for majority voters when replicating the triple redundancy.



A	B	C	V
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 6.1: Majority Voter Circuit and its truth table

Device outputs are the key to the overall TMR strategy. Since the full triple module redundancy generates every logic path in triplicate, there must ultimately be a method for bringing these triple logic p aths back to a single path that does not create a single point of failure. This can be accomplished with TMR outputs. A TMR output is constructed using the OBUFT library primitives as shown in the next figure.

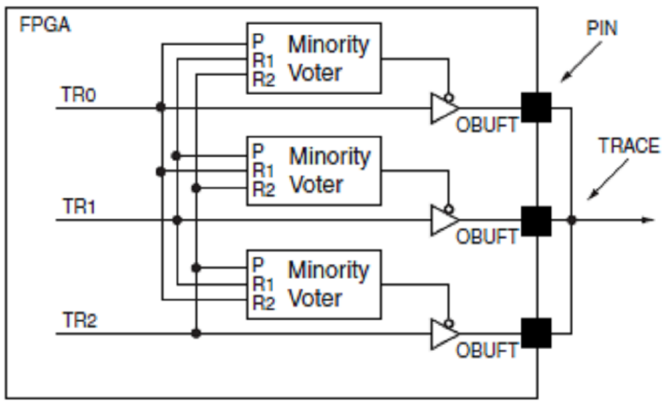
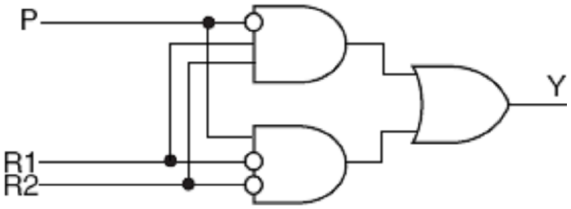


Figure 6.2: Minority Voted TMR FPGA Output s

Each redundant logic path exiting the FPGA on an output does so through an OBUFT. The “enable” (T pin) of each OBUFT is controlled by a “minority voter” circuit. The minority voter indicates whether the path in question (primary path) agrees with either of the two redundant paths. If the primary path agrees with at least one of the redundant paths, then the primary path is considered to be part of the majority. If the primary path disagrees with both redundant paths, then the primary path is in the minor ity. The minority voter is shown in Figure 6.3. If the primary path is part of the majority, then the minority voter will enable the corresponding (active Low) OBUFT allowing the data on its primary path to be driven out through the OBUFT and onto the Pad -Pin. If the primary path is not a part of the majority, then the OBUFT is disabled placing its output in a high -impedance state allowing the redundant outputs to drive the correct data.



P	R1	R2	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Figure 6.3: Minority Voter Circuit and its truth table

According to the previous design steps, the transformation of an FSM in order to have the TMR version of the same state machine can be described by the following figures:

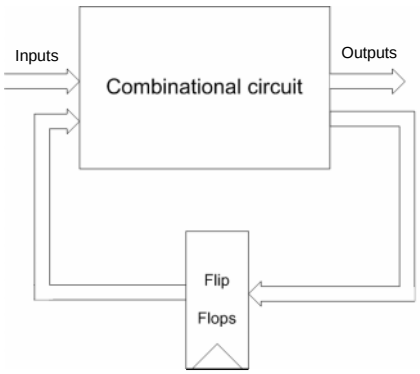


Figure 6.4: Simple FSM

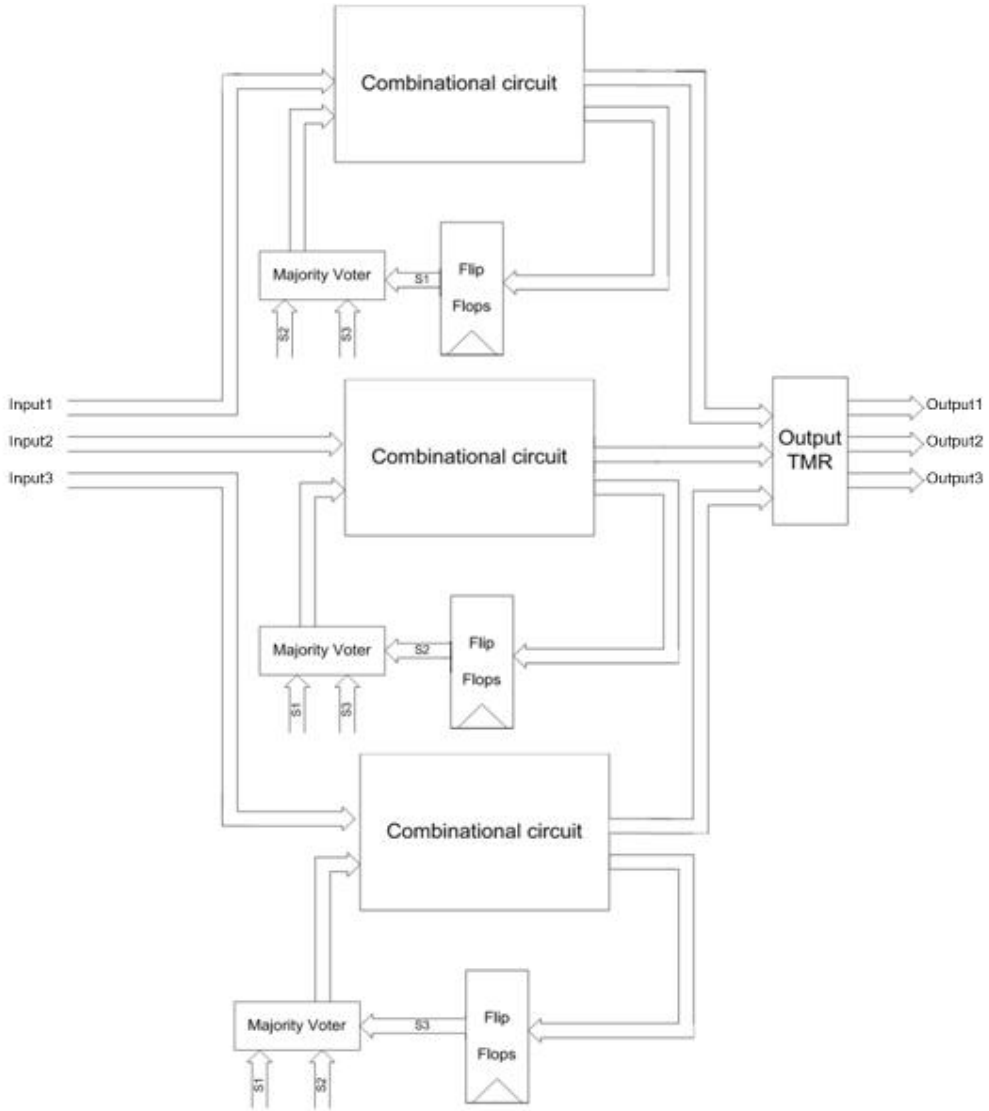


Figure 6.5: TMR FSM

Following the previous scheme we transformed our VHDL benchmarks in order to have designs with TMR support. All changes were made manually.

6.3 Synthesis Parameters

The tool Xilinx ISE 10.1 was used for the implementation of the vhdl code. The target device was the Virtex5 XC5VLX30, one of the smallest FPGAs of this family. We synthesized the circuits first using the original source, and then we synthesized the TMR version.

Synthesis process when having redundancy is not a straightforward task. CAD tools are designed in order to optimize user design and move any redundant component. When observing figure 6.5 we notice that each of the three majority voters, added, have the same three inputs. ISE finds this condition and removes two of the three majority voters. This optimization which is most welcome in typical cases, now is an unwanted one. Same undesirable optimization happens at another CAD tool we tried, Synplify Premier 9.6.1. Although it seemed easy to bypass this obstacle, it came up as a crucial problem. The reason is that these tools have been created to have as much optimizations as possible and some of them –the most obvious maybe- are not optional. Some properties that should have solved this problem did nothing. Finally after several experiments we found a way to avoid the optimization.

We had to use synthesis directives in the form of supported VHDL attributes. In the code of majority voter we inserted an attribute that is responsible for optimizations. Its value was set as "OFF". We give the code in figure 6.6.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity maj_voter is
        port(
                A       :in std_logic;
                B       :in std_logic;
                C       :in std_logic;
                V       :out std_logic);

attribute optimize : string;
attribute optimize of maj_voter : entity is "OFF";

end maj_voter;

architecture Behavioral of maj_voter is

begin

        V<=(A AND B) OR (A AND C) OR (B AND C);

end Behavioral;
```

Figure 6.6: VHDL code of majority voter. Attribute optimize was set to "OFF".

Except this change we also had to choose a proper synthesis option in ISE. So we select the following: at synthesize – XST ->Synthesis Options we set the attribute Keep Hierarchy to “Soft” (property display level must be advanced). If we left it “No”, which is the default value, it still does the optimization. The other choice is “Yes” but with that choice we couldn’t accomplish the implementation step. The two changes eliminated the optimization problem.

Since we were interested to have the minimum area cost another attributed was changed also in order to have minimum resources util ization. At synthesize – XST right click Properties->Synthesis Options we set the attribute Optimization Goal to “Area”. Default value is “Speed”. Next figure (6.7) presents a typical set of our parameters during the tests.

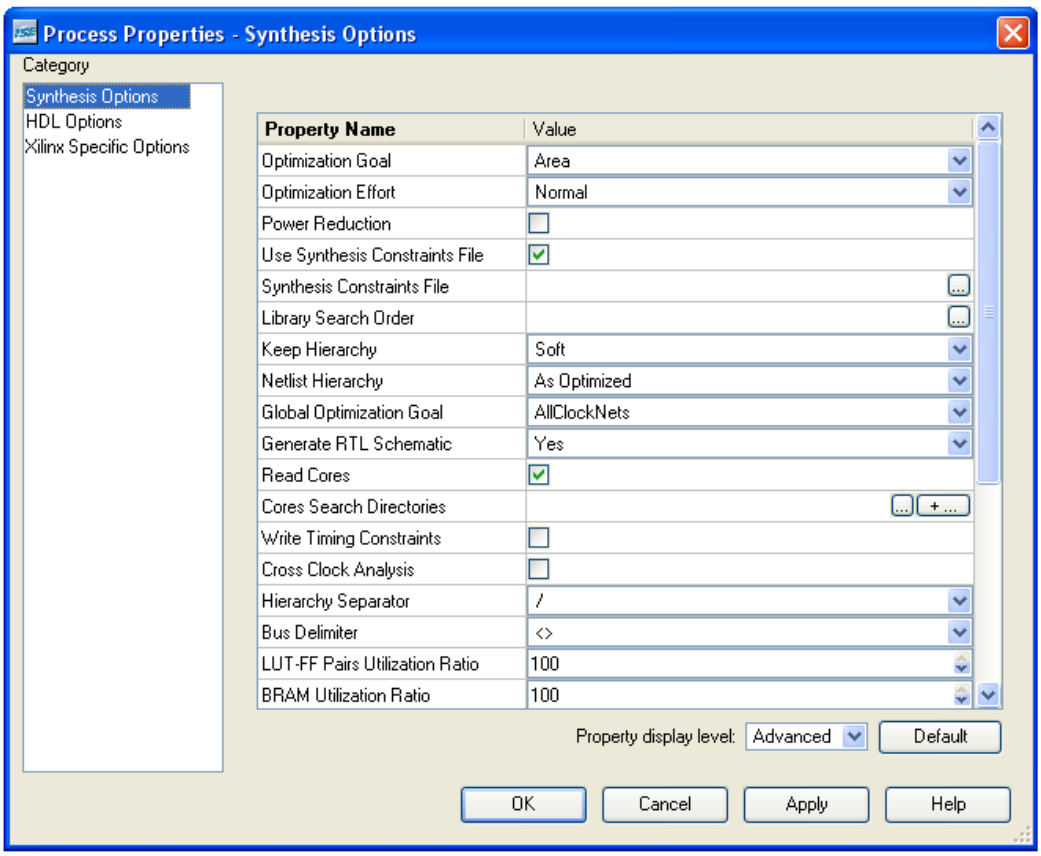


Figure 6.7: Synthesis Options of ISE tool during TMR design evaluation.

In order to compare designs with same synthesis options the above methodology was used for simple as well as TMR designs of the same benchmarks.

After area calculation of common TMR designing we had to find the area cost of our architecture. First we had to find the space overhead when restricting the mapping to 5-input functions. This is not obvious. Unfortunately Xilinx does not have any FPGA with 5-input LUTS. Families before Virtex 5 use 4-input LUTs. For this reason overhead computation was based on estimation. In order to verify our estimated results we use, except Xilinx, another cad tool flow. It was taken from AMDREL (Architectures and Methodologies for Dynamic Reconfigurable Logic) , a project develops methodologies, tools and intellectual property blocks to be integrated in a mixed granularity dynamically reconfigurable SOC implementation platform for the efficient realization of wireless communications systems. Next sections describe AMDREL and ISE flow that was applied in order to obtain the desirable results.

6.4 AMDREL Flow

6.4.1 The Supported FPGA Architecture

In the AMDREL project, a fine-grain reconfigurable block is included between the different functional blocks of the Mixed-granularity AMDREL platform. The most popular “island style” architecture, where an array of logic blocks are surrounded by routing channels, as illustrated in figure 6.8, is the general architecture that is supported by the AMDREL design framework. The I/O pads are evenly distributed around the perimeter of the FPGA. The used configurable logic blocks are cluster based.

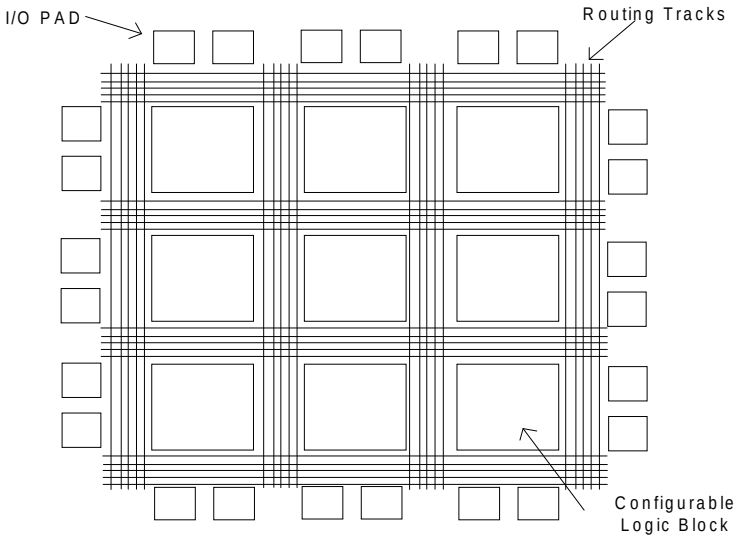


Figure 6.8: Island Style Architecture

Configurable Logic Block (CLB) Architecture

The Logic Block in the cluster-based architecture has two levels of hierarchy; the overall block is a collection of basic logic elements (BLEs). In addition with the reduced area, local interconnect between BLEs results to better routing flexibility. Figure 6.9 shows the structure of the two-level hierarchy Logic Cluster. Figure 6.9a shows the structure of the BLE, which is formed by a LUT, a D-FF and a two to one multiplexer. Then these BLEs, connected together, with the use of I+N to one multiplexers as in Figure 6.9b, form the Logic Cluster. There are a number of parameters that their values have to be determined. These parameters are:

a) the number of the inputs of the LUT (K)
b) the number of BLEs in the CLB (cluster size, N), and
c) the number of the inputs of the CLB (I).

The Configurable Logic Block (CLB) was implemented at the physical level (layout) with the 0.18um STM technology. All information, required by the supported tools to perform simulations, such as capacitance, resistance, delays etc, was extracted from this physical implementation, to ensure a more accurate approach.

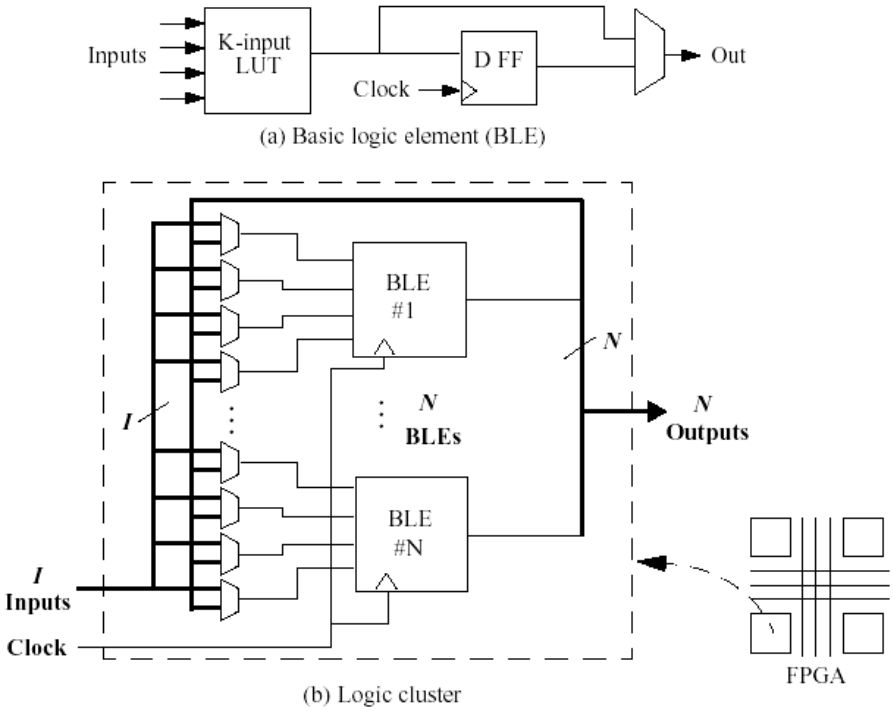


Figure 6.9: Structure of Basic Logic Element (BLE) and Logic Cluster

6.4.2 Exploration Flow

A number of open source software tools were used in our exploration flow. Figure 6.10 illustrates this flow and the corresponding tools. All these tools are part of the AMDREL design flow.

At first each benchmark that is syntactically correct passes through technology-independent synthesis using Leonardo. In this way we retrieve an EDIF file which is a netlist presentation of the circuit, suitable for the next tool. DRUID is a tool that converts the EDIF format netlist produced by a commercial synthesis tool to an equivalent EDIF format netlist compatible with the next tool of the design flow. E2FMT is used for translation of the netlist from EDIF to BLIF format. Logic optimization is performed using the SIS program. SIS is used also for mapping the optimized circuit into 5-LUTS or 6-LUTS and DETF/Fs using one of the seven mapping algorithms supported. Then an estimation of the activity at the nodes of the benchmark circuit is performed using the ACE tool. The mapped benchmark circuit with the activity information is then fed to the T-VPACK tool that performs the packaging of BLEs into logic clusters. VPR uses the packed benchmark and the activity information to place and route the circuit in an FPGA with the desired

architecture and with minimum number of tracks. The circuit is re -routed with VPR under a Low-Stress condition giving an extra 20% numbe r of tracks. The PowerModel tool gives then powe r estimation, in addition to delay information which has been taken from VPR.

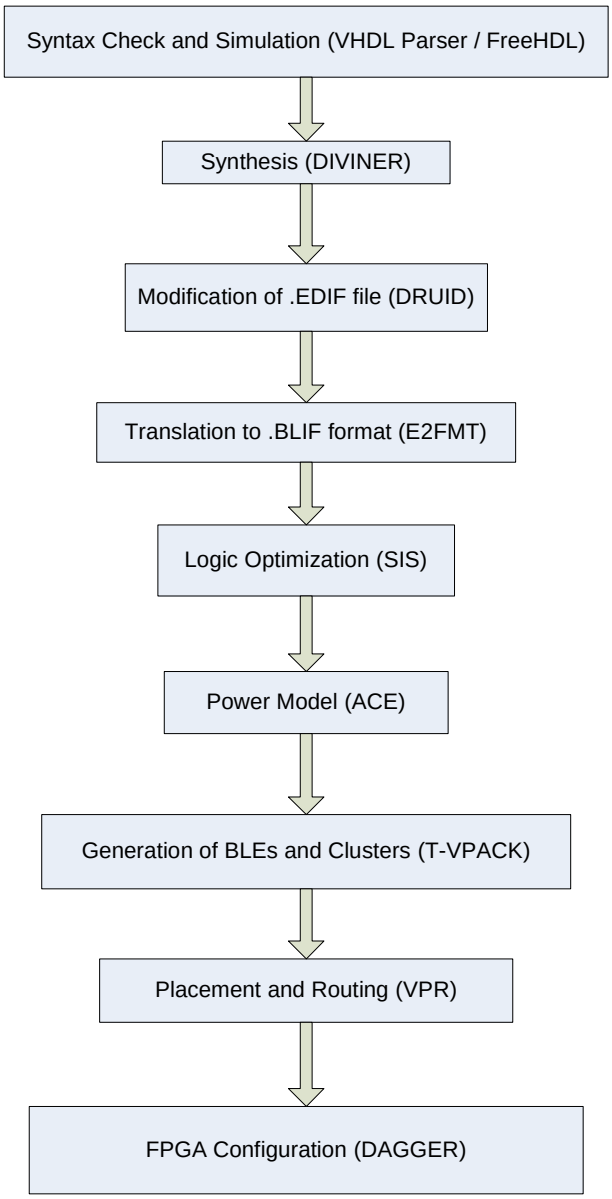


Figure 6.10: AMDREL design framework

In this work we are interested in the area cost of a mapped design. For this reason, information about power has not been retrieved. We noticed that DRUID created a file with syntax errors most of the times. Those errors corrected manually in order to take the desirable EDIF files. Unfortunately there were benchma rks that

couldn't pass the entire flow successfully. The problem was that these designs used memory blocks and the particular toll flow cannot support memory implementation. That is the reason why we present results for less benchmarks than those showed before. During SIS operation we choose the LUT size and the mapping algorithm. For each benchmark we run SIS having 4, 5 and 6 input LUTs. We didn't choose a specific mapping algorithm because we were interested in having the best result. So we selected "ALL" option where, the algorithm with best results is used. Finally we used ACE tool to obtain useful information about the number of BLEs and LUTs.

6.5 ISE Flow

To compute the previous overhead, using Xilinx's tools, we used the following technique: Fist we synthesized each benchmark using a Virtex 4 FPGA that uses a 4-input LUTs. Then we took the derived netlist (ngc file) and implemented it using Virtex 5. ISE tool gives a mapping option that controls input number of mapped function. We employed this method three times for four-, five- and six-input LUTs respectively. To make sure that this procedure leads to reliable data we compared results of 4 and 6 input mapping with normal implementation of same designs in Virtex 4 and Virtex 5 and found this approach to be accurate.

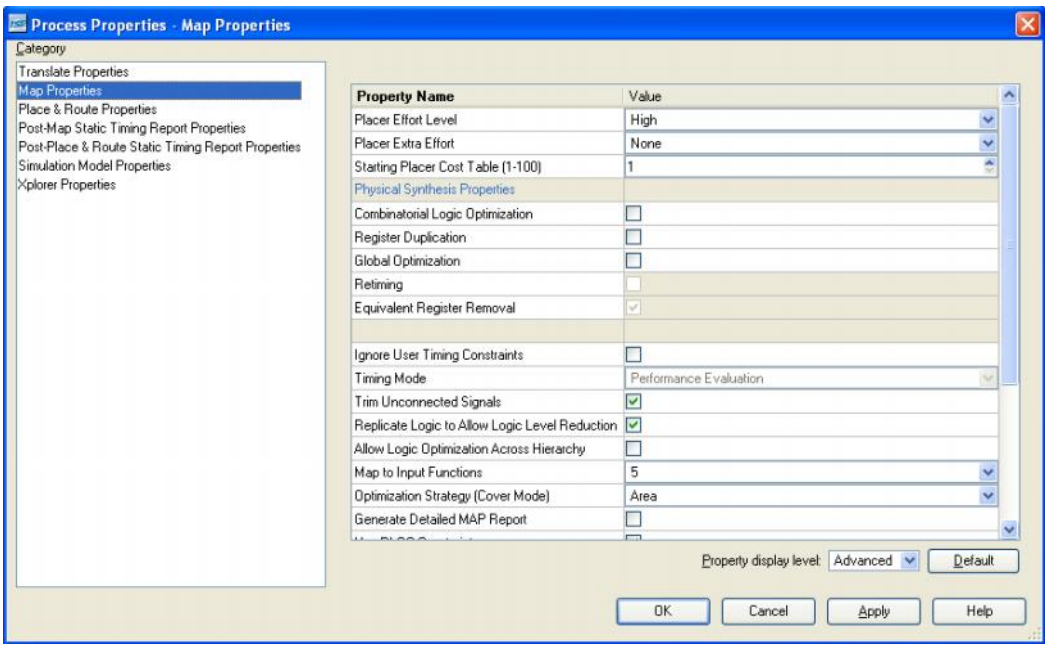


Figure 6.12: Implementation Options of ISE tool. We change the value of Map to Input Function property.

ISE 10.1 tool was used to synthesize the codes. The target device was the Virtex4 XC4VLX25, a device which was similar to the one we used from Virtex5 family throughout this work. Synthesis properties were the same as those we used during high-level TMR evaluation. After having the ngc file, we created a new project and interchange the device by Virtex5 XC5VLX30. We selected the type of top-level source for the project as ngc. With this option ISE can do directly the implementation process. Before running this process we select the following: at implement design menu select Properties->Map Properties we set the property named Map to Input Functions to 4, 5 and 6 (property display level must be advanced), one at a time. By this selection we attempt to emulate the mapping procedure as if we had real 4, 5 or 6 input LUTs. Figure 6.12 shows an example of 5-input function. We remind that there isn't a Virtex device with a 5-input LUT architecture.

6.6 Results and comparison

6.6.1 TMR overhead

Our first aim is to measure the overhead of common TMR implementation. Synthesis report costs for single and TMR designing is presented in Table 2. Besides triplicating the logic as outlined earlier, the TMR methodology inserts majority voters after flip-flops to vote the state of the FSM. Another voter is needed for each output to choose one of the three copies of the output signals. This overhead logic is the reason why smaller designs have a larger relative overhead (b01, b02, b06, b10). For larger designs the final TMR area increases by a factor that ranges in between 3 and 4 times. Results comnfirm that coarse grain TMR costs over than 3x of initial design in spatial resources.

benchmark	unmodified circuit	TMR (unmodified LUT structure)	TMR overhead
b01	5	30	6x
b02	4	24	6x
b03	37	120	3,24x
b04	96	324	3,38x
b05	170	600	3,53x
b06	8	51	6,38x
b07	82	294	3,59x
b08	19	84	4,42x
b09	34	138	4,06x
b10	33	165	5x
b11	93	321	3,45x
b12	247	870	3,52x
b14	932	2958	3,17x
b15	1939	6828	3,52x
Total	**3699**	**12807**	**3,46x**
Arithmetic Average			**4,23x**

Table 2: Number of LUTs for the regular and TMR circuit versions.

6.6.2 Area ovehead from half using LUTs

In this section we present the results implemented benchmarks in or der to calculate the ovehead inserted by allowing mapping only in 5 -input LUTs instead of 6-input LUTs. We name this cost as, DMR ovehead, because this equals to the area cost of our DMR architecture.

Table 3 presents the results taken from AMDREL flow. Nine benchmarks have passed this flow successfully.

	4-input LUT	5-input LUT	6-input LUT	DMR overhead (#LUTs)	DMR overhead (%)
b01	12	9	5	4	80%
b02	7	4	4	0	0%
b06	17	10	9	1	11%
b07	190	148	117	31	26%
b08	52	43	31	12	39%
b09	81	59	48	11	23%
b10	123	82	53	29	55%
b11	250	185	150	35	23%
b12	1415	918	766	152	20%
Total	**2147**	**1458**	**1183**	**275**	**23%**
Average					**30.8%**

Table 3: Synthesis results using the AMDREL design flow: number of LUTs required using 4, 5 and 6 input LUT.

We measured the number of 4-input LUTs because we would like to compare these results with the results gained from ISE, with a different technique. As we can notice the overhead, when changing LUTs from 6 to 5 -input size, is not as much as we could think. Although LUT's size is decreased to half, the LUTs number overhead has an average of 23%. Fig. 6.11 presents results scaled to the cost of 4-input LUTs.

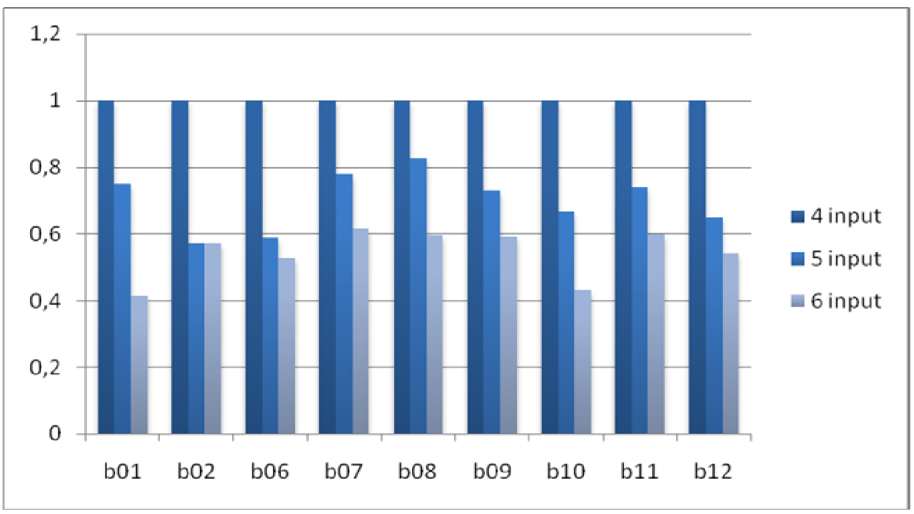


Figure 6.11: Synthesis results using the AMDREL design flow: Area overhead for 5 and 6-input LUTs, scaled to the cost of 4-input.

Table 4 presents the different spatial costs taken from procedure using ISE flow. The area cost is also shown in Figure 6.13 that presents the same data scaled to the cost of 4-input LUTs. The reduction in the bars corresponds to the benefits from using larger LUTs.

	4-input function	5-input function	6-input function	DMR overhead (%)
b01	13	5	5	0%
b02	4	4	4	0%
b03	65	59	49	20%
b04	135	132	129	23%
b05	234	212	187	13%
b06	9	8	8	0%
b07	121	105	93	13%
b08	33	28	27	4%
b09	49	46	38	21%
b10	52	48	37	30%
b11	131	123	107	15%
b12	374	336	291	15%
b14	2420	1844	1664	11%
b15	2972	2696	2330	16%
Total	**6612**	**5646**	**4969**	**14%**
Average				**12.9%**

Table 4: Synthesis results using the ISE flow: number of LUTs required, mapping to 4, 5 and 6 input functions.

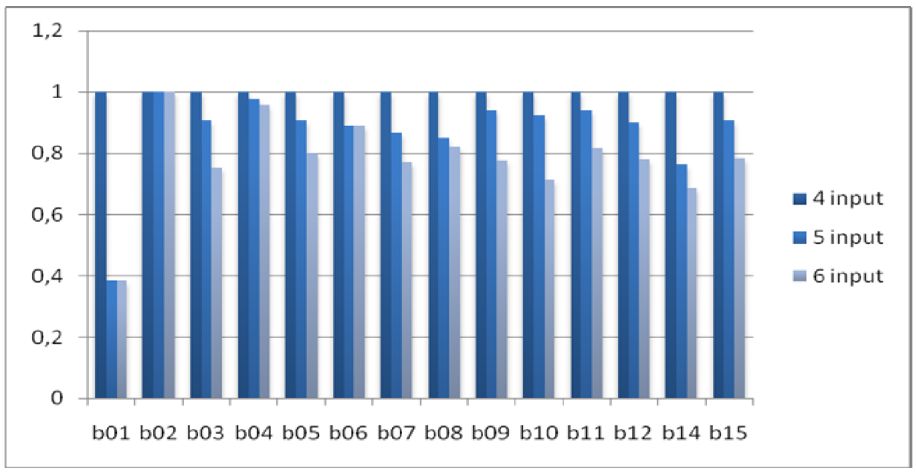


Figure 6.13: Synthesis results using the ISE flow: LUT overhead for 5 and 6-input LUTs, scaled to the cost of 4-input.

As we can observe in Table 4, the DMR cost is in most cases between 10% and 20%, and is always smaller than 40%. The average result of 14% is much lower than the AMDREL results. Even we remove some benchmarks in order to have the same subset we end with an average of 15%.

Next step was to try to estimate if this procedure leads to reliable data. We compared results of 4 and 6 input mapping with normal implementation of same designs in Virtex-4 and Virtex-5 to see how similar the results are. With same Virtxex-4 and Virtex-5 devices and same synthesis options, implementation results were better than the previous procedure but the improvement was similar to both directions. Table 5 compares these results.

	Virtex4 (4-input LUT)	Virtex5 (6-input LUT)	Native Virtex-4/5 implementation LUT overhead (%)	Input function 4/6 implementation LUT overhead (%)
b01	9	5	80%	160%
b02	4	4	0%	0%
b03	64	37	73%	33%
b04	135	96	41%	5%
b05	220	170	29%	25%
b06	9	8	13%	13%
b07	112	82	37%	30%
b08	32	19	68%	22%
b09	47	34	38%	29%
b10	52	33	58%	41%
b11	122	93	31%	22%
b12	341	247	38%	29%
b14	1921	932	106%	45%
b15	2496	1939	29%	28%
Total	**5564**	**3699**	**50%**	**33%**
Average			**45.8%**	**34.4%**

Table 5: Native ISE implementation results and comparison to ISE flow using input function mapping parameter.

As we can see 6 to 4 input overhead is 50% with normal implementation, quite more than 33% that observed using the input function mapping method. But the

difference between 6 to 5 input overhead will be less. Figure 6.14 compares 6 to 4 input LUTs area overheads, from AMDREL and ISE implementation. We can observe that ISE presents less area overhead. Nevertheless, in 6 to 5 case, DMR overheads present convergence.

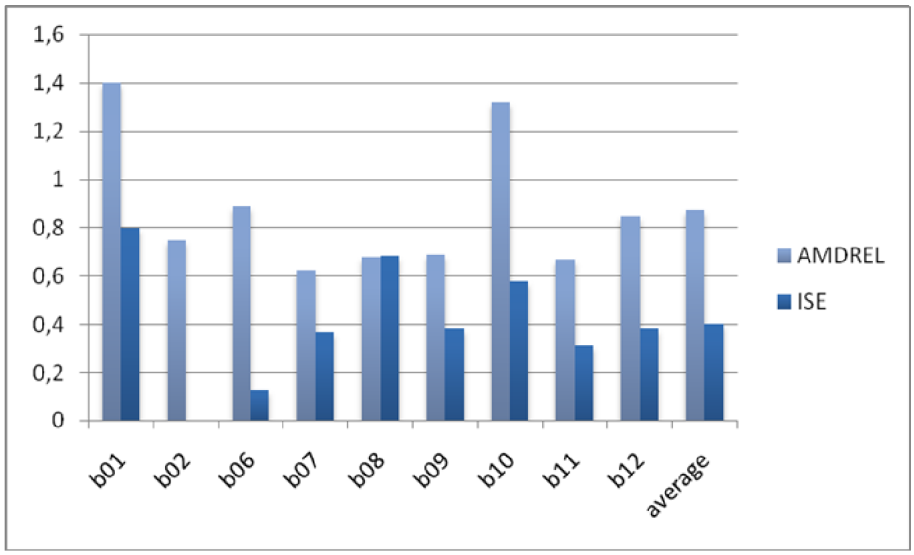


Figure 6.14: Spatial overhead for restricting mapping from 6 to 4-input LUTs: Comparison of AMDREL and ISE results.

If we assume that there is a linear relation between the two procedures, then we can have estimation of this overhead as if we use a real 5 input LUT Virtex.
We divide the two overheads from the first technique:

[%(6 to 4) overhead] / [%DMR overhead] =33/14=2.357.

If we assume that this ratio is fixed then we estimate 6 to 5 overhead from normal implementation as:

[%DMR overhead]= [% (6 to 4) overhead]/2.357=50/2.357=21%

This estimated value is very close to the overhead measured by AMDREL flow.

A more pessimistic estimation is that moving from 6 to 5 input LUTs gives half of the overhead between 6 and 4 input. So we get the value 25%.

Finally if we count the previous ratio from AMDREL's results and follow the make division, we obtain again an optimistic value, 14%.

METHOD	%overhead (6 to 5)
AMDREL flow	23%
Input function selection implementation	14%
Estimation combing ISE results	21%
Pessimistic estimation from normal implementation	25%
Estimation combing AMDREL and ISE results	14%

Table 6: Overhead estimation for each method

Table 6 summarizes all the estimated overhead. In our opinion the third method (21%) gives the most reliable results. Even with the most pessimistic approach the area overhead cannot exceed 30%. By inference, forcing a Virtex 5 like FPGA to use only half of each 6-input LUT can't have dramatic area penalties. And we suggested, free space will be used by redundancy techniques.

Literature was another source to confirm our experimental results. Xilinx has published a white paper named "Retargeting Guidelines for Virtex-5" [14]. There it points that Designs that run fast in a Virtex-4 device (i.e., at 300 MHz or above) see little improvement in performance or reduction in LUT utilization with a Virtex -5 device.

This is because fast designs that are well-optimized to the 4-input LUT structure, with low fan-in logic and few logic levels between synchronous objects, are more likely to have one-to-one mapping from the 4-input LUT to a 6-input LUT. This means that there is little potential for LUT or logic reduction with the new Virtex -5 device logic structure, and thus, little improvement in performance.

Many times, the retargeted designs that benefit most from the larger LUT structure in Virtex-5 devices are those that previously ran relatively slowly, with many logic levels and larger LUT-to-register ratios. In these designs with large fan-in logic cones, there is greater potential for the 6-input LUT to considerably reduce the number of logic levels as well as the number of LUTs necessary to build a logic

function. Such designs achieve a greater LUT reduc tion and performance boost when retargeting to the Virtex-5 device architecture.

Some types of logic functions benefit from the 6-input LUT more than others. For example, a 32-bit XOR gate consumes seven 6-input LUTs (with some logic to spare) where the same function would require eleven 4-input LUTs (with no logic to spare). This represents a 36% reduction in the required number of LUTs. However, a single 2-to-1 MUX maps into a single 4-input LUT in the same manner as a single 6-input LUT, so there is no advantage to using the 6-input LUT. A 2-input adder requires one 4-input LUT and one 6-input LUT per bit of addition. Soft multipliers do not benefit very much from the Virtex-5 device logic structure over previous generations. Thus, certain types of desi gns that use MUXs, adders, soft multipliers, and other logic functions do not get better utilization or performance from Virtex-5 devices. For example, DSP designs generally use these logic functions and a lot of pipelining as the core for many of their op erations and, therefore, do not realize many benefits from the Virtex-5 device architecture."

So the area reduction is about 36% in good cases, in case we switch Virtex 4 with Virtex 5. This corresponds to a 56% overhead in case we go backward, from a 6-input LUT to a 4-input LUT. This pessimistic value agrees perfectly with our calculated value 50%. If we assume that area reduction from 5 to 6 input LUT is half of 36%, thus 18%, then we obtain an overhead of 22% from 6 to 5 mapping. Again this result agrees with our estimation.

Using these results we can compute the cost of DMR and TMR using our architecture. On average, reducing the mapping to 5-input LUTs increases the average area cost by about 14%-25% , with best estimation 21%. Hence, this would be the cost of supporting DMR in our proposed architecture. To find the expected TMR cost, we must compute how many more LUTs we need. As described earlier, implementing TMR for 2 (5-input) LUTs we use a third one to have triple redundancy. So the overall area cost is 1.5 times the cost of the circuit us ing 5-input LUTs, or 1.5 * 1.21 times (best estimation) the original 6-input mapped design. Therefore support ing fine grain TMR in our proposed architecture increases the area by a factor of 1. 815 times compared to at factor of at least 3 in traditional TMR even for coarse granularity. These results are summarized in Table 7 and Table 8.

METHOD	TMR spatial cost
AMDREL flow	1.845x
Input function selection implementation	1.71x
Estimation combing ISE results	1.815x
Pessimistic estimation from normal implementation	1.875x
Estimation combing AMDREL and ISE results	1.71x

Table 7: TMR spatial cost estimation for each method

No fault tolerance	High-level DMR	Our DMR	High-level TMR	Our TMR
1x	**2x**	**1.21x**	**3.46x**	**1.815x**

Table 8: Comparison between high-level TMR and proposed architecture spatial cost.

Chapter 7

Conclusions and future work

In this thesis we propose a new LUT structure for efficient TMR support in Virtex-5 like LUT structures. We reuse as much as possible the existing LUT circuitry and augment it to provide TMR support at reduced cost. Our technique achieves significant cost savings for TMR operation at the cost of a few gates overhead per LUT. Since these modifications would be done in full-custom logic, the overhead is very small.

Supporting TMR as an option gives our architecture the advantage of flexibility. The same device can be used for ordinary or TMR mapping and so this type of FPGA has quite large target group of applications. Our approach offers several advantages: it allows the use of DMR with just 21% and TMR with 81.5% area overhead compared to 100% and 246% for the traditional implementations. It also provides a fault detection mechanism and column-based fault location, minimizing the number of required reconfigurations, saving both reconfiguration time and energy. It also allows the extensive use of fine grain TMR that offers the best fault tolerance and resolution.

Our work concentrated on LUT configuration faults only. A considerable portion of the FPGAs programming bits refer to the interconnection network, for which this approach is not possible. Providing fault interconnection can be implemented by TMR or other architectural techniques. Designing an FPGA architecture that uses our slice structure, is a challenge.

Next step is VLSI implementation of the FPGA. This is a difficult task that needs investment in development, test and fabrication. Another issue is the development of tools. An efficient mapping tool will take advantage of our architecture and map any design providing TMR or not according to user's decision. Combing FPGA chip and a supporting tool flow we can have a novel development platform that provides fault tolerance implementation.

References

Internet

[1] The official Xilinx Company site, http://www.xilinx.com/

[2] MEANDER - FPGA DESIGN FRAMEWORK :http://proteas.ee.duth.gr/meander/meander.html

Bibliography

[1] KASTENSMIDT F. L., CARRO L., REIS R. Fault-Tolerance Techniques for SRAM-Based FPGAs. Series: Frontiers in Electronic Testing , Vol. 32

[2] NASA. Radiation Effects on Digital Systems. USA, 2002.

[3] KATZ, R. et al. Radiation effects on current field programmable technologies. IEEE Transactions on Nuclear Science, New York, v.44, n.6, p. 1945-1956, Dec. 1997.

[4] O'BRYAN, M., LABEL, K. Recent Radiation Damage and Single Event Effect Results for Candidate Spacecraft Electronics. In: IEEE NUCLEAR SPACE RADIATION EFFECTS CONFERENCE, NSREC, 2001.

[5] BARTH, J. Applying Computer Simulation Tools to Radiation Effects Problems. In: IEEE NUCLEAR SPACE RADIATION EFFECTS CONFERENCE, NSREC, 1997.

[6] MOORE, G. E. Progress in Digital Integrated Electronics. Digest of the 1975 International Electron Devices Meeting, New York, p. 1113, 1975.

[7] SIA SEMICONDUCTOR INDUSTRY ASSOCIATION. The National Technology Roadmap for Semiconductors. USA, 1994.

[8] JOHNSTON, A. Scaling and Technology Issues for Soft Error Rates. In: RESEARCH CONFERENCE ON ELIABILITY, 4., 2000.

[9] O'BRYAN, M. et al. Current single event effects and radiation damage results for candidate spacecraft electronics. In: IEEE RADIATION EFFECTS DATA WORKSHOP, 2002.

[10] DUPONT, E.; NICOLAIDIS, M.; ROHR, P. Embedded robustness IPs for transient-error-free ICs. IEEE Design & Test of Computers, New York, v.19, n.3, p. 54-68, May-June 2002.

[11] NORMAND, E. Correlation of in-flight neutron dosimeter and SEU measurements with atmospheric neutron model. IEEE Transactions on Nuclear Science, New York, v.48, n.6, p. 1996-2003, Dec. 2001.

[12] JOHNSTON, A. Scaling and Technology Issues for Soft Error Rates. In: RESEARCH CONFERENCE ON RELIABILITY, 4., 2000.

[13] LABEL, K. et al. A roadmap for NASA's radiation effects research in emerging microelectronics and photonics. In: IEEE AEROSPACE CONFERENCE, 2000.

[14] NORMAND, E.; BAKER, T. J. Altitude and latitude variations in avionics SEU and atmospheric neutron flux. IEEE Transactions on Nuclear Science, New York, v.40, n.6, p. 1484-1490, Dec. 1993.

[15] NORMAND, E. Single event upset at ground level. IEEE Transactions on Nuclear Science, New York, v.43, n.6, p. 2742-2750, Dec. 1996.

[16] BAUMANN, R. Soft errors in advanced semiconductor devices-part I: the three radiation sources. IEEE Transactions on Device and Materials Reliability, New York, v.1, n.1, p. 17-22, Mar. 2001.

[17] BOREL, J.; GAUTIER, J.; GASIOT, J. Silicon Redemption. In: EUROPEAN CONFERENCE ON RADIATION AND ITS EFFECTS ON COMPONENTS AND SYSTEMS, RADECS, 2001.

[18] ANGHEL, A., ALEXANDRESCU, D., NICOLAIDIS, M., "Evaluation of a Soft Error Tolerance Technique based on Time and/or Hardware Redundancy," Proc. of IEEE Integrated Circuits and Systems Design (SBCCI), Sept. 2000, pp. 37-242.

[19] ANGHEL, L., NICOLAIDIS, M., "Cost Reduction and Evaluation of a Temporary Faults Detecting Technique," Proc. 2000 Design Automation and Test in Europe Conference (DATE 00), ACM Press, New York, 2000, pp. 591-598.

[20] DUPONT, D., NICOLAIDIS, M., ROHR, P., "Embedded Robustness IPs for Transient-Error-Free ICs", IEEE Design and Test of Computers, May-June, 2002, pp. 56-70.

[21] CRAIN, S. et al. Analog and digital single-event effects experiments in space. IEEE Transactions on Nuclear Science, New York, v.48, n.6, Dec. 2001.

[22] ALEXANDRESCU, D.; ANGHEL, L.; NICOLAIDIS, M. New methods for evaluating the impact of single event transients in VDSM ICs. In: IEEE INTERNATIONAL SYMPOSIUM ON DEFECT AND FAULT TOLERANCE IN VLSI SYSTEMS WORKSHOP, DFT, 17., 2002.

[23] LEAVY, J. et al. Upset due to a single particle caused propagated transient in a bulk CMOS microprocessor. IEEE Transactions on Nuclear Science, New York, v.38, n.6, p. 1493-1499, Dec. 1991.

[24] HASS, J. et al. Mitigating Single Event Upsets From Combinational Logic. In: NASA SYMPOSIUM ON VLSI DESIGN, 7., 1998.

[25] HASS, J. Probabilistic Estimates of Upset Caused by Single Event Transients. In: NASA SYMPOSIUM ON VLSI DESIGN, 8., 1999.

[26] NICOLAIDIS, M.; PEREZ, R. Measuring the width of transient pulses induced by radiation. In: IEEE INTERNATIONAL RELIABILITY PHYSICS SYMPOSIUM, 2003.

[27] DODD, P. E.; MASSENGILL, L. W. Basic Mechanism and Modeling of Single-Event Upset in Digital Microelectronics, IEEE Transaction on Nuclear Science, vol. 50, pp. 583-602, June 2003.

[28] ZOUTENDYK, J.; EDMONDS, L.; SMITH, L. Characterization of multiple-bit errors from single-ion tracks in integrated circuits. IEEE Transactions on Nuclear Science, New York, v.36, n.6, p. 2267-2274, Dec. 1989 .

[29] REED, R. A. et al. Heavy ion and proton-induced single event multiple upset. IEEE Transactions on Nuclear Science, New York, v.44, n.6, p. 2224-2229, Dec. 1997.

[30] VELAZCO, R.; CHEYNET, P.; ECOFFET, R. Effects of radiation on digital architectures: one year results from a satellite experiment. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 1999.

[31] D'ANGELO S. et al, "Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems", Int. Symposium on Defect and Fault Tolerance in VLSI Systems, p 233-40, 1998.

[32] D'ANGELO S. et al, "Transient and permanent fault diagnosis for FPGA-based TMR systems", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, p 330-8, 1999.

[33] MOJOLI G.A. et al, "KITE: A behavioural approach to fault-tolerance in FPGA-based systems", International Workshop on Defect and Fault Tolerance in VLSI Systems, p 327-334, 1996.

[34] CARMICHAEL C., "Triple Module Redundancy Design Techniques for Virtex FPGAs", Xilinx Application Note XAPP197, 2006.

[35] KASTENSMIDT F. L. et al, "Designing fault tolerant systems into SRAM-based FPGAs", Design Automation Conference, p 50-655, 2003.

[36] DEMARA R.F. et al, "Autonomous FPGA Fault Handling through Competitive Runtime Reconfiguration", ASA/DoD Conference of Evolution Hardware, 2005.

[37] ALDERIGHI M. et al, "A fault-tolerant FPGA-based multi-stage interconnection network for space applications", IEEE nt. Workshop on Electronic Design, Test and Applications, p. 302-6, 2002.

[38] DURAND S. et al, "FPGA with self-repair capabilities", Int. Workshop on Field Programmable Gate Arrays, p.1-6, 994. M. Alderighi et al, "A fault-tolerant FPGA-based multi-stage interconnection network for space applications", IEEE nt. Workshop on Electronic Design, Test and Applications, p. 302-6, 2002.

[39] REBAUDENGO, M.; REORDA, M. S.; VIOLANTE, M. Simulation-based Analysis of SEU effects of SRAM-based FPGAs. In: INTERNATIONAL WORKSHOP ON FIELDPROGRAMMABLE LOGIC AND APPLICATIONS, FPL, 2002.

[40] CAFFREY, M.; GRAHAM, P.; JOHNSON, E. Single Event Upset in SRAM FPGAs. In: MILITARY AND AEROSPACE APPLICATIONS OF PROGRAMMABLE LOGIC CONFERENCE, MAPLD, 2002.

[41] BERNARDI, P.; REORDA, M. S.; STERPONE, L.; VIOLANTE, M. On the evaluation of SEUs sensitiveness in SRAM-based FPGAs, 10th IEEE International On-line Testing Symposium, 2004. pp. 115-120.

[42] ALFKE, P.; PADOVANI, R. Radiation Tolerance on High-Density FPGAs. San Jose, USA: Xilinx, 1998.

[43] LUM, G.; MARTIN, L. Single Event Effects Testing of Xilinx FPGAs. San Jose, USA: Xilinx, 1998.

[44] FULLER, E. et al. Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex® FPGA for Space Re-configurable Computing. In: IEEE NUCLEAR SPACE RADIATION EFFECTS CONFERENCE, NSREC, 2000.

[45] FULLER, E. et al. Radiation test results of the Virtex ® FPGA and ZBT SRAM for Space Based Reconfigurable Computing. In: INTERNATIONAL CONFERENCE ON MILITARY AND AEROSPACE APPLICATIONS OF PROGRAMMABLE LOGIC DEVICES, MAPLD, 2002.

[46] STURESSON, F.; MAUSSON, S.; CARMICHAEL, C.; HARBOE -SORENSEN, R. Heavy ion characterization of SEU mitigation methods for the Virtex ® FPGA. In: EUROPEAN CONFERENCE ON RADIATION AND ITS EFFECTS ON COMPONENTS AND SYSTEMS, RADECS, 2001.

[47] OHLSSON, M.; DYREKLEV, P.; JOHANSSON, K.; ALFKE, P. Neutron Single Event Upsets in SRAM based FPGAs. In: IEEE NUCLEAR SPACE RADIATION EFFECTS CONFERENCE, NSREC, 1998.