POLITECNICO DI TORINO

Doctoral School

Department of Control and Computer Engineering – XXVII cycle

PhD Thesis

Dependable System Design for Reconfigurable Safety-Critical Applications

Anees Ullah

Tutor
Prof. Luca Sterpone

Coordinator of the PhD program
Prof. Matteo Sonza Reorda

February 2015

Dedicated to the prayers of my mother, the dream of my father who could not live to see this day, to the love of my wife and to my sweet children.

Preface

Scaling of transistor's channel length is entering the realm of atomic and molecular geometries making possible the design of powerful miniature sized computing device and enabling their omnipresence in every aspect of human life including safety-critical applications, for example, automotive, space and avionics and bio-medicine. However, these scaled nanoelectronic systems are increasingly vulnerable to transients and permanent faults posing severe threat to life and expensive equipment. Therefore, safety-critical applications should consider dependability from design, implementation, layout, fabrication to in-field operations. The evaluation of dependability of such systems is equally important.

State-of-the-art SRAM-based FPGAs are interesting devices because they are not only the early adopters of latest technology nodes making them vulnerable to all sorts of nanoelectronic faults but their reconfiguration properties have potential counter-measure applications in dependable system design. This dissertation is focused on the usage of reconfiguration for improving and evaluating the dependability of nanoelectronic systems. The main research problem pursued in this work is the effective mitigation of soft errors like Single Event Upsets (SEUs) and Multiple Bit Upsets (MBUs) in FPGA's configuration memory, optimizing the reconfiguration times for fault injection and fault removal and emulation of permanent faults.

To address these problems this work proposes solutions based on applying redundancy and reconfiguration at different levels of granularity leveraging the basic building blocks of FPGA's architecture and reconfiguration capabilities in an unconventional manner usually not supported by standard tools. The solutions presented effectively mitigate MBUs by using techniques of fine-grain and coarse-grain redundancy and reliability-oriented placement. Recovery times are optimized with fine-grain error detection, error localization and local repairing. The usage of carry-chain based error detectors promises very fast error detection times in orders of nanoseconds and has extremely low-area overhead. The programmable nature

of LUTs is exploited for permanent stuck-at fault emulation of custom ICs using controlled LUT-mapping resulting in significant speed up against traditional fault simulation times.

Acknowledgements

The memory of my arrival to Torino as a PhD candidate on one fine "rainy" day three years ago is still fresh. It was the first time for me to go outside my country. Everything back then looked quite challenging and unpredictable but here I am at the end of my formal studentship life. It was indeed unforgettable experience of my life that shaped my academic, social and intellectual capabilities. I would like to express my gratitude to all the people who I came across during this journey.

This accomplishment would have not been possible without the continuous support, interaction and mentor-ship of my supervisor Prof. Luca Sterpone. His energetic involvement with me through brainstorming sessions, discussions and research problems identification has been exemplary and inspiring. His abilities to recognize and polish his students has endowed me with self-confidence and trust. I would like to express my gratitude to Prof. Matteo Sonza Reorda for his active involvement during this course of time looking after my academic and professional growth and providing his valuable feedback at times when I needed them most. Everyone in the CAD and reliability group deserves acknowledgement for being supportive and friendly. I will always remember the parties that we had together which was an opportunity for me to become familiar with Italian culture and delicious foods especially "bagna cauda".

I deeply appreciate the interaction with Dr. Neil Steiner of Information Science Institute, University of Southern California, USA for his prompt response and insights regarding tools for open-source reconfiguration framework which I used and developed upon for the work presented in this dissertation.

This dream of mine would not have be realized without the financial support of Higher Education Commission (HEC), Pakistan. I am confident and hopeful that the strong Pakistani student community mostly supported by this project will play

a positive role in the establishment of a knowledge-based economy and its advancement.

I am greatly indebted to my family for keeping head during this time. It would have never been easy on them all the way long. Their prayers and steadfastness were the source of my motivation. The "sand" of times would have not passed so delightfully without the company of my friends who stood with me through thick and thin. Many thanks to Javed Iqbal, Ihtesham Ul Islam, Affaq Qamar and Ali Zahir. I really enjoyed the time spent with you and the places I traveled with you across Europe.

Looking at the future from here everything again seems unpredictable and challenging but equipped with a befitting training and experience I am hopeful for bright times ahead for me and my family.

Contents

List of Tables

List of Figures

Chapter 1

Toward Reconfigurable Computing Paradigm

The ubiquity of electronic systems in todays world owes their existence to the aggressive technology scaling of transistor channel lengths. This miniaturization towards the atomic and molecular dimensions has produced computationally powerful and high performance systems enabling the integration of computing systems in every imaginable aspect of human life. This astounding progress is the result of daunting and complex electronic systems design and implementation efforts by semiconductors chip manufacturing industry in the past decades. Since, the famous prediction by Intel's co-founder Gordon Moore, that number of transistor on chip will double every two years [1], the Integrated Circuit (IC) manufacturers used it as a driving goal for survival. This required great efforts in the design, implementation, layout, fabrication and verification efforts and technologies. Todays VLSI chips, by no exaggeration, are the most complex systems that humans have created.

The companies that were involved in chip manufacturing business in the early days of computing revolution had to design as well as fabricate them. These integrated device manufacturing (IDMs) companies required a huge capital to maintain its human resource as well as building and material expenditure. Furthermore, the technological scaling process which was vital for the survival in the chip industry meant that these IDMs would have to spend a good part of what they already earned in the previous two years to stay up-to-date. This perpetual cycle made it very hard for new companies to join the race and demanded a whole new model to cope with this challenge. In early 1980s, the introduction of fabless chip model by the pioneers of FPGAs helped to disintegrate the IDMs business model and accelerated the pace of chip manufacturing [2].

The reason that FPGAs established the fabless chip model lies in the regular and homogenous architecture. It consists of computing nodes in form of Configurable Logic Blocks (CLBs) connected together by flexible Interconnect architecture. These logic and routing blocks are repeated to build up an array supplemented by Input and Output blocks (IOBs) to connect to the outside world. For example, the first FPGA chip, XC2064 utilized 64 CLBs and 58 IOBs [2] and was realized in 2.5 μm CMOS process. It required 85,000 transistors and was already larger than most microprocessors produced at that time. The properties of regularity and homogeneity to a great extent helped the FPGA manufacturers to become the very early adopters of the latest technology nodes and helped with the market penetration over the past three decades. Today FPGA chips are devices that are produced with state-of-the-art technology nodes, for example, recently Xilinx Inc. has produced FPGAs with 16nm FinFET [3] while Alteras startix-10 line uses Intel 14 nm tri-gate technologies [4].

Another reason that FPGAs are early adopters of new technology nodes is their usage of SRAM cell technology [5]. SRAM based FPGAs do not require changes in fabrication process, therefore, they are amenable to the usage of new technology nodes. More importantly, the reliance on SRAM technology brings the reconfigurability which means that the same chip can be used many times for implementing different functionalities or changes to the existing functionality. This property made SRAM based FPGA attractive for rapid prototyping and was frequently used by custom IC manufacturers for this purpose. Reconfigurability and amenability of SRAM based FPGAs to dimensional scaling were the prime drivers for deciding the victory between two competing FPGA programming technologies, the anti-fuse and SRAMs in the early days. Today large market share in the industry is held by the SRAM based FPGAs.

On the other hand, SRAM cells are inherently sensitive to radiation effects in form of Soft Errors [6] [7]. This translates to faults in the FPGAs configuration memory which is responsible for the functionality of the circuits mapped on FPGAs [8]. Soft errors can flip the memory content of an SRAM cell which may be responsible for holding the configuration for a routing connection, a combinational logic function or an instantaneous flip-flop value [9]. Therefore, soft errors can induce transient or permanent faults in the FPGA configuration memory. The effects of these radiation on circuits based on nano-scale geometries can be catastrophic for the functionality of overall system and can result in huge losses in terms of life and capital. Although, FPGAs are more susceptible to radiation effects due to

dependence on SRAM technology, nonetheless, ASICs and microprocessors are no exception, therefore, proper analysis of radiation effects and mitigation techniques are vital for applications where radiation effects post a threat to safety [10].

Electronic systems utilized in applications where safety and reliability are a major concern are often called safety-critical systems. A failure in such systems means a loss in life, property and/or damage to environment, for example, space, avionics, automotive applications, biomedical applications, weapons and nuclear power plants [11]. The design and implementation of electron systems for safety-critical applications needs to consider dependability-oriented optimization goals and properties as outlined in the famous paper [12]. Although, dependability concepts encompasses a large number of attributes [12], the attributes considered in this dissertation are mainly related to reliability and availability. The reliability of a system is the ability to continues the correct services while the availability means the readiness to offer the correct service. The dependability threats that are considered in this work are soft errors produced by radiation effects and permanent faults produced during VLSI chip fabrication process.

These dependability threats and the corresponding mitigation techniques will be considered in this dissertation as applied to state-of-the-art FPGAs. The attractive properties of short time to market, huge logic densities, high performance and particularly reconfigurability made FPGAs a designers choice even in saftey-critical applications. Nowadays, SRAM-based FPGAs with proper systems design and mitigation techniques are used in a variety of safety-critical applications [13] [14] [15] [16]. Therefore, design techniques, tools and methodologies for dependability-oriented applications of FPGA based systems is an active research topic these days. This dissertation focuses on harnessing the powers of reconfiguration of modern SRAM based FPGAs for improving and evaluating the dependability of nano-electronic systems.

The reconfiguration capabilities of modern SRAM based FPGAs can enable systems to recover from the radiation induced upsets in the configuration memory. The fascinating technology of partial reconfiguration makes it possible to selectively reconfigure a portion of the design mapped on the FPGA while the rest of system is still running in real time [17]. This enabling technology primarily introduced for improving the area, power consumption and re-usability is very handy for recovery from faults. Combing spatial redundancy based techniques for concurrent error detection and correction using partial reconfiguration ensure reliable operation of the systems respecting real-time deadlines. Traditionally, vendor tools do not optimize

designs for reliability, therefore, design methodologies and tools are needed to effectively utilize the FPGA fabric and fully exploit partial reconfiguration.

With the dimensional scaling approaching the atomic and molecular dimensions, the physical barriers for etching circuits in silicon are also becoming more and more prominent. The photo-lithography driven chip fabrication process is suffering from manufacturing defects in form of stuck-at faults, stuck-open faults and bridging faults to name a few. To quantify this problem, yield of a fabrication process is an important metric to consider which is determined by fault simulation process. Fault simulation process injects faults in Circuit under Test (CUT) and applies the inputs to analyze the effects on the circuit's output. This process can be accomplished using the partial reconfiguration capabilities of modern SRAM-based FPGAs to speed up the fault simulation process due to at-hardware speed processing.

Therefore, this dissertation focuses on two main themes related to dependability of critical systems. The former is related to the improvement of fault tolerance and recovery time capabilities of circuits mapped on SRAM-based FPGAs against soft errors. The latter is related to the usage of FPGA platforms for reducing the simulation times required for testing of custom ICs against permanent faults during manufacturing. These objectives are achieved by exploiting the FPGA's primitive architectural resources and partial reconfiguration capabilities. The abundant and underused architectural elements are used for novel techniques to error detection, error localization, local repairing, fault injection and emulation in order to achieve improved reliability, availability and evaluation times. The standard FPGA CAD tools are not suitable for the achievements of these dependability-oriented goals and requires the development of custom and semi-custom CAD flows and tools. The following sections introduce in details the main problems undertaken in this work and summarizes the main contributions.

1.1 Mitigation of Soft Errors

Traditionally, Triple Modular Redanduncy (TMR) and Dual Modular Redundancy (DMR) are two widely used fault tolerance methodologies. TMR technique uses a voting mechanism among the three redundant copies to mask faults and offer a continued service until the accumulation of faults results in failure in more than one TMR copies. This requires a periodic scrubbing of the configuration memory to remove the accumulated faults. The DMR techniques use comparison mechanism between the copies to detect faulty conditions, therefore, no masking mechanism are

used. The adopted redundancy technique can be applied at a coarser or finer granularity of the design. Coarse-grain redundancy based approaches uses the voting or comparison at a modular-level of the design. This means that the individual copies of redundancy have larger logic inside them. This would translate to the increased probability that MBUs will affect the redundant copies at the same time resulting in un-detection or unmasking. Compared to the coarse-grain redundancy approaches, fine-grain redundancy would mean that the comparison or voting is applied at the smaller logic level and the individual redundant copies are of smaller size. This decreases the probability that more than on redundant copy is affected at the same time, therefore, it is a better methodology for dealing with MBUs.

If error detection and correction is the adopted methodology, as is possible with FPGAs, then, the error detection and recovery time needs to be considered. For coarse-grain approaches the error has to propagate through more logic to reach the output or point of detection compared to fine-grain approaches, therefore, the error detection time is larger for coarse-grain approaches. Similarly, the required time for the recovery from a failure will be large in the coarse-grain approaches as the individual copies are of large size. However, with fine-grain error detection approaches comes the added advantage of fine-grain diagnosis capabilities meaning that local repair procedure can be applied if the fault location is pinpointed. This can largely contribute to reducing the recovery time of system as the recovery times is the contribution of delay due to error detection and repairing procedure.

Although, fine-grain approaches are quite attractive from dealing with MBUs, error detection and recovery time, however, the challenge is how to handle the huge complexity of error detection logic for this method to become feasible for larger designs. Consider that the comparison or voting is applied at a gate-level then every two gates in the dual modular redundancy will generate an error flag signal. This would require a huge routing overhead and control logic mechanisms and would render the method useless. This work presents several novel contributions related to fine-grain techniques for error detection, error diagnosis and repairing when applied to traditional TMR and DMR techniques. The main techniques investigated and developed in this work for the mitigation of increased probability of Multiple Bit Upsets (MBUs) in the FPGA's configuration memory and reduction of recovery times are based on granularity of the employed redundancy and reconfiguration. These contributions are outlined in the following lines.

Fine-grain techniques for DMR

The underutilized carry-chain resources available in abundance in state-of-the-art FPGA's can be used for fine-grain error detection, error diagnosis and fast local repairing in self-repairing systems. A self-repairing system contains a static region and a dynamic region. The static region consists of microprocessor, memories and IOBs, namely, the resources that are always required for the correct operation of the system and are fixed while the dynamic region contains the Circuit Under Test (CUT). The static region is responsible the local repair procedure when error is detected and diagnosed in the dynamic region. The CUT is duplicated and placed in the dynamic region using carry-chains for error detection, flag convergence and diagnosis purposes whenever they are not used for arithmetic computations in the design. The availability of carry-chains for error detection purposes and the sizes of synthesized LUT in the dynamic regions leads to logical and physical division of design having different error detection capabilities. The large number of error flag signals generated due to fine-grain comparison of DMR LUTs in dynamic regions are aggregated to produce less number of flag signals along the FPGA's CLB column concatenating carry-chains. This greatly reducing the complexity of flag control and management. The utilization of carry-chains not only brings the feasibility of fine-grain error detection to large designs but it also offer a very fast error propagation times making fast error detection a reality. This also enables the fine-grain diagnosis of errors at the level of FPGA's configuration frames. Moreover, as a configuration frame is the smallest unit of reconfiguration, the techniques greatly improves the recovery times. As the carry-chain resources are only utilized by CAD tools for realization of fast arithmetic computations, therefore, custom tools were developed for insertion of error detection and flag convergence logic, packing of LUTs in neighborhood and controlling placement of different error detection regions.

Fine-grain techniques for TMR

Cross-Domain Errors (CDEs) are a type of MBUs that are produced when a charged particle hit the FPGA's configuration memory and flip more than one SRAM-cell belonging to two different TMR domains, thus, breaking the fault masking ability. The probability of such faults in today's huge density SRAM FPGAs due to close proximity of configuration memory cells are on the rise with each new generation of FPGAs. This problem is not taken care of by commercial FPGA's CAD tools, therefore, often TMR domains are placed on the FPGA array in a manner that the neighboring configuration cells belongs to different domains. Mixed placement of TMR domains not only makes TMR scheme vulnerable to CDEs but it also renders partial reconfiguration of domains impossible which is vital for optimizing the increasing scrubbing times with growing size of configuration memory. Utilization of fine-grain approaches and non-overlapping TMR domain placement

can reduce the number of CDEs and can bring down the recovery time for TMR circuits from full scrubbing to partial scrubbing of individual domains. Different types of logic circuits are investigated for error detection purposes by connecting them with majority voters in different structures. These error detectors generates a large number of comparator signals which are aggregated together utilizing carry-chains in CLB columns. This unconventional usage of carry-chains for flag aggregation and the non-overlapping domain placement is not supported by commercial CAD tools therefore custom tools were developed for accomplishing the goals.

1.2 Emulation of Permanent Faults

Photo-lithography driven VLSI chip fabrication technology is reaching its physical limits due to aggressive technology scaling towards nano-metric dimensions. These limitations manifest itself in form of manufacturing defects like stuck-at, stuck-open and bridge fault to name a few. For the fast moving Application Specific Integrated Circuits (ASIC) industry increasing the yield within stringent time-to-market requirements is essential. Fault simulation is a mandatory step to determine the yield of any VLSI chip fabrication process. However, the time required for fault simulation is prohibitively huge. One reason of lengthy simulation times is the adoption of software based mechanisms. Although they provide flexibility but are inherently slow due to sequential nature of computations. An alternative solution is to use hardware based fault emulation on state-of-the-art reconfigurable FPGAs which can greatly reduce the timing requirements. The challenge is the fault equivalence which is mandatory for a surety that every ASIC fault can be emulated with an FPGA configuration. This problem is addressed using a perseverance based approach while mapping a synthesized ASIC netlist on FPGA resources. The gates are clustered together into LUTs with a controlled mapping phase bypassing the standard CAD flow which lead to deterministic housing of gates to FPGA LUTs. The stuck-at faults in netlist cluster housed in a particular LUT are emulated by reconfiguration of the LUT. The exploitation of LUT resources for permanent fault injection, fault emulation and fault removal required the development of custom tools for controlled mapping, equivalent fault dictionary generation and test and binary format conversion of test patterns. The developed approach not only achieves the same fault coverage and significantly optimizes the fault emulation times but also avoids re-synthesis and re-implementation altogether.

Chapter 2

State-of-the-art

According to [12], dependability is defined as the ability of a system to avoid service failures that are more frequent or more severe than is acceptable. The concept of dependability requires the context of a **system**. Where a system refers to an entity that interacts with other entities (or systems) including hardware, software, humans and physical world in order to exhibit a certain **behavior** according to the a **specification**. The deviation from the the correct behavior for the user of the system is called a **failure**. This failure can be caused due to the deviation of system from the specification or because the specification are incomplete. The reason for the failure is the inability of the system to follow the correct sequence of its **states**. This malfunction is the system's state is referred to as an **error**. The induction of the error to the system output depends on whether the error propagates to the system outputs, therefore, it in not necessary for an error to always produce a failure. The physical reason of an error lies in the system or its interaction with environment and is often modeled with a **fault**. Thus, a fault is a hypothesized cause of an error. A fault that produces a failure is active otherwise dormant. A fault represents an abstracted view of a **defect** which is an unwanted anomaly between the specified system and implemented one. Often, different kind of defects occurs in the system which are modeled by appropriate **fault models**. The chain of dependability threats is that a defect results in a fault, which results in an error which leads to a failure. The period of time required for a fault to manifest itself as an error is called **fault latency** while the time for a fault to convert to an error is called **error latency**.

Today's nanoelectronic circuits, due to shrinking device dimensional and lower operating voltages and currents have high susceptibility to faults of different kind. Permanent faults are created by manufacturing limitation in today's photo-lithography

based chip fabrication technologies or due to aging effects in circuits [38]. Aging effects are wear-out mechanisms which include time-dependent degradation of operating characteristics of device [39]. Main source of aging effects that creates permanent faults are time-dependent dielectric breakdown, hot carrier injection, electromigration and Negative Temperature Bias Instability (NBTI) [40]. Particularly, NBTI reduces the static noise margin of SRAM-cells leading to the instability of SRAM-cells in the configuration memory [42]. The resulting permanent faults includes shorts, opens, timing-induced failures and cross-talk to name a few. Transients faults are the result of temporary environmental influences like power supply and interconnect noise, electromagnetic interference and electrostatic discharge. Electronic components and systems in space as well as on ground are also effected by different types of radiation sources. High energy protons and heavy ions are the prime source of radiations in space while neutrons and alpha particles effects electronic systems for terrestrial applications. Radiation sources have the potential to increase the charge distribution near the channel of transistors. This charge accumulation, if crosses, a certain critical charge value, which is quite likely with decreasing dimensional sizes, can cause an output glitch at the output of transistor resulting in recoverable (often called soft errors) and non-recoverable errors (often called hard errors) at the systems level. These effects are often termed as Single Event Effects and are the most critical failures in logic and memory devices today [41]. SEEs are further divided into soft errors and hard errors [10]. Soft errors are upsets to the device operation which have transient effect and may disappear with time or may require a memory re-write. Following are different kind of soft errors that exist in today's FPGA-based systems.

Single Event Transients are a voltage/current spike induced in the combinational path due to impact of high energy-particles with the device. The pulse width of this spike for sufficient duration can cause it to propagate through the circuit and possibly sampled by a memory element in which case it changes the state of the system.

Single Event Upsets are caused by the interaction of high-energy particles with storage element of a device which changes the memory content. Because SRAM-based FPGAs have a high density and larger size of SRAM-cell required for configuration memory, SEUs can effect not only the design flip-flop and latches but it also effect the configuration cells which defines the logic and routing of the mapped circuit. SEU effecting a single bit are called Single Bit Upsets (SBU) while SEUs effecting more than one bit are called Multiple Bit Upsets (MBUs). MBUs are common in latest technology FPGAs because of the growing size and density of SRAM-cells.

Single Event Functional Interrupts are disruptions to the normal operation of a device and usually are related to a control or communication mechanism crucial for the functionality of the circuit and typically requires a reconfiguration or reset or power cycle.

Hard errors causes lasting damage to the device and are described below

Single Event Latchup is created by high-energy particles when a low-impedance path is is induced between supply voltage and ground for CMOS devices resulting in high current able to damage the device. It may be removed by a power cycle of the device.

Single Event Burnout is caused by high-energy particle hitting the source of transistors resulting in forward biasing, consequently resulting in short-circuit and an avalanche effect. These errors are typically relevant for power MOSFETS, IGBTs and high-voltage diodes.

Single Event Gate Rapture is caused by heavy-ions resulting in the rupture of gate-oxide isolation between gate and channel.

Xilinx FPGAs are quite robust against SEB and SEGR effects and therefore are less of a concern [10]. Finally, the long time exposure of devices to radiation sources result in the transistor's threshold voltage to change. Consequently, the timing constraints are violated as the performance of transistors degrades and result in permanent failures. This effect is called the total ionizing dose (TID) of a device.

This dissertation specifically considers SEUs, MBUs and stuck-at faults models. Furthermore, the considered attribute of dependability in this work are reliability, safety and availability as defined in [12]. After discussing the dependability threats to nano-electronics systems in general and FPGA's in particular, system design and mitigation techniques are considered next. System design techniques are the means to attain dependability [12] and includes fault prevention, fault removal, fault forecasting and fault tolerance which are explained in more details in the following lines.

Fault prevention techniques eliminates or reduces the probability of faults in the system. These techniques are designed to counteract the procedures that creates the faults or the development of improved technological solutions to withstand faults. One example is the usage of rad-hard components in space applications. The downside is that such solutions are quite expensive from economical point of view.

Fault removal techniques detects, locates and removes faults in the system. The related methodologies can be applied statically as the design and specification phase or dynamically during the operation of the system. For example, formal verification detects, locates and corrects fault in the design and specification phases while FPGAs enable the dynamic detection, diagnosis and correction of faults.

Fault forecasting techniques estimates that a fault occurs and evolves to failures. The probabilistic estimates depends upon the fault distribution and propagation to the output and is an important tool to assess whether the required values of target dependability attributes are achievable.

Fault tolerance assumes that the faults may arise even if fault prevention and fault removal techniques are in place and aims at withstanding its effects while ensuring a certain level of reliability in the service. Fault tolerance techniques are based on different types of redundancy. Redundancy can be applied to information, time and space. The information redundancy refers to the usage of error detection and correction codes like hamming codes, SEC/DEC codes, turbo codes to name a few. The time redundancy means that the computations are executed more than one and the results are compared. Finally, the space redundancy means that the hardware is replicated more than once to simultaneously compute and compare. The hardware redundancy can be further divided into passive, active and hybrid. The passive redundancy means that the hardware has the capability to operate even in case of a fault because it can mask the effect of a fault. Usually, passive redundancy uses more than two redundant copies and votes them with a majority voter. The values that occurs in majority is allowed to pass to the user. Triple Modular Redundancy (TMR) is an example of passive redundancy. In active redundancy, faults are detected and corrected and usually requires two redundant copies but the operation of the system has to stop for the correction procedure to be carried out, for example, Duplication with Comparison (DWC) based approaches. The hybrid redundancy schemes combines the fault masking and fault correction capabilities and usually requires more hardware resources in form of spare unit.

The design of dependable systems is a process that may requires iteration of design and **assessment** until the desired levels of dependability attributes are attained. Dependability assessment can be achieved at the design phase or latter after the system is produced. The evaluation at design phase relies on models and may not accurately describe the physical defects but can help with early mitigation. While the evaluation carried out latter after the system production is carried out with

different kind of testing procedures. This latter type may require longer duration of times and is typically expensive but more accurately analyses the systems and the mitigation strategies deployed.

For chip manufacturers, the classification of faulty vs non-faulty chips is an important economic problem. As the complex electronic system design and fabrication process is far from fault free, a process known as fault simulation is mandatory. This procedure utilizes a model of the actual system, a fault model, a set of test inputs and determine the percentage of faults detected. This percentage of detected faults ratio to total faults is called the fault coverage and is an important metric for the dependability assessment of electronic systems. The process of fault simulation helps in the critical effects analysis of faults, diagnosis of faults and improvement of test input quality.

For after production, in field assessment of faults, mainly relevant permanent and transient faults, fault injection is an important alternative to actually inducing the fault from sources of faults like radiation, electromagnetic interference etc. The fault injection process applies a model of the actual fault in the Circuit Under Test (CUT), applies the appropriate inputs and analyses it effects. The ease and speed with which faults can be injected are important parameters for a given fault injection approach.

2.1 Fault Injection and Fault Emulation

In this section, we outline some of the most relevant techniques used for emulating ASIC faults on reconfigurable hardware based on FPGAs. These methods can be mainly classified according to the adopted fault injection methodology.

2.1.1 Circuit Instrumentation based approaches

Circuit Instrumentation approaches add extra hardware resources in the design for fault injection purposes. A dynamic fault injection approach is presented in [81] [82] which instruments the model of circuit with a global shift register controlling the activation of signals for fault injection. The activation signals selects between duplicated LUT representing faulty and fault-free functions. The faulty behavior is achieved by reconfiguration of LUT site designated to be faulty therefore no

recompilation is needed. Independent faults are identified and instrumented in such a way to inject multiple faults at the same time reducing the reconfiguration time. To avoid reconfiguration and for fast injection of faults the authors in [83] [84] included all the desired faults in the model and design and synthesized them necessitating the activation of control signals in run-time using a scan-chain based fault injector circuit. Instrumentation at the gate-level for fault injection and scan-chain based activation mechanisms is used by the authors in [85] enabling them to avoid time-consuming re-compilation steps for generating fault bit-streams. However, the size of fault list is directly proportional to the hardware complexity of the injector circuitry, introducing a bottleneck for large VLSI circuits. To attain more speed up for the whole process of fault injection, test patterns applications and faults classification, a system is proposed by authors in [86] [87] that exploits the idea of minimum communication between host and emulation platform. To summarize, techniques based on circuit instrumentation are intrusive in nature and in cases have large area overhead.

2.1.2 Reconfiguration based approaches

Reconfiguration based fault injectors modify the configuration bit-stream of the design to emulate faults. The authors in [88] utilize a commercial fault emulation platform that constrains the technology mapping of logic cones to LUTs and generates a corresponding bit-stream for each fault in the logic cone in form of FPGA reconfiguration list. The authors show that for designs with more than 100,000 gates hardware emulation is two times fast compared to software fault simulation. JBits based tool-flow [89] is used for directly changing the configuration bits of a CLB in order to inject faults by authors in [90] [91] [92]. However, JBits is no longer supported for state-of-the-art commercial FPGAs. A direct bit-stream manipulation based injection is used by the authors in [93] [94] to inject faults in LUTs without constraining technology mapping. They reduce the size of fault list by considering only the active inputs of LUTs. However, this technique does not guarantee that every ASIC fault will be covered. The fault injection for a wide variety of fault models was presented in [95] where the authors develop faulty bit-stream by changing the HDL models. This requires time-consuming re-compilation for each injection of the fault model. Another interesting work exploiting partial reconfiguration for fault injection is presented in [96]. The authors present a methodology for the correlation of stuck-at fault model with that of Single Event Upset (SEU) fault model. Improving and generalizing upon the methodology presented by authors in [88] this work develops a general framework for fault emulation on commercial FPGA platforms without resorting to expensive platforms and vendor tools.

2.2 Fault Tolerance Techniques for FPGAs

In FPGA community, the terminology of fault tolerance is loosely used to cover the fault masking as well as fault removal design mitigation techniques. The following subsections describes in more details the approaches.

2.2.1 Error Masking and Correction

Error masking techniques ensures the correct operation of a system while withstanding the effects of errors in the system. Spatial redundancy utilizing majority voting function among the redundant copies is the most commonly used masking approach termed as N-modular redundancy. The values of **n** is an odd number usually and is often taken to be **3**. This kind of 3-modular redundancy is called Triple Modular Redundancy (TMR). The overall system is operational until two out of three copies continues to work correctly.

Spatial redundancy can be applied at several **granularities** of the design as defined in [61] [62]. Mainly, the granularities of applications are coarse-grain and fine-grain. In a coarse-grain TMR scheme like Block TMR [61] or large-grain TMR [63], the redundancy is applied to larger chunks of logic design and are then voted by a majority voter. The issues with this approach is that the internal flip-flop's state can not be restored even after the system is reconfigured. Therefore, in order to store the state each flip flop must be feed with the corrected values produced by the voting mechanism. This kind of feedback structure with voting ensures that after reconfiguration the state of redundant copies will be synchronized after a few clock cycles. A scheme that uses feedback with voting for each flip flop is called local TMR [62]. LTMR protects against SEUs but as the combinational datapath is vulnerable to SETs, therefore, SETs with enough width can be sampled by user flip flop's and can be converted to SEUs. Moreover, the global routing and reset signals are not triplicated in LTMR and can become a single point of failure. To cope with this issue combinational datapath is also triplicated along with global clock and reset signals. This scheme is called Global TMR (GTMR) and is able to withstand SEUs and SETs. The difficulty lies in using three different clock domains due to the clock skew and synchronization problems. A relaxed scheme that do not triplicates clock and reset signals is called Distributed TMR (DTMR) [61]. The fine-grain TMR approaches encompases LTMR, GTMR and DTMR schemes.

Xilinx TMR (XTMR) is another fine-grain TMR approach that closely follows

the GTMR methadology but offers flexibility regarding applying the redundancy to I/O's, FF's and clock resources. Figure 2.1 shows the typical scenario of circuits implemented with Xilinx TMR tool [64]. It is possible to notice that the architecture consists of partitions and domains, where domains are the replicas of the original circuit while partitions are the division of the logic resources by the three voter elements. According to the data structure to be hardened, commercial redundancy tools (e.g., Xilinx TMR tool), handles throughput logic, state-machine logic, I/O logic and special features differently. For example, partition 2 as shown in figure 2.1, represents state-machine logic where the majority voter is inserted on the output of a state register that loops back to the combinational logic in the same domain. This arrangement ensures that an SEU will be automatically corrected by the inherent synchronization due to three voting structures in feedback path with the registers before the next upset occurs. The output logic in partition 1 in figure 2.1 shows a minority voting based structure to combine the output from the three redundant copies into a single output. In case of a fault in any domain, the corresponding minority voters will tri-state the corresponding output. This helps to avoid signal contention on the output package pins of the FPGA. The TMR architecture shown is figure 2.1 has only one signal group where a signal group refers to the triplet of majority voters that receive the same inputs signals. However, in real circuits each domain consists of multiple signal groups that cross the boundary form one partition to the next partition.

Authors in [65] implement the XTMR architecture on Virtex FPGA and test it with fault injection and radiations. They conclude that there were faults created by single bit flips that effects more that one redundant copy. Investigations showed that this faults occurs due to the placement of TMR domains in close proximity to each other such that they share the same routing matrix. With the growing device densities the probability that these Cross-Domain Errors (CDEs) will become significant rises as shown in [66]. This motivated researchers to develop methodologies to mitigate such faults [37] [67] [63]. The first approach presented in [37] decreases the chances of faults effecting more than one TMR domain by careful placement and routing avoiding the resources to be present in close proximity. The authors in [67] instead takes a different approach by introducing redundancy in routing paths which enhances and increasing the fault tolerance ability because even in the presence of faults a connection for the delivery of information is more probable to exist. The third approach presented in [63] is based upon converting the fine-grain TMR to a more larger-grain TMR by removing the internal majority voters introduced for each flip flip state restoration. However, the proposed methodology resorts to a much complex synchronization scheme which has only limited applications.

Works in [68] [69] investigates the trade-off that exists while resorting to different level of granularity with regard to application of redundancy. The investigations of these works shows that coarse-grain redundancy has low area-overhead compared to fine-grain redundancy approaches, however, fine-grain approaches makes it less likely two redundant copies are simultaneously effected.

Another aspects with which redundancy is applied is **diversity**. Diversity means that the individual redundant copies can have different implementation realizing the same functionality [70]. There may be difference in execution times of different implementations of Diversity TMR, therefore, it may require mechanisms for synchronizing the results from the redundant copies. Recently, the authors in [71] showed that DTMR increases the fault tolerance of circuits implemented on SRAM-based FPGAs by more than 36% compared to traditional TMR schemes.

In order to introduce the overhead of TMR scheme, the authors in [72] proposes a Partial TMR (PTMR) that **selectively** applies redundancy to critical elements of the design. The idea exploits the fact that not all configuration bits effects the implemented design. The effective bits were termed as the *sensitive bits* out of which a small portion of bit were identified as *persistent bits*. The *persistent bits* required reset operation and were not repairable by simple reconfiguration. The authors showed an adequate level of reliability with reduced overhead, however, the method's attractiveness reduces when applied to designs with large number of feedback paths.

Although, the techniques discussed above improves the masking capabilities of circuits implemented on SRAM-based FPGAs, the **accumulation** of faults in the configuration memory can, overtime, render the masking capabilities useless. Therefore, for the correction of faults in the memory correction procedures are compulsory. The most commonly known methodology is called scrubbing. TMR with blind scrubbing technique [73] writes back the whole bit-stream after a certain amount of time known as the scrub cycle. The scrub cycle is selected according to the SEU rate and is usually set to ten times the upset rate for the application and device technology at hand [73]. This technique has the advantage of avoiding the accumulation of Multiple Bit Upsets (MBUs) that may cause multiple bit flips in the configuration memory. However, the frequent scrubbing and the comparatively long duration of scrub cycle, exposes this method to increased probability of an SEU corrupting the configuration data during scrubbing. This can lead to catastrophic effects including the corruption of functionality and on-chip contention, which in turn causes high currents large enough to damage device subs-tram.

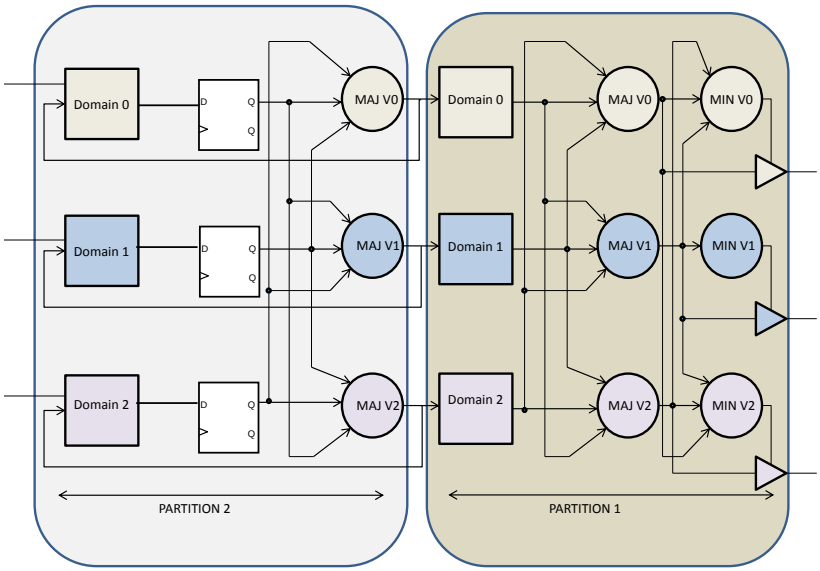

Figure 2.1. Xilinx TMR Architecture

2.2.2 Error Detection and Correction

The error detection and correction techniques identifies the existence of errors and removes them using reconfiguration. Following sub-sections categories the existing approaches.

2.2.2.1 Bitstream-level techniques

The correction of faults in the FPGA configuration memory requires the write-back of the golden version of the bitstream. In order to reduce the overhead of blind scrubbing [73], several other approaches based on manipulation of the bitstream are developed which involves some form of error detection and then correction. *Readback scrubbing* [74] is a process in which the configuration bitstream is readback and compared with a golden version of bitstream stored in an off-chip non-volatile memory. The write-back is only performed when there is a difference between the two bitstreams. Although, the readback scrubbing is more efficient than the blind scrubbing, still readback has to be performed continuously which is a time consuming process.

There are other techniques based on information redundancy added to the bitstream. Cyclic Redundancy Codes (CRC) are computed for each bitstream by the software tools and are stored as a part of bitstream. The FPGA configuration engine circuitry contains CRC checksum calculation during the configuration (writing or readback) process and is able to detect if the CRC codes computed by the software tools and the currently computed one matches [75]. This technique can only detect errors and can not locate it. Authors in [76] proposes a technique to use a CRC at the level of individual frame enabling them to identify the faulty frame and solely repair it.

Another methodology is to use Error Correcting Codes (ECC) techniques which has the ability to localize the faults in the bitstream frames. State-of-the-art FPGA's, like Virtex-5, contains a 12 bit ECC code embedded inside each of its 1312 bits high frame. These codes are based on hamming and parity codes usages and are often termed as Single Error Correction and Double Error Detection (SECDEC) codes. It is possible to combine these capabilities with partial reconfiguration for realization of frame-level error detection and correction functionality. The advantage of this technique is that the reconfiguration time is very low and the probability of SEU corrupting the configuration data is nullified. However, the configuration engine of the FPGA is unavailable for run-time dynamic partial reconfiguration which is required for adaptive hardware systems. Moreover, the continuous error detection and correction procedure requires energy. A scheduling based technique is proposed by the authors in [77] to deal with the aforementioned issue which introduces its own challenges in terms of implementation and synchronization and can end-up using more resources in hardware for achieving this task. Moreover, read-back scrubbing cannot detect MBUs in multiple frames which can cause functional faults of the system.

2.2.2.2 Circuit-level techniques

Detecting SEU effects by configuration read-back and/or comparison or error detection and correction codes removes every bit-flip from the FPGAs configuration memory. However, every bit is not significant and could not result in circuit level functional fault [74]. Therefore, circuit-level error detection, localization and correction can significantly reduce the overhead related to continuous/regular scrubbing. Error detection, localization and correction can be applied to a variety of redundancy techniques (like TMR and DMR) at various levels of granularity (coarse-grain vs fine-grain). Fine-grain error detection can accomplish the fast detection of fault,

fine-grain localization enable accurate and precise diagnosis of fault areas while fine-grain reconfiguration results in fast recovery time.

The advent of run-time partial reconfiguration and the growing device densities of SRAM-based FPGAs are motivating researchers to explore ways to combine TMR based masking techniques with partial reconfiguration. These approaches can reduces the recovery time compared to the full scrubbing of the bitstream. The authors in [78] [63] [79] proposed a methodology to selectively reconfigure a faulty domain, by combining large-grain TMR with partial reconfiguration. A scan-chain based error detection mechanism is adopted by [78] for TMR circuits, however, the method is applied at a larger granularity and the error detection time is directly proportional to the time for traversing sequentially through the scan-chain. This severely limits the scalibility of the approach. An error-detection logic embedded in the voters in [63] [79] enabled the identification of faulty domain. The effects of CDEs were minimized by removing the voters in the internal TMR partition [63]. This leads to synchronization issues after the faulty TMR partition is reconfigured. This is solved by predication based approach which can only be applied to small finite state machines. Another interesting but preliminary work in [80] proposed a dynamic domainlevel reconfigurable TMR architecture which has the abilities of error detection, localization, repairing and resynchronization. The authors adopt a high-level methodology for triplication and insertion of error detection logic at a coarse-grain level.

Error detection, localization and correction techniques can be applied to DMR systems. Authors in [69] investigates the usage of fine-grain duplication with comparison at the granularity of LUTs. The approach proposes to closely pack two duplicated LUTs and compare them with XOR gates. The probability that faults will effects the same comparing group is very low due to the fine-granularity used. Therefore, the proposed technique increase the reliability of circuits, however, it requires modifications in the FPGA fabric and is not directly applicable to commercially available FPGAs. This triggered research work in directions of utilizing the fine-grain approaches for reducing the recovery time for circuits mapped on commercial FPGAs [54]. The proposed approach realizes fine-grain duplication with comparison placing two duplicate LUT in a slice and compares them with carry-chain resources. As carry-chains can be extended along the column of CLBs, the proposed method reduces the generated check flag signals by aggregating them along the CLB column with carry-chains. The author also improves the routing reliability by special placement of duplicated LUTs avoiding the usage of switch matrix to effect two copies simultaneously. Furthermore, the author also presents local diagnostic and repair methods to enable fast repairing thus reducing recovery time. However,

the methodology cannot detect multiple bit errors in frames due its dependence on a chain of XOR and XNOR gates which acts like an odd parity circuit.

2.3 Limitations of current approaches

The presented techniques in section 2.2 and section 2.1 does not take the full advantages for the available architectural and reconfiguration capabilities of modern SRAM-based FPGAs for fault tolerance and fault emulations. The architectural primitive like LUTs, carry-chains, TIEOFF elements can be used in novel and unconventional ways to improve the dependability aspects of circuits mapped on SRAM-based FPGAs.

The abundance and availability of carry-chain resources can be put to different applications for improving the fault tolerance capability as pointed out by [54]. The carry-chains based circuits were only applied to detection of single bit errors in the configuration frames, however, the detection of multiple bit errors is left open to investigate. Moreover, carry-chains along-with error detection logic can be used to detect errors in TMR domains which can then be selectively reconfigured. However, this reconfiguration requires that the TMR domains should be placed separately and the resource should not overlap for partial reconfiguration requirements to be fulfilled. The current CAD tools for commercial FPGAs do not support such non-overlapping placement. The non-overlapping placement can simultaneously reduce CDEs and reconfiguration times.

The techniques presented in section 2.1 often required re-compilation and re-implementation which are slow and time-consuming processes. A techniques presented in [88] improves upon it by eliminating the need of re-compilation and re-implementation by exploiting the flexibility of LUT for emulation of stuck-at faults. However, the technique cannot be directly applied to state-of-the-art FPGAs. This fine-grain fault injection can potentially speed up the whole emulation process and can be extended to other fault models.

Chapter 3

FPGAs based Dependable Sytem Design

3.1 Field Programmable Gate Arrays

An FPGA is a pre-fabricated circuit consisting of Input/Output ports, programmable logic and programmable routing resources that enables them to realize any logic function just by downloading a configuration bitstream. With latest generation of FPGAs, it is possible to change the functionality of a portion of the running design while the rest of system is still running opening up doors to many applications not realizable before. Sophisticated set of CAD tools designed for optimizing different goals usually related to performance and cost are used for mapping user design on FPGAs. While dependability and reliability is becoming a growing concern for some users, to most users the traditional goals of performance and cost are more important, therefore, vendors tools and support are often limited and leaves room for research and development. In order to fully exploit the potential of these devices in dependability applications it is important to deeply understand the architectures, design techniques and reconfiguration details. The following sections presents the architecture and configuration memory layout of state-of-the-art commercial FPGAs. This knowledge is particularly relevant for identifying the main reason of different type of radiation effects and provides insights of how to exploit the FPGA's architecture in cases where commercial tools fail to provide the required capabilities.

3.1.1 Architecture

Fundamentally, an FPGA contains three main components: Logic Blocks (LB), Input and Output Blocks (IOBs) and programmable routing interconnect network. The LBs are responsible for the realization of combinational and sequential logic while the routing network connects them together. The interface to the outside world is provided by the IOBs. Furthermore, the programmable routing interconnect network is composed of switch boxes, connection boxes and pre-fabricated routing network. Commercial FPGA devices can be classified into three groups based upon the routing networks utilized: Xilinx and Lucent use Island Style, Actel FPGAs are row-based while Alteras FPGAs are hierarchical and Island Style. A more detailed explanation of the routing architectures can be found in [18] [19]. As the methodologies and techniques developed in this dissertation are applicable to Xilinx FPGAs only, therefore, Island-Style routing model will be considered. A simplified Island-Style FPGA architecture [20] is shown in figure 3.1. The Connect Block provides In and Out connectivity to LBs while the switch box provides connectivity to global routing resources. The following subsections provides details about the architectural component of Island-Style model as applied to Virtex-5 FPGAs.

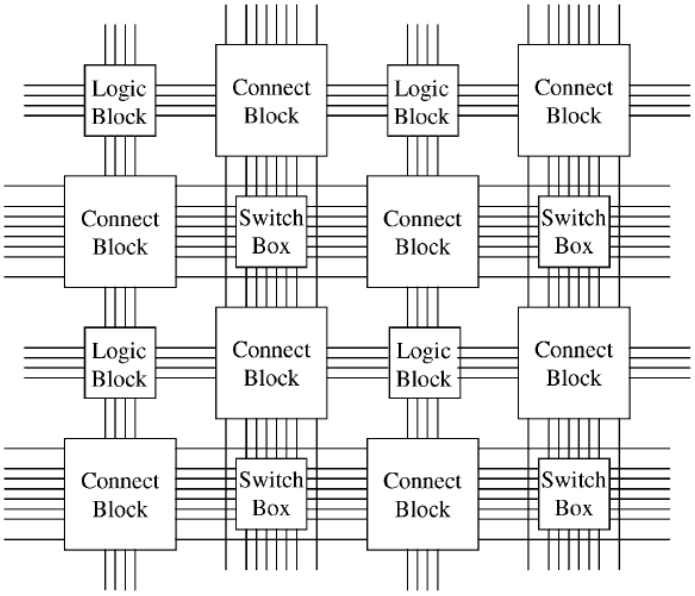

Figure 3.1. Island-Style FPGA architecture

3.1.1.1 Configurable Logic Blocks

The Configurable Logic Block (CLB) is the main resource for implementing combinational and sequential logic. In a Virtex-5 FPGA, a CLB is composed of two slices often represented with even and odd coordinates as shown magnified view in figure 3.2. Moreover, a slice contains a number of Basic Logic Elements (BLEs) depending

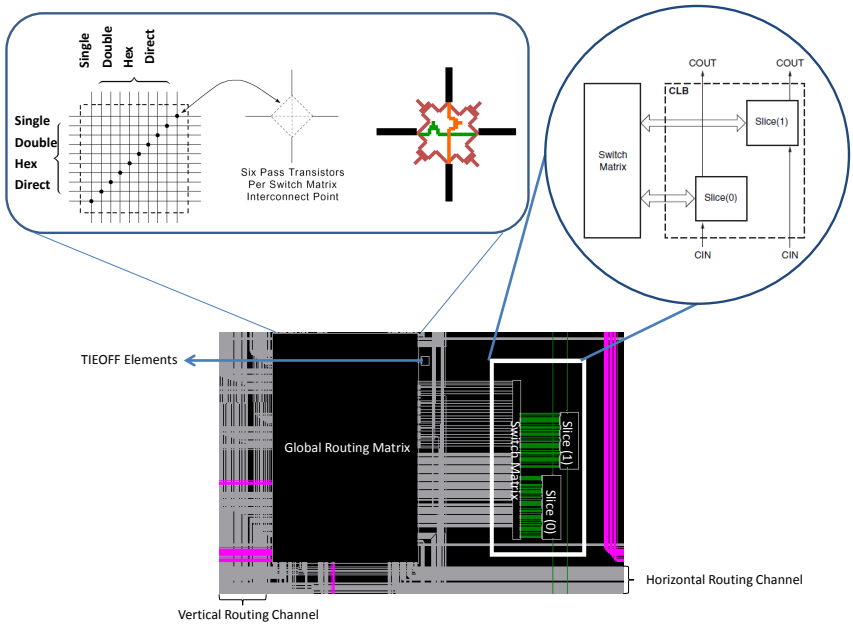

Figure 3.2. Virtex-5 Slice and Interconnect Tile

upon the architecture and family, figure 3.3 shows a BLE for Virtex-5 FPGA.

It can be noted that a BLE contains Look Up Table (LUTs) which acts as function generators, some control and carry-logic elements and sequential logic in form of storage elements which can act as flip-flops or as latches. In Virtex-5 architecture, four BLEs come together to form a slice as shown in figure 3.4.

The even and odd slices have no direct connections while the slices in an even/odd position along the column can be connected through dedicated wires of the carry-chain elements. This is very important feature for realization of fast arithmetic computation and a feature that is thoroughly exploited for dependability-oriented applications in this dissertation as will become clear in the next chapters. The 6-input LUTs are an important feature of these FPGAs which has an impact on efficient resource utilization. The architecture of 6-input LUT is that of a fracturable LUT [21] which can act as two 5-input LUTs or as one 6-input LUT as shown in

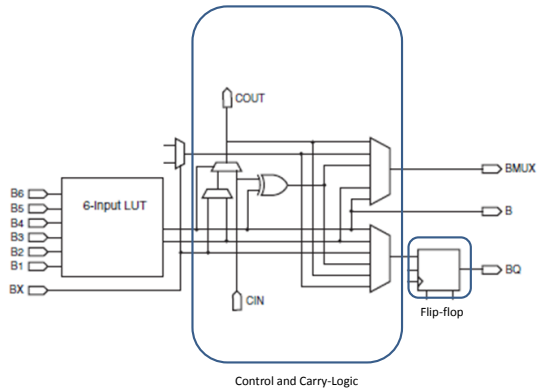

Figure 3.3. Virtex-5 Basic Logic Element

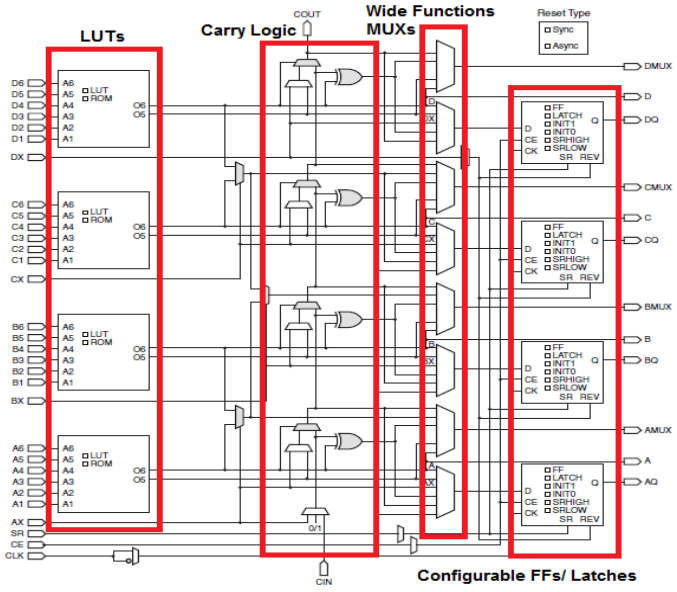

Figure 3.4. Virtex-5 Slice Internals

figure 3.5. The fracturable nature of LUTs have alot of applications for dependability oriented designs given that the synthesis and packing are properly controlled [22] [23]. In order to understand the mechanics of how combinational logic functions are realized with LUTs, consider the example shown in figure 3.6. It can be noted that the truth table output is stored in SRAM cells which are selected by a tree of multiplexers controlled by the LUT address lines i-e I2, I1, and I0. The circuit on the left uses wires a, b and c as inputs. In this case, wire *a* is binded to *I2*, wire *b*

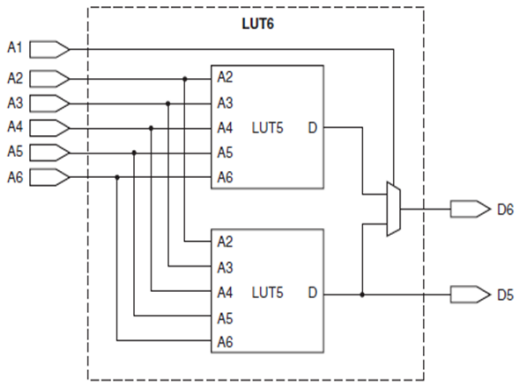

Figure 3.5. Virtex-5 Fracturable LUT

is binded to *I1* and wire *c* is binded to *I0*. This binding of functional input wires to LUT address lines has important consequences for the delay of mapped circuits, therefore, the router optimizes this binding to have optimal delay [24]. Moreover, this binding determines the LUT mask for configuration of the LUT functionality as shown in figure 3.6, the LUT mask *D5* will configure a 3-LUT in Xilinx world to the function under consideration.

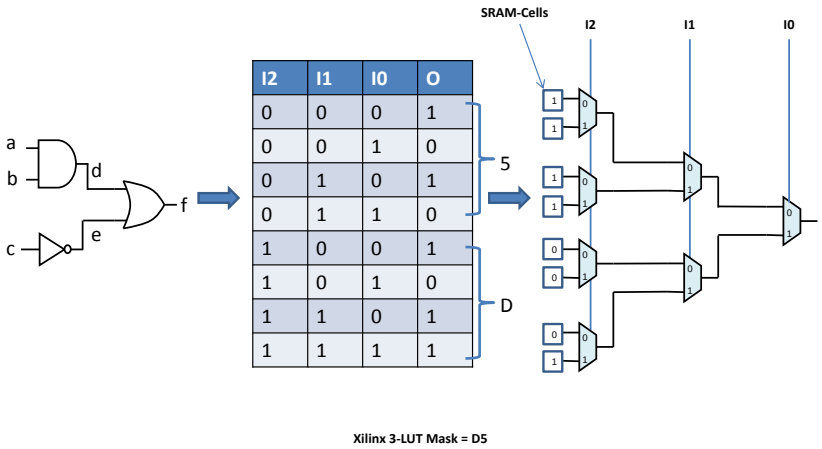

Figure 3.6. 3-LUT logic implementation

3.1.1.2 Interconnect Network

The routing network of FPGAs consists for pre-fabricated wires segments of varying lengths and programmable switches. These wire segments can be extended by configuring one or more switches, thus, multiple wiring segments can be connected in this manner to form routing tracks. The wires segments are organized in horizontal and vertical channels as shown in figure 3.2. The FPGA routing network should be flexible and efficient and is designed such that most of the wires are short while there are spare long wires as well. The abundance of short wires are there to fully exploit the principle of locality as most user designs requires short interconnects more frequently that long ones [18]. However, the long interconnects using fewer number of switches can greatly reduce routing area and delay [18]. The problem with the long wires is that it decreases the routing flexibility and consequently the design can become un-routable. The wires distribution in Virtex-5 can be discovered using the FPGA editor tool from Xilinx. In particular, the wires include bounce-across, double, pent, long and global lines. The bounce-across wires can connect a CLB directly to a neighboring CLB. The double wire can connect a CLB to the first and second neighboring CLBs. The pent wires are used to interconnect the second and fifth neighboring CLBs. The long lines are capable to connect sixth, thirteenth and twentieth neighboring CLBs. The global lines can connect CLBs from the first to twentieth neighbors. With respect to Virtex-4 routing architecture, hex lines which were used to connect the second and sixth neighbors are missing. Moreover, the accessible locations for a CLB forms a diagonally symmetric patterns providing connectivity in less hops and improving routing delay considerably [25]. The horizontal and vertical channels can are connected through a Global Routing Matrix (GRM) as shown in figure 3.2. GRM consists for a collection compound cross-point Programmable Interconnect Points (PIPs) as shown in the magnified view in figure 3.2. Each PIP is a pass transistor that can connect or disconnect two wire segments. It can be noted that the cross-point PIP uses six transistors and provides turns for horizontal to vertical wires and vice versa. GRM provides connectivity to the global routing network, locally, a Switch Matrix (SM) located near CLB provides connectivity to the CLB. SMs makes it possible to realize direct connections among the BLEs for a particular CLB and its immediate neighbor thus avoiding the global routing network which can have large routing delay. SMs are a collection of different kind of PIPs including break-point PIPs and multiplex PIPs [26]. These PIPs types provide direct, multiplexed and de-multiplexed connectivity for the CLBs. Each type of PIP switch is controlled by different number of SRAM-cells consequently the amount of configuration bits required for programming the routing interconnections in modern SRAM-based FPGAs is huge compared to the configuration bits required for implementing the logic functionality. This has important consequences

on the dependability aspects of circuits mapped on SRAM-based FPGAs making them extremely vulnerable to routing failures of different kinds as will be explained in details in sections 3.2.

3.1.2 Configuration Memory Organization

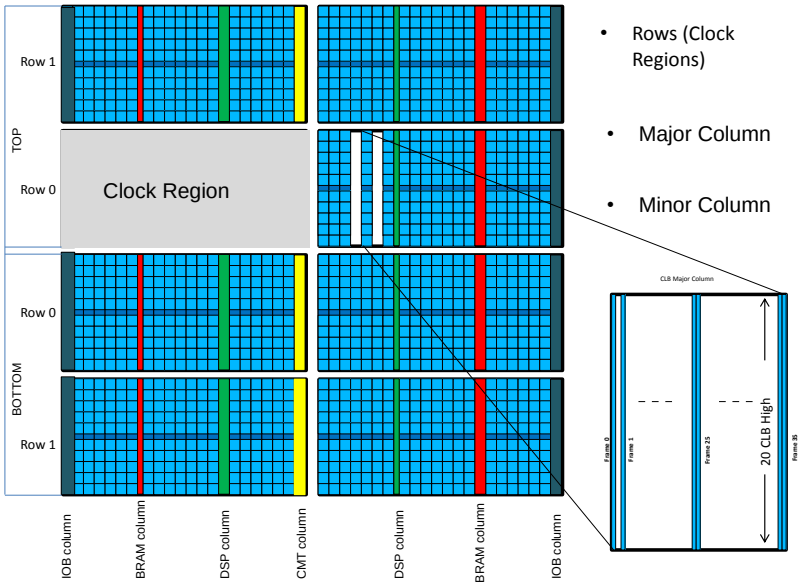

Figure 3.7. Architectural and Configuration Memory Layout of modern heterogeneous FPGAs

The reconfiguration capabilities of state-of-the-art FPGAs is a direct consequence of their configuration memory layout and architectures. Figure 3.7 shows a simplified view of the Virtex-5 FPGA architecture. Horizontally, FPGA's architecture is divided into two halves represented by TOP/BOTTOM which is further fragmented into a number of rows as illustrated by white separations in figure 3.7. The number of rows is architecture and FPGA family dependent. Each row signifies a clock region, which is an area of the chip able to be clocked from a local clock source derived from the global clock tree [27]. Vertically, the chip is divided into columns of resources of different types, for example, a column of IOBs, DSPs, BRAMs and CLBs to name a few. Each column of resource is called a Tile. In figure 3.7, the Interconnect Tile (INT) are not shown explicitly because in the configuration bitstream they are not separately addressed but are considered as a unit with the neighboring tile [28]. The configuration bitstream consists of header, dummy and synchronization data and packets [29] [30]. Header lets the configuration engine

know about the FPGA part, data and time of creation while the synchronization information is used to lock onto the sequence of words the controller recieves from configuration interface like JTAG. Packets acts as instructions to the configuration controller and write/reads to different configuration registers. An important register is the Frame Address Register (FAR) which enables access to configuration data of individual tiles. Each tile of resource can have different complexity and requires different amount of configuration frames [30]. The configuration bitstream is organized in frames which is the smallest unit of reconfiguration. The CLB tile requires 36 frames in Virtex-5 configuration bitstream as shown in the magnified view in figure 3.7. A frame is always 1-bit wide and 20 CLBs high in Virtex-5. A detailed description of a INT+CLB configuration frames are illustrated in figure 3.8. These frames are responsible for programming the interconnect tile and slices of a CLB. It can be noted that a single slice is reconfigured by using 4 consecutive frames and as each LUT requires 64 configuration bits each frame has 16 bits for a LUT in the slice. This detailed information is useful for the purposes of fine-grain reconfiguration for dependability-oriented applications. Each frame in the configuration bitstream is identified by a unique frame address. The frame address is 32 bit wide entity and contains several fields as explained in details in configuration user guide [30]. The frames that are responsible for the configuration of certain resources are a property of the location of chip area that is utilized by the resources mapped and is determined by the placement phase of CAD flow of commercial FPGAs.

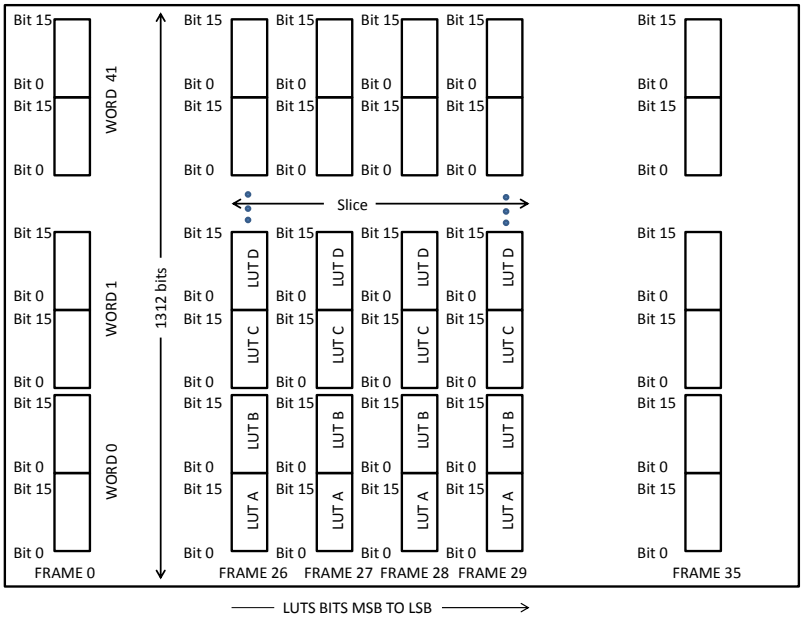

Figure 3.8. Configuration frames organization of INT and CLB tiles in Virtex 5

3.1.3 FPGA CAD Flows

The CAD tools are an important piece of software that automates and optimizes the design and implementation of circuits for a particular target technology. The design and implementation of VLSI circuits in today's world can not be accomplished without the use of sophisticated software tools. The flow starts by design entry using HDLs or schematically or more recently from a higher-level description like C/C++ (High level synthesis) [31]. The synthesis process is responsible for extracting the gate-level circuit for the realization of RTL level description of the design. This process is often called multi-level synthesis for modern VLSI circuits as the synthesized circuit are realized in multi-level logic as opposed to the two-level logic. This has important optimization with respect to area, delay and or both [32]. The multi-level representation is a general circuit often utilization gates that may not be available in target technology library. Therefore, the technology mapping step is responsible for converting the multi-level technology independent netlist into a technology dependent netlist. For FPGAs, the end product of this process is a LUT-level netlist. In Xilinx CAD flow these two steps are tightly integrated into a synthesis front-end process called Xilinx Synthesis Technology (XST). The LUT-level netlist is then passed through the packing phase which clusters the LUTs together in order to optimize for local routing and reduces the number of LUTs using fracturable LUTs and forms slices. This is also an important phase which has consequences on area and delay [19]. This phase, if controlled, can also enable the utilization of fracturable nature of LUT for dependability-oriented purposes [23] [22]. The next phase is placement which is the assignment of physical locations to the slice-level netlist on the chip. The physical location again has important consequences on area, delay and reliability [33] [34] [35]. These processes of packing and placement are integrated into a combine phase called mapping in the Xilinx terminologies. The next phase of routing is the optimization of routing interconnections utilizing the pre-fabricated and configurable routing network. This is often a timing consuming and difficult optimization process and one that has important consequences of speed, area, power and reliability of mapped circuits [36] [37]. The final phase of the flow is the generation of configuration bitstream for the final placed and routed design.

For utilization the capabilities of partial reconfiguration in run-time environment the CAD flow differs from the standard flow that is described before. In particular, the run-time partial reconfiguration puts its own design rules and physical constraints that must be satisfied in order to successfully utilize this technology [17]. The commercially available CAD flow is based on partitions-based partial reconfiguration required for systems which changes its functionality in the run-time. This kind of designs are characterized by the requirement that different hardware tasks should

be able to use the same area on the chip given that they are not used at the same time. Instead, this dissertation is focused on the localization of faulty area of the design and its correction at a fine-grain level, therefore, the commercially available partial reconfiguration CAD flows are not adequate for these purposes. Therefore, the reconfiguration approaches utilized in the upcoming chapters are not following the standard partial reconfiguration flows and proposes alternative methods.

3.2 Dependability on reconfigurable devices

This section describes the strengths and weaknesses of SRAM-based FPGAs technology that are relevant from dependability aspects.

3.2.1 Radiation effects on mapped circuits

For designs mapped on SRAM-based FPGAs, SEUs are the most widely studied and researched [43] [44] [45]. The SEU in the configuration memory can effect the CLBs of the design as well as the routing interconnections [37]. In case a CLB is affected target primitives can be LUTs, flip-flops, carry-chains and inputs/output muxes [46]. In the case a flip-flop is effected, the error induced alters the stored value in the flip flop, however, in the next cycle the flip flop samples a new values which is correct. The critically of this type of error depends upon the location of flip-flop in the design and the probability of it being sampled and reaching the primary output of the circuit. In case a LUT is affected, the logic functionality of the mapped circuit changes and the effect stays on until reconfiguration. The muxes inside CLB responsible for realization of wide-functions and feeding of carry-chains/flip-flops can also be affected in which case wrong signals may be propagated or signal path can break. The abundance of routing resources in SRAM-based FPGAs makes them particularly vulnerable to upsets in the routing and therefore different types of affects are possible [47]. An SEU in the GRM of the routing network can create short, open and antenna effects as presented by the authors in [48] and re-produced in this chapter as depicted in figure 3.9. An open is created if an SEU hits a PIP breaking the connection and resulting in dangling inputs to the CLBs or the outputs as shown in figure 3.9a. It is possible that an SEU affects the GRM in such away that a new connection is created between an output PIP and an input PIP while the connection to the old input PIP still exist. This creates a short effect as shown in 3.9d. Another type of routing effect is open/short or antenna effect in which case the old connection is deleted and a new one is created, however, the new one is not driven by any known logic values as shown in figure 3.9c. Short and antenna faults are multiple

effects of SEUs, often termed as Multiple Bit Upsets (MBUs), and are commonly created due to the usage of decoded-mux PIPs [37]. It has been shown that with dimensional scaling the probability of occurrence of MBUs is rising [49]. These routing failures have the potential to disrupt the normal operation of unprotected as well as protected designs utilizing redundancy based fault tolerance techniques like Duplication with Comparison (DWC) [50] and TMR [37] if proper placement and routing methodologies are not employed. Routing failures can also effect the design utilizing DPR thereby inducing faults between two Partially Reconfigurable Modules (PRMODs) severally handicap the static and dynamic regions of a reconfigurable systems [51].

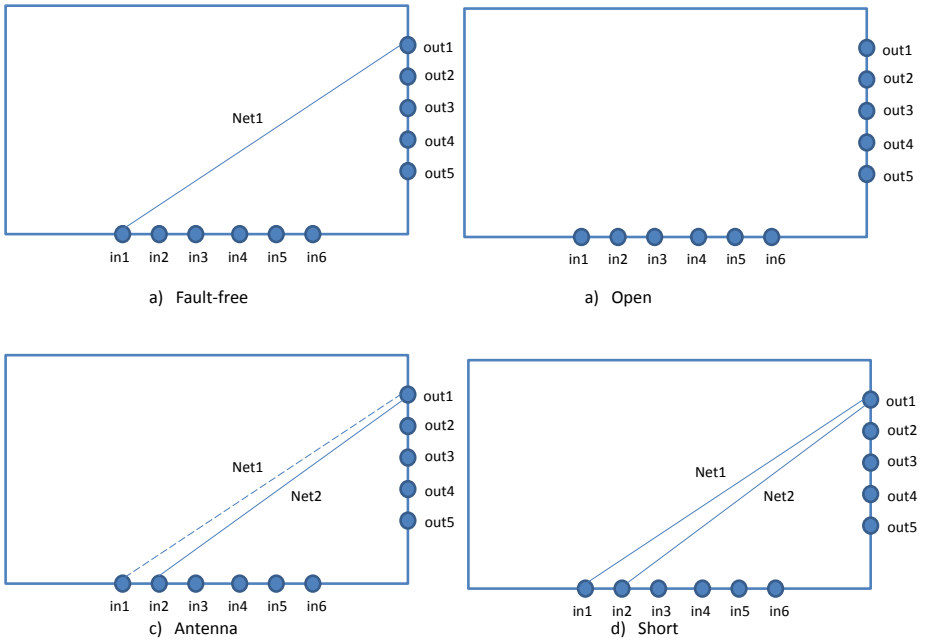

Figure 3.9. Single Event Effects on Routing

3.2.2 Reconfigurability

The vulnerability of SRAM-based FPGAs to radiation-induced upsets is a manifestation of the underlying programming technology i-e SRAM-cells. However, the same programming technology gifts them with reconfiguration capabilities which are increasingly becoming an important design tool for achieving flexibility, power, cost and reliability goals [52] [53]. For example in Software Defined Radio (SDR) the

same reconfigurable hardware platform can be used by swapping in and out modules for on-demand waveform processing according to user needs. Similarly, the SDR can communicate over a number of frequency ranges and communication standards thanks to reconfiguration. Partial reconfiguration has the potential to reduces the static and dynamic power components of the design [52]. As the focus of this dissertation is dependability therefore the aspects of reconfiguration applied to achieving dependability goals are interesting and one that has gained significant attention recently. The growing size and density of configuration memory in state-of-the-art SRAM-based FPGAs have important effects on the performance and dependability of implemented systems. Although, the growing configuration memory translates to powerful computation capabilities that the fabric offers but unfortunately it increases the probability of MBUs in the configuration memory [49] and increases the reconfiguration times [54]. To cope with the increased probability of MBUs, hardening techniques applied from the design phase to the final place and route phase are necessary along with reconfiguration. The reconfiguration times are increasing steadily with each new generation of FPGAs. This increase, on one side is due to the increasing size of configuration memory but on the other side is due to the unimproved speed of reconfiguration interfaces available on the fabric. For real-time fault tolerance capabilities the reconfiguration time is an important factor that determine if deadlines can be met. Fine-grain reconfiguration at the level of frames can be used to reduce the reconfiguration overhead in case of upsets in the configuration memory. Combined with fast error detection this can significantly reduces the recovery times of systems. However, fine-grain approaches to reconfiguration requires CAD support which is often not available with standard vendor tools therefore alternative and ad-hoc tools that can integrate with standard flows are necessary.

3.3 Tools for Open-source Reconfiguration

This dissertation is focused on the usage of FPGA's primitive resources in an unconventional manner to achieve dependability-oriented goals including fast error detection, fast reconfiguration and more control than offered by standard tools. Therefore, it is necessary to work with and develop custom ad-hoc tools and flows that can integrate with standard flows to achieve the desired ends. In the research community, there has been a serious effort since many years to come up with an open-source platforms that brings reconfigurable computing in reach of everyone [55] [56] [57] [58]. The Versatile Place and Route (VPR) has been the primary research tool for architectural exploration and CAD tools development encompassing technology mapping, placing and routing. This tool can work with generic FPGA architectural models and has been primarily used with in academic world and latter has been integrated

by Altera for its applications to commercial FPGAs. An effort to use VPR with Xilinx FPGAs has been recently proposed in Verilog to Routing project (VTR) [56]. Rapidsmith [57] and Torc [58] are two very powerful software platform that are able to work with commercial Xilinx FPGAs. They utilize the XDLRC description [59] to build up an accurate and precise architectural database upon which the tools offers several APIs for development of algorithms related to mapping, placing and routing. Both the projects are offering quite similar capabilities, however, Rapidsmith is based on JAVA while Torc is based on C++ and boost libraries [60]. This work utilizes the Torc framework extensively and develops upon it.

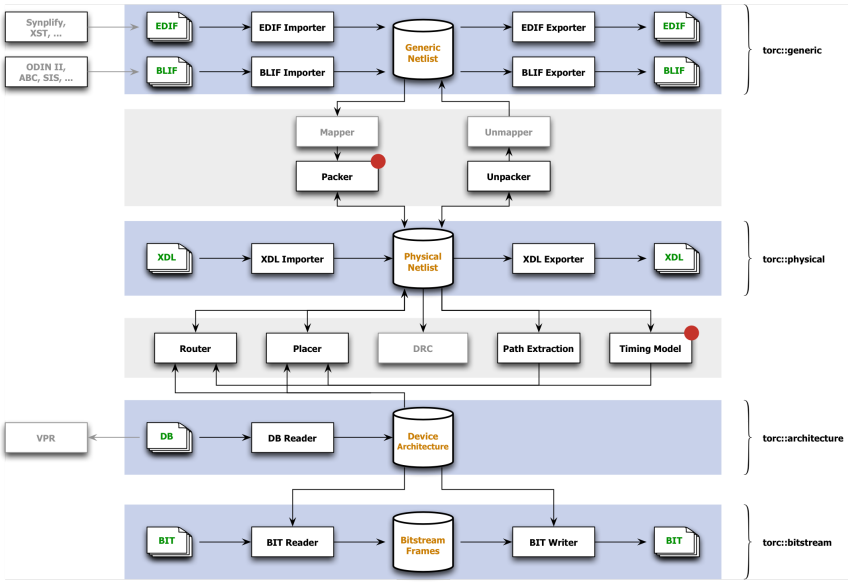

Figure 3.10. TORC structure

Figure 3.10 represents the design structure of Torc APIs. The generic API can read, modify and write netlists in Electronic Design Interchange Format (EDIF). This is a very important and useful interface which can be used to import designs synthesized with other synthesis tools, modify it according to user needs and then export it in a format compatible for FPGA tools to work with. For example, this work uses it for insertion of error detection and flag convergence logic at a post-synthesis phase and converting flat netlists to hierarchical netlists. Utilizing the generic API, users can work with unified generic object models and can transform and cluster the input graph nodes to FPGA primitive resources like LUTs, FFs and carry-chains. This phase denoted by mapper in figure 3.10 is currently incomplete, however, this work develops a limited version of this mapper that can convert gate-level synthesized netlist to LUT-level clustered netlist without duplicating gates

while mapping. This mapper is utilized for emulation of permanent faults of custom ICs on FPGAs. The physical API of Torc has the capability to read, modify and write slice-level Xilinx Design Language (XDL) format netlist. This is a powerful interface for back-end CAD tools development for placement and routing. This work utilized the physical APIs for error detection insertion and reliability-oriented placement. The architecture APIs pulls in the original wiring and resource information from non-proprietary vendor source files (XDLRC) and stores them in database accessible through several APIs. These APIs can be used to understand deeply the logic and routing architecture of commercial Xilinx FPGAs. Lastly, bitstream APIs works with the Xilinx bitstream format but to avoid breaching proprietary information, frame-level details are not supported. More detailed information of the usage and the algorithms developed upon the Torc framework will be presented in the next chapters.

Chapter 4

Fine-grain Error Detection in Self-repairing Systems

Reconfigurable systems are gaining an increasing interest in the domain of safety-critical applications, for example in space and avionic applications. In fact, the capability of reconfiguring the system during run-time execution and the high computational power of modern FPGAs makes these devices suitable for intensive data processing tasks. Moreover, such systems must also guarantee the abilities of self-awareness, self-diagnosis and self-repair in order to cope with errors due to the harsh conditions typically existing in some environments. In this work we propose a self-repairing method for partially and dynamically reconfigurable systems applied at a fine-grain granularity level. Our method is able to recover and correct errors using the run-time partial reconfiguration capabilities offered by modern SRAM-based FPGAs. Fault injection campaigns have been executed on a dynamically reconfigurable system embedding a number of benchmark circuits. Experimental results demonstrate that our method achieves full detection of single and multiple errors, while significantly improving the system availability with respect to traditional error detection and correction methods.

4.1 Self-repairing systems

A self-repairing system is presented in this chapter consisting of two regions: static and dynamic. The static region, also called base region, typically consist of a microprocessor, memory modules and input/output ports, as described in figure 4.1. In

general, these components are not re-configured and their full functionality is constantly required for implementing the correct operations of the system; for this reason the static region is often hardened using traditional redundancy-based approach, such as Triple Modular Redundancy (TMR). The static region is also responsible for the reconfiguration of the modules placed into the reconfigurable regions. On the contrary, the components in the dynamic region correspond to partially reconfigurable resources that can be configured in different ways depending on the system requirements. The main contribution of the this chapter is the development of an

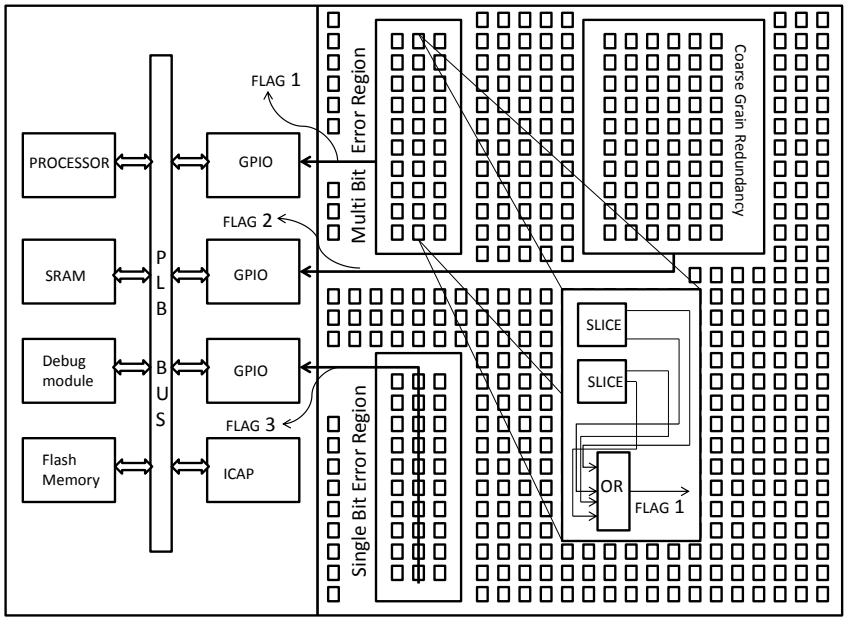

Figure 4.1. Placement space division into Static and Dynamic Region Reporting single bit error region, multiple bit error region and the coarse grain error region.

autonomous recovery approach of Partially Reconfigurable Modules (PRM) when errors are detected inside them. The approach is assigned to the static region providing effective capabilities of error detection and correction of faults within the dynamic region. Besides, our approach allows resilience to MEUs since we adopt a static region protected with fine-grain redundancy approach as described by authors in [97]. In detail, our solution provides a new fine-grain fault detection mechanism applied to FPGA resources, since the approach is based on the comparison of Look-Up Tables (LUTs) outputs by using the logic available to support carry propagation, which is generally used for fast arithmetic computations and mostly not inferred by design tools, following the approach preliminary introduced in [98] for fault detection; in this chapter we extend this idea so that the system is able to also correct

the identified errors by applying internal reconfiguration. In particular, the proposed method is characterized by the ability of detecting MEUs into the FPGA's configuration memory. Our solution is adoptable on all modern SRAM-based FPGAs equipped with Internal Configuration Access Port (ICAP) and that is based on LUT slice architecture. In order to practically prove the effectiveness of the approach, we developed a complete set of tools for the automatic generation of the constraints used for the partitioning of the dynamic regions. The developed set of tools directly acts at the physical level, automatically inserting a carry chain into the physical net-list and adding comparator check flags into the circuitry; moreover, the tool is able to cleverly place the different partitions of the dynamic region into proper sub-regions, thus allowing SEUs and MEUs correction. The proposed approach drastically improves the solution in [98] which uses the built-in slice carry chain for error detection only. In particular, the approach in [98] cannot deal with MEUs, while our approach supports the recovery of any number of faults in each dynamic partition. Our approach introduces a minimal area overhead, which is strictly dependent upon the number of user-defined partitions. On the average, the overhead introduced by our approach is around 11% with respect to duplication-based approach since the proposed technique is using far less computational resources if compared to the standard TMR solution. Furthermore, correction is performed on a single reconfigurable frame, which is the smallest amount of reconfigurable information that can be read or written; therefore, we can achieve the highest availability limits offered by the current reconfigurable technology.

4.2 The Overall Scheme

The proposed method consists of two flows: one applied to the dynamically reconfigurable region for implementing error detection, the other one for instrumenting the circuit mapped on the FPGA so that it supports the execution of the self-repairing method against single and multiple-bit errors. A dynamically reconfigurable system, from the architectural perspective, is partitioned into static and dynamic regions as illustrated in figure 4.1. The static region consists of a processor with static-RAM, general purpose IOs, flash memories, and hardware resources for managing the internal configuration access port connected to the processor local bus. The static region contains the main processor, which is in charge of controlling the partially reconfigurable system operational functionalities: therefore, it is very important to tolerate and recover errors in these modules. Hence, in this chapter we assume that this module is implemented using Triple Modular Redundancy. By suitably mapping the three copies of the circuit elements on the device the static region can be protected against any single point of failure. From the point of view of the circuit

architecture, the proposed method is based on the Duplication With Comparison (DWC) technique applied at two different levels of granularity, herein called Coarse-grained DWC (C-DWC) and Fine-grained DWC (F-DWC). The C-DWC is applied for slices that use the carry chain for computations such as fast additions or multiplications. In this case, the duplication is performed at the module level and the outputs are compared at the physical level by LUT elements configured to implement XOR combinational functions. Our approach is able to directly modify the circuit physical description in order to use the XOR logic function to compare the module's outputs. In case of error, the software tools running on the reconfigurable system partially rewrite the C-DWC region. F-DWC is applied at the place and route level, by suitably duplicating each LUT function in two copies that are placed in a single slice using two consecutive LUT positions. The outputs of the two LUTs are then compared with hardwired physical resources built into the slice in form of a carry chain by using internal and not programmable resources, such as hardwired MUXes and XORs. The outputs generated by the XOR functions are connected in a chain of OR logic functions in order to provide a single error detection flag for each column. Practically, the F-DWC approach can be adopted by acting at the Hardware Design Language (HDL) level: the combinational functions are duplicated and both copies of the circuit LUTs are placed in a single FPGA slice using two consecutive available LUT positions. Please note that the outputs of any pair of LUTs pass through the carry chain and at each pair position of the XOR generates a comparison signal called check flag. Since we are generating a check flag for each pair of LUTs the number of check flags may drastically increase. This means that a considerable amount of routing resources could be required by the implementation of these check flags because they have to be routed to the static region for the potential detection and correction of errors. Moreover, any such scheme will not only have a large overhead, but it will also be fruitless because the smallest unit of reconfiguration is a frame. In order to have a single check flag for each frame we propose to merge the individual check flags in two different ways. The check flags in the Single Bit Error (SBE) region are merged through the built in slice carry chain as shown in figure 4.2 (further details are provided in section 4.3.1). Furthermore, a whole column of slices are connected by the carry chains to produce a single flag for each column of slices (see for example flag 3 in the SBE region of figure 4.1). In this way, we achieve a huge reduction in the number of check flags, but we can only detect Single Event Upsets (SEUs) in the SBE region because multiple LUT pairs are connected together by a long chain of XORs and XNORs and thus even number of errors will go undetected due to the logical configuration of the detector. In the Multiple Bit Error (MBE) region each pair of LUTs generates a check flag and thus we have two check flags per slice. The number of check flags can be reduced by OR-ing some of the flags corresponding to the slices in the same slice column, as shown in the magnified MBE region in figure 4.3. Although some higher overhead is

introduced in this way, we have the ability to detect Multiple Event Upsets (MEUs) in the frames mapped on this region because compared to SBE region, the individual check flags are not merged along the carry chain passing through multiple XORs.

4.3 Error Detection Method

In order to fully explain our proposal, in this section we will specifically refer to the architecture of Xilinx Virtex-5 FPGAs. As described in the previous section, the error detection mechanism implemented in the reconfigurable region is based on LUT-based checkers and carry chains for propagating the check flags. Please note that the LUT checkers are only deployed when the carry chain is unavailable of comparison purposes. This allows reducing the performances degradation of the circuit implemented with our method, although in this case the detection mechanism is implemented at the modular level. In this section, we focus on the method adopted for the error detection using the carry chains for comparison; a more detailed explanation of both the LUT checkers and the carry chains insertion inside the physical place and route description of the circuit are given in section 4.5.

4.3.1 Single Bit Error Region

In order to detect single-bit errors, we propose to duplicate each original LUT function into two identical LUTs. Furthermore, we place the two LUTs in a single FPGA slice, where we set the Carry Input and the generic AX inputs to 1 and 0, respectively, as illustrated in figure 4.2. Consequently, the hardwired XORCY logic gate in the bottom of the slice is acting as an inverter, while the MUXCY multiplexer in the bottom first position is simply acting as a buffer to pass the value of LUT A. The multiplexer **M2** is receiving an inverted (through AMUX_2_BX hardwired connection) and buffered copy of LUT A output at its **0** and **1** inputs while the selection line is tied to LUT B (which is the copy of LUT A) thus effectively performing EX-NOR function. The XOR gate named **X2** receives LUT A and LUT B outputs on its inputs. Similarly, LUT C and LUT D can also be connected with such a scheme by extending the EX-NORs and EX-ORs along the slice. In fact, this scheme can be extended to an entire clock region covering 20 CLBs using the COUT and CIN of slices thus generating two flags for the even and odd slice columns of the same CLB. This convergence strategy can only be applied if the CLB column has no empty slices. In case the CLB column contains empty slices the dedicated COUT connection cannot be used to propagate flag signal upwards along the column. For such a case, a ORing LUT in introduced in the CLB column and placed in available

empty slice. This will be discussed in greater details in section 4.5.3. It is interesting to investigate an upper bound on the number of check flags that can be generated for the most complex design. The flag signal are generated per CLB tile columns and is directly related to the device rows and columns, for example, for Virtex-5 VLX110T the maximum number of check flags for any design cannot be greater than 160x8. As the FPGA must contain the control processor the actual number will be quite less than 1,280 and will determine the GPIO port size that are used by the controller to detect errors. Surprisingly, now it is possible to pinpoint single bit upsets in any of the four LUTs in any slice column in a clock region. However, errors in FFs cannot be directly detected instead they are detected when they propagate to the next logic levels of LUTs but we still cannot pinpoint the configuration frames required for correction and thus a larger area has to be reconfigured to deal with error in flip flops.

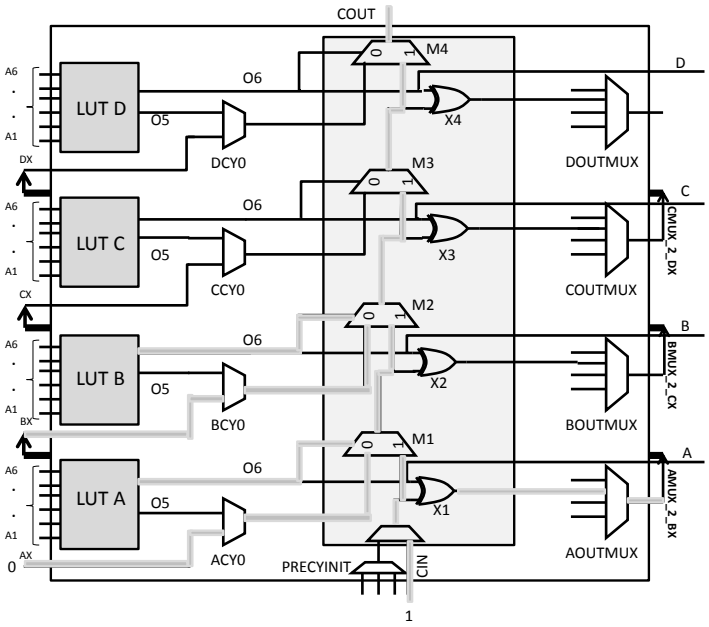

Figure 4.2. Single-Bit Error detection scheme implemented in a single slice

4.3.2 Multiple Bit Error Region

In case a further protection level is needed, our approach is able to provide detection against multiple-bit errors in a slice column. However, multiple bit errors can only be detected if the error detecting carry chain is inserted in a specific pattern that we will mention in this section. Furthermore, the logic connectivity pattern required

dictates that the LUTs used in this scenario can have 5 inputs or less. Therefore, all LUTs in design that have number of inputs less than 5 can be designated to this region. It is interesting to know that a recent survey of designs for Xilinx FPGAs suggests that 37% of LUTs inferred by the synthesizer are having inputs less than 5. The scheme implemented in order to achieve multiple-bit error detection is shown in figure 4.3, where the output of the LUT A is connected to the output O5, while the output of the LUT B is taken from O6. The two signals O5 and O6 are internally hardwired to the input of the XOR gate at the second LUT position, which acts as an error detection logic. Please note that we have to tie up the **A6** input of LUT A because we want to take output from the **O5** output. This is achieved by connecting it to the **TIEOFF** element. In this way every pair of LUTs in the design generates a check flag signal, as shown in figure 4.3 (check flag 1 and check flag 2). However, the number of flags grows linearly with the number of LUTs in the dynamic region, and so the method becomes unfeasible for large designs. In order to reduce the number of flags we propose the usage of 2 slices (out of the available 20) for merging the check flags by OR-ing them. In fact, two levels of OR gates placed in the bottom two slices of a slice column are used as a flag reduction strategy. As we are producing two flags for each clock region (one for odd and one for even slices) we can have a maximum of 72 LUTs (out of 80 LUTs in an even or odd slice column) configured for computations in any slice column location (even or odd) in a clock region. Thus the MBE regions require an overhead of 11.11% for flag reduction.

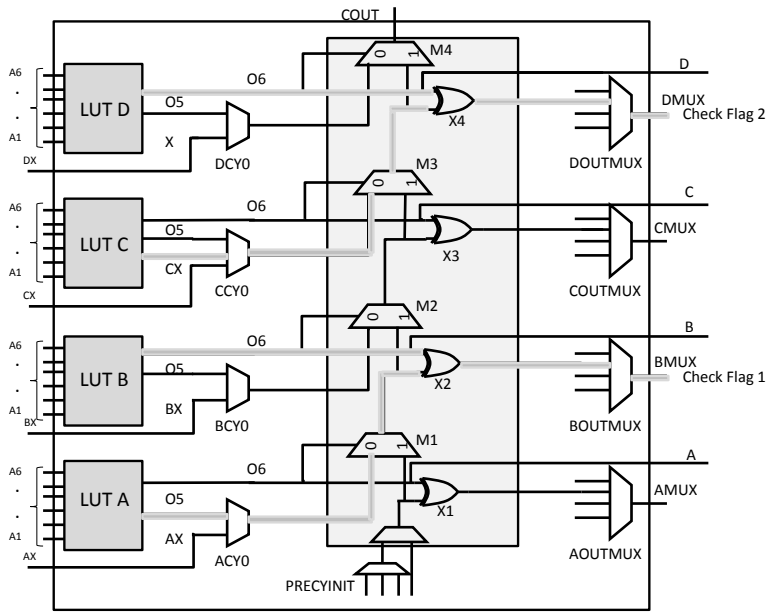

Figure 4.3. Multiple-Bit Error detection scheme implemented in a single slice

4.4 The Error Correction Method

The error correction method we propose is based on the assumption that the DUT Dynamic Regions are physically placed at positions preliminary known at the design time and that the original configuration memory frames are stored into an FPGA external memory in such a way that they can be retrieved when an error on the corresponding flag from the DUT region is signaled. The error signal can be used to drive the processor interrupt signal: in case of an interrupt from any flag, the main processor controller responds to the interrupt in the following way. First, it determines the clock region (in terms of CLB row) and the slice position (even or odd) the error was triggered by. This is particularly relevant, since the main processor controller needs to selectively reconfigure the faulty frames of the DUT design with the correct copies of the corresponding frames that are stored in the outside memory. Secondly, the clock enabling signals should be deactivated to disable the propagation of errors to the next stages in the design. This is possible since both static and dynamic regions have well-defined interfaces with clock enabling registers. Similarly, the DUT region should also have latched outputs at every stage of the design. Thirdly, the main processor controller accesses the configuration layer of the FPGA and reconfigures the faulty region with the golden copy of each frame. Lastly, the main processor controller enables the clock to re-start the normal operation in the DUT region involved in the correction.

4.5 CAD Flow

In this section we describe the tool flow we developed in order to insert fine-grain duplication with comparison using the built-in slice carry chains. The developed flow is shown in figure 4.4, and consists of two phases applied before MAP and after MAP process. The pre-map step generates a number of constraints for directed packing, placement and sites prohibitions while the post-map step inserts the error detecting carry chains and the convergence logic required to reduce the number of flag signals. This post-map modification is implemented by modifying the XDL file which is the Xilinx interface for interacting with the Xilinx CAD flow. This is a must for researches wanting to develop custom packing, placement and routing solutions. The tool flow has been developed as a C++-based software environment making heavy use of boost library [60] and Tools for Open Source Reconfiguration (TORC) [58].

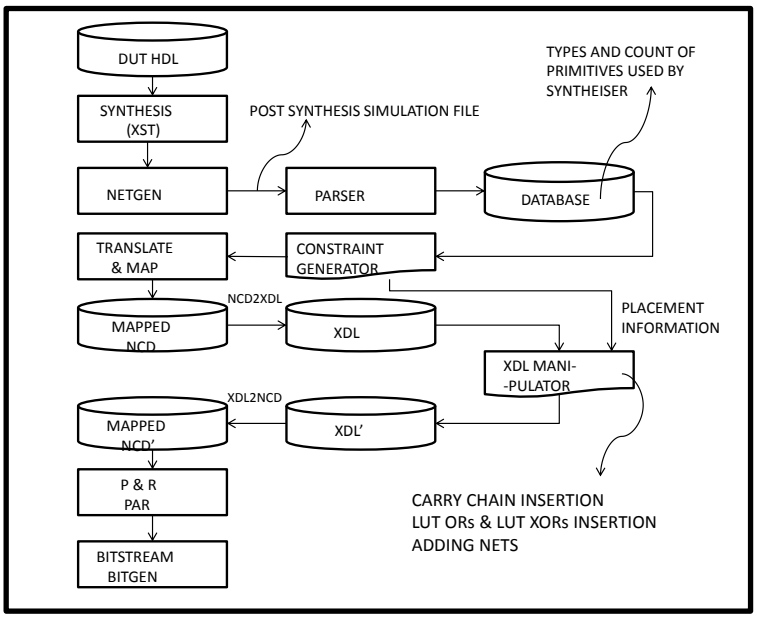

Figure 4.4. The developed design flow.

4.5.1 Netlist Extraction

The flow starts by parsing the net-list description of the circuit implemented into the dynamic region which was duplicated at the Hardware Description Level (HDL). It is important that both instances of the design should be labeled with **inst1** and **inst2** so that all the synthesized elements contains the hierarchical information of top level instance to which it belongs. However, global reset/clock signals are not duplicated at the module-level as will be explained in section 4.5.2. After synthesizing the design a post-synthesis simulation Verilog file is generated using the Xilinx NETGEN utility. The post-synthesis Verilog file contains the circuit net-list using the Xilinx primitive cell library elements. The parser module traverses the post-synthesis Verilog file and extracts the Net-list of the circuits in form of a graph. Each node of graph is modeled as a structure with a number of fields for example, functional string, instance name, inputs vector, outputs vector and type of primitive element (LUT or FF). When a new type of element is encountered on the current line a new node is dynamically created and all parameter of this node are populated by parsing the consequent lines. After all the nodes are properly initialized we iterated over the nodes of the graph picking up one node's output and finding it in other nodes inputs. Wherever, we find it we create a link thus the whole net-list is created in this manner.

4.5.2 Logical and Physical Partitioning of DUT

Once the circuit net-list is created in form of a graph, it is necessary to generate user constraints (UCF) in order to perform the DUT physical space division into regions and for packing the primitive cells into slices. The information obtained from the net-list is parsed in a hierarchical manner individuating the couple of LUTs that have been already duplicated at the HDL level. The hierarchical organization consists of three categories of components. Firstly, all the components which use MUXs and XORs are grouped together for modular duplication without carry chain usage. Secondly, LUTs with less than 5 inputs are grouped together to form multiple error regions. Thirdly, LUTs with 6 inputs are grouped to form single bit error detection regions. The division of flip-flops into groups is dependent upon the LUTs to which it connects and thus each LUT-FF pair is identified and stored in a way that each FF can be indexed by LUT. Similarly, a LUT-LUT pair is created that for each LUT used in the design locates the corresponding LUT in other instance. These LUT-LUT pairs are necessary for fine-grain comparison and need to be packed together in the same slice. A slice object is modeled to house the packing of LUTs and FFs. The slice object model is an abstraction of the actual slice on the corresponding FPGA architectures and considers the key attribute for example the number of LUTs, FFs and the unique clock and reset signals at the input of the slice. The clock and reset signals (along with Clock enable, Set/Reset signals) are important constraints for the application of our method because each slice can use a unique clock/reset signal that should be used by the FFs residing in the slice. That's why the global clock and reset signal were not duplicated due to the architectural limitation of state-of-the-art FPGA devices. Currently, the approach picks up a LUT-LUT pair and its corresponding FFs and checks the legality for packing them in a single slice object and packs them together if the legality constraints are fulfilled. The legality constraints are that they FFs should use the same clock/reset signal, although, the global clock/reset signal are unique, however, depending on the type of computation the circuit is implementing there can be local control signal used , for example, an FSM that will have different control signals for each DUT design instance. As in this work, the synthesizer is not constrained in any manner for the duplication performed at the HDL level; different duplication strategy applied at the synthesis level may allow the FSM to use unique control signals. Now that the graph is fully annotated with the resource division information a constraint file is generated that uses the Xilinx packing constraints (XBLKNM) for packing and for forming area groups (AREA_GROUP). It is important to note that each XBLKNM constraint uses a unique slice name by concatenating the region identifier with an identification number. For example, slices in single bit region uses name like **SBESlice1**, **SBESlice2** and so on. This naming convention is necessary for forming

the area groups and also for identification at the XDL level as will be discussed in section 4.5.3. After the completion of this packing step, each slice is included in an area group of either SBE region or MBE region depending on the slice name assigned to it during the packing process. Furthermore, each area group is floor-planned by selecting a slice range constraint to which it will be mapped. The numbers of slices required are calculated from the number of graph nodes that falls in a certain group. For multiple bit error regions, a number of CONFIG PROHIBIT constraints are generated for each slice tile. These constraints are generated in such manner that the top two slices of each tile column are left empty. It is important to note that both the packing and the floor-planning steps have not been optimized in our case and this can have considerable effects on the circuit operating frequency. Once the constraints are generated, the translation and mapping processes are executed and produce the mapping of LUTs copies in the same slice, as illustrated in figures 4.2 and 4.3. The routing and check flag signal insertion are performed in the following phase.

4.5.3 XDL-Level Manipulations

Once the mapping is performed, the insertion of the carry chain and the definition of the comparator resources are implemented by modifying the physical place and route description of the circuit in order to properly use the hardwired combinational gates. This process is executed in the following distinct steps. The carry chain insertion step is applied to the slice where carry chain primitives are not used for fast arithmetic computation. The insertion mechanism of the carry chain is the same for the single- and multiple-bit error detection scheme; however, they differ in multiplexer's settings and wiring details. Each inserted carry chain should have a different name, as the carry chains can be multiple slice long XDL uses two levels of identification to differentiate carry chains and its indivisible members. Each carry chain that spans multiple slices along a column forms a Relatively Placed Macro (RPM) identified by macro number and individual carry chain number for example Shape_0:0,0 and Shape_0:0,1 carry two carry chains that belong to RPM Shape_0 and each one identified by its position in the RPM chain. First, the XDL file should be checked to generate a unique RPM identification name for the next carry chain to be inserted. As for two different kinds of error detectors are used, single bit region and multiple bit regions, the flag reduction strategy differs for both cases. For single bit region, the flags are merged using the dedicated COUT line that run from a slice to the next slice. Each XDL instance that belong to single bit error region is augmented with a carry chain each one using a unique name as discussed previously. Once the carry chains are generated, multiplexer's settings for single bit error region is performed according to figure 4.2. Now that each single bit error

region slice contain carry chain for error detection it is necessary to connect the carry chains along the column to converge the error flags because at this stage the number of error detection flags are linearly proportional to the number of slices used in the single bit error region which is huge and its impractical to route it to the control processor. As the placement was constrained, in the previous step with a generated UCF file, however, the placer stills tries to optimize for timing and is not considering the fact the each slice column should be filled to the maximum extent possible resulting in a placement such that the single bit region slices columns have empty slices positioned in the middle of slices that uses carry chain based detectors. This is a serious problem for the flag convergence because the dedicated COUT line can only be used if the slices are positioned in consecutive positions above and below each other. In order to solve this problem, multiple carry chain detectors were combined using an OR LUT generating a single flag signal per CLB tile column. It is also interesting to note that for each OR LUT an automatic procedure searches for an empty slices in the same CLB column and picks up the nearest one in terms of the slice site distance for the OR LUT placement. OR LUT placement is a necessary step because for the Virtex-5 FPGA architecture the placement occurs as a part of the MAP process and should be completed before the PAR routes the design. In this way, the single bit error region flags are converged resulting in error detection carry chains of varying lengths. The maximum the length of a carry chain, the maximum will be the error detection latency as will be discussed in experimental results section. For multiple bit error regions, the carry chains used for error detection used OR LUTs in a manner similar to single bit region. However, the number of flags to be merged is quite large compared to the single bit region as each slice is generating two flags. Therefore, the slice sites that were prohibited from usage by the constraints in section 4.5.2 are utilized for the placement of OR LUTs. However, in some case it is possible that there are some empty slices that are not utilized by the placer and those can be used for OR LUT based on the metric of close proximity. The area overhead introduced by the OR LUTs in the design utilizes the post-placement empty slices and can be reduced if the slice tiles are filled to the maximum extent possible. However, this situation can cause the routing congestion and resultantly the routing time will increase considerably. The last phase of the low-level manipulation consists of inserting nets with the source and sinks added to them. Nets are added for all the components that have been previously inserted in form of carry chain based detectors or in form of OR LUTs. The routing implementation will be performed by Xilinx PAR tool that automatically routes all the nets between the inserted components and add the precise Pips that will be used for routing. It is important to note that if the placement is confined too much the router will be facing congestion problems and it is possible that the router may take a very long time or in the worst case will be unable to complete the routing of the design. Therefore, the placement should be such that an optimal balance among the usage of OR LUTs

for flag convergence and routing congestion is achieved.

4.6 Experimental Results

We implemented the proposed method on a Xilinx Virtex 5 LX110T SRAM-based FPGA. We designed a dynamically reconfigurable system where a Xilinx Microblaze processor core is mapped into the static region, while the design under test is mapped into the dynamic region. Although other controller solutions exist for managing the reconfiguration (e.g., based on ad-hoc hardware unit), we adopted the Microblaze processor since it represents one of the state-of-the-art solution for a dynamically and partially reconfigurable system based on static and dynamic regions. For design validation and fault injection, the same Microblaze processor is used to apply the test patterns to the DUT and to read the outputs from the DUT through the GPIO port. Moreover, another GPIO port connected to the flags stemming from the DUT region and configured in interrupt mode is responsible for informing the Microblaze in case of errors. After the bit-stream is downloaded to the FPGA the Microblaze memory needs to be initialized with a golden copy of the DUT bit-stream by reading the configuration area of the DUT with the Xilinx hardwired Internal Con-figuration Access Port (X-HW-ICAP). As the placement of the different regions was made at design time the Microblaze processor uses that information to build up a bit-stream database for different error regions, as discussed in section 4.2. For F-DWC re-gions, the bit-stream is stored in such a way that if an error is detected by a given flag, the frame can be recalled to reconfigure the affected area. However, the bit-stream for the C-DWC region is stored as a partial bit-stream by reading with the ICAP from the start address to the end address. In the following sections, we present several results mainly related to the ability of quick error detection, localization and repairing. Furthermore, cost in terms of area overhead and speed is also presented. We selected a wide range of circuits as benchmarks and proof-of-concept of our approach. The circuits include some relevant ITC'99 benchmark circuits with various complexity, two implementations of the Cordic arithmetic processor, a mini Mips processor, a lightweight 8080 SoC, an RS-Decoder and a DCT core from the opencores repository.

In Table 4.1, we reported the number of LUTs and FFs of the implemented circuits. We compared the used resource of our approach with the resources used by the original circuits implemented without any error detection or mitigation techniques and with the detection mechanism based on the duplication with comparison using Double Modular Redundancy (DMR) and detection and mitigation mechanism based on Triple Modular Redundancy (TMR). Please note that we did not include the amount of resources related to the static region within the area count since it

Table 4.1. Characteristics of the implemented circuits

Circuit	Original		DMR		TMR		Our Approach	
	LUTs	FFs	LUTs	FFs	LUTs	FFs	LUTs	FFs
B03	39	35	78	69	180	111	85	69
B04	115	67	213	131	449	201	250	131
B05	163	42	326	83	792	135	390	83
B07	98	50	196	99	406	153	18	16
B08	20	21	40	42	89	63	49	42
B09	49	28	98	56	117	84	108	56
B10	37	24	72	47	157	72	80	47
B14	1,161	217	2,322	434	6,417	669	2,552	434
B15	1,900	425	3,480	849	8,358	1,275	4,311	849
Cordic_rp	1,024	1,019	2,048	2,038	3,658	3,387	2,329	2,038
Cordic_pr	689	690	1,378	1,380	2,259	2,259	1,446	1,380
miniMIPS	3,200	1,883	6,439	3,764	5,649	5,649	6,789	3,764
L80SoC	261	237	473	1,524	726	609	609	473
RS_Decoder	4,191	2,801	5,600	18,452	8,403	9,056	9,056	5,620
DCT	1,254	1,935	3,866	5,608	4,755	2,811	2,811	3,866

remains constant for different circuits implemented within the dynamic region. The resources usage values report that our approach is far better than TMR, since on the average, TMR requires 3.64 times more hardware resources than the original circuit, while our approach requires 2.10 times more resources only. If compared with DMR, our approach requires 10% more resources on the average; however, DMR cannot correct errors, while our approach corrects errors and reduces the probability of single points of failure thanks to the developed fine grain combinational logic infrastructure. It is important to underline that the area comparison has been performed directly on the basis of LUTs and FFs counts; while if comparison is made considering the number of FPGA slices, the ratio may by slightly different due to stringent packing and placement requirements adopted for the fine-grain redundancy with comparison logic. In particular, slices are used as a route-through and FFs may be placed in separate slices, since the FFs require different control signals that were not possible to be packed together with LUTs.

The measurement of the error detection latency is the key factor for making

proper self-repairing system able to autonomously repair itself obeying real-time constraints. The results we obtained are illustrated in Table 4.2, where it is shown the maximum error detection latency for SBE and MBE regions. In detail, the table reports the length of the carry chain detector, the delay latency with routing and logic contribution of the SBE region, while the distance from the detector and the delay latency for the MBE region. It is notable that the SBE regions latency is more than the MBE regions because all the carry chains in each CLB that resides in the same column have been connected in a unique CLB column. The analysis of the obtained data demonstrated that the error detection latency is affected by two aspects: the routing congestion and the detector location. Where for the SBE region error detectors the contribution of latency due to logic delay for a fully connected column is around 22.94 ns (almost the 40%) compared to the routing delay contribution while amounts to 35.53 ns (almost 60%). Detailing the delay analysis, it is possible to note that the logic delay is proportional across a column of slices, since the number of carry chain detectors provides a proportional delay contribution. Vice versa, the routing delay has a higher variability due to the routing congestion induced by the adjacency of slices making the interconnection tiles less available for longer routing tracks. This aspect does not only increase the overall latency but introduces severe constraints on the routing usage in terms of design working frequency. Two alternatives have been used in order to reduce the routing delay time. The former one is oriented to relax the placement constraints of the overall design logic resources thus leaving some empty slices in the SBE regions; this permits placing carry chain flag interconnections only on a single CLB column avoiding carry chain flag interconnections convergence across two or more slice columns. The latter solution consists on the usage of LUT's configured as OR gate in order to converge flag signals. This introduces a trade-off between the area overhead and the router completion time. It is worthwhile to mention that although the area overhead increases, however, the maximum error detection latency decreases. Similarly, the MBE regions use LUT's configured as OR gate for the reduction of the interconnection flags. As a result, the error detection latency for MBE region has a logic delay contribution of 0.19 ns while the routing delay is 2.92 ns which correspond respectively to almost the 6% and 94% of the overall logic delay. These asymmetrical values are due to the fact that in the MBE region the carry chain detector is one slice long and the error detection latency directly depends on the placement location of the OR-LUTs and their distances from the flag generation node. The effectiveness of the proposed approach concerning the error correction and detection capabilities have been evaluated through the execution of fault injection campaigns. The experiments have been performed on the Xilinx Virtex-5 LX110T SRAM-based FPGAs by injecting transient faults into the FPGA's configuration memory and evaluating the circuit's response through the execution of circuit specific workloads. Please

Table 4.2. Error Detection Latency for carry chain detectors

Circuit	SBE Region				MBE Region	
	Length	Latency (nanosecs)	Logic Delay (nanosecs)	Routing Delay (nanosecs)	Distance	Latency (nanosecs)
B03	4	8.81	5.73	3.09	4	1.25
B04	8	19.62	9.77	9.85	8	1.52
B05	14	42.61	16.03	26.58	14	2.07
B07	10	22.87	11.67	11.22	10	2.76
B08	3	6.63	4.63	2	2	1.15
B09	2	3.8	3.47	0.33	14	2.82
B10	4	11.98	4.59	7.39	2	1.023
B14	20	55.81	22.93	32.88	18	3.10
B15	20	51.26	22.10	29.16	19	3.42
Cordic_rp	0	0	0	0	3	1.99
Cordic_pr	0	0	0	0	2	1.85
miniMIPS	20	52.43	22.98	28.46	19	3.99
L80SoC	19	40.50	21.86	18.64	16	2.44
RS decod	20	44.80	22.98	21.82	19	3.86
DCT	13	23.84	14.97	8.86	17	2.99

note that the faulty bitstreams are generated by corrupting the FPGA's configuration memory bits belonging to the dynamic region, while the static region which contains the reconfiguration controller, is kept fault free. Table 4.3 shows the fault injection results, where for each circuit 10,000 Single Event Upsets (SEUs) have been randomly injected into the whole FPGA configuration memory bits related to the reconfigurable region. All the circuits have been emulated at 50 MHz and SEUs are practically injected by downloading the corrupted bitstreams into the FPGA configuration memory.

In details, the table reports the number of SEUs and MEUs provoking a wrong answer on the circuit output that following activate the error correction mechanism; while the corrected column reports the number of SEUs and MEUs properly corrected by our approach. Please note that the MEU effect considered in our experiments always occur in different slice columns involving the modification of two configuration memory bits. This correspond to the worst case scenario; in fact, correction of SEUs and MEUs in a single slice columns takes always the same amount

Table 4.3. Fault injection campaign experimental results

Circuits	SEUs		MEUs	
	Wrong Anwer	Corrected	Wrong Answer	Corrected
B03	2,448	2,416	2,944	2,875
B04	584	583	632	621
B05	782	781	942	894
B07	4,762	4,639	5,682	5,434
B08	1,425	1,423	1,704	1,676
B09	1,784	1,776	8,938	8,852
B10	1,903	1,892	5,986	5,940
B14	5,121	5,041	5,443	5,331
B15	6,930	6,883	7,240	7,124
Cordic_rp	4,932	4,915	5,230	5,148
Cordic_pr	3,142	3,089	3,350	3,268
miniMIPS	8,903	8,895	9,104	8,957
L80SoC	1,238	1,211	1,469	1,429
RS decod	9,236	9,177	9,491	9,379
DCT	5,232	5,173	5,523	5,372

of time, vice versa, errors in different columns require a longer computational time before to be corrected. The obtained results demonstrate the efficiency of our approach, which is able to correct more than the 98% of the injected errors provoking wrong answers for all the considered circuits.

We also measured the recovery time as presented in Table 4.4 we reported the worst recovery time measured for all the circuits during the execution of the fault injection campaigns. As it is possible to notice, the advantage provided by our approach is extremely large on all the considered circuits. We also measured the average recovery time for the tested circuits which corresponds to 76.3 microseconds and 158.9 microseconds for SEUs and MEUs respectively. Comparing these recovery times with the ones required by redundancy approaches such as TMR and DMR using active configuration memory scrubbing of all the reconfigurable region area, which is about 1.2 milliseconds; our approach has an improvement of more than one order of magnitude.

We evaluated the impact on the circuit maximal working frequency on all the

Table 4.4. Recovery Time comparison (worst case)

Circuit	DMR [μsecs]	TMR [μsecs]	Our Approach [μsecs]
B03	383.7	1,033.2	119.3
B04	678.9	1,239.8	237.7
B05	1,269.4	3,070.1	238.2
B07	531.4	1,594.1	238.9
B08	88.6	856.1	119.2
B09	206.6	501.8	120.9
B10	501.8	974.2	237.2
B14	2,922.5	5,667.8	829.7
B15	5,077.4	8,796.9	2,010.8
Cordic_rp	3,483.4	4,693.7	120.1
Cordic_pr	1,416.9	4,073.8	119.9
miniMIPS	8,029.4	11,335.7	2,011.4
L80SoC	2,154.9	2,892.9	474.8
RS_decoder	11,040.5	17,062.6	2,129.3
DCT	4,250.9	19,128.9	2,364.6

implemented benchmark circuits comparing our approach with the DMR and TMR redundancy based techniques. The results are illustrated in figure 4.5 which is reporting the maximum clock period of our benchmark set comparing among different redundancy based approaches. As it is possible to notice, our approach has a reasonable behavior for medium complexity circuit, while it has a penalization up to the 22% of the nominal working frequency for larger circuits. This phenomenon is due to the unconventional block placement of logic resources on slice columns for different circuit regions. This aspect affects the timing of the circuit because our technique does not include an optimal floorplan implementation of the different circuit region. The maximal circuit clock period can be further optimized using floorplan oriented placement algorithms by acting the fine-grain logic packing. These logic resources can be packed per slice columns in order to tightly place the fine grain duplication with comparison LUTs resources in the optimal way.

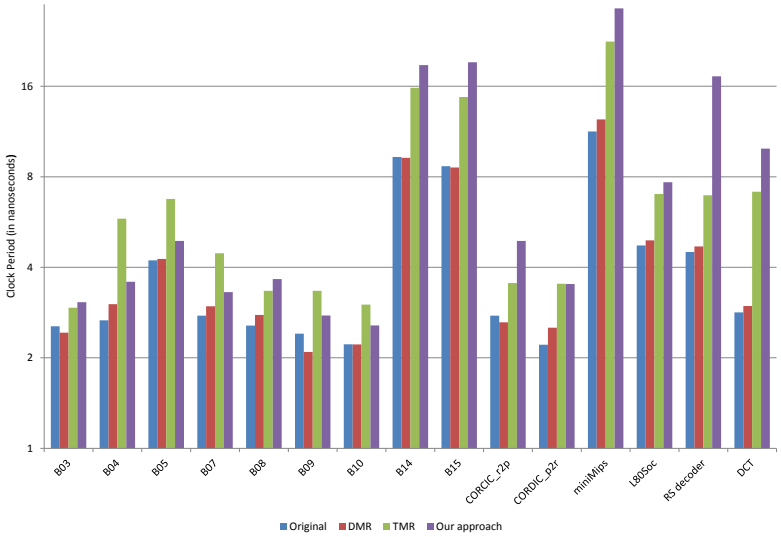

Figure 4.5. Clock period comparison of the implemented circuits using different error detection and correction approaches.

Chapter 5

Dynamically Reconfigurable Triple Modualar Redundancy

The rapid adoption of FPGA-based systems in space and avionics demands dependability rules from the design to the layout phases to protect against radiation effects. Triple Modular Redundancy is a widely used fault tolerance methodology to protect circuits against radiation-induced Single Event Upsets implemented on SRAM-based FPGAs. The accumulation of SEUs in the configuration memory can cause the TMR replicas to fail, requiring a periodic write-back of the configuration bit-stream. The associated system downtime due to scrubbing and the probability of simultaneous failures of two TMR domains are increasing with growing device densities. We propose a methodology to reduce the recovery time of TMR circuits with increased resilience to Cross-Domain Errors. Our methodology consists of an automated tool-flow for fine-grain error detection, error flags convergence and non-overlapping domain placement. The fine-grain error detection logic identifies the faulty domain using gate-level functions while the error flag convergence logic reduces the overwhelming number of flag signals. The non-overlapping placement enables selective domain reconfiguration and greatly reduces the number of Cross-Domain Errors. Our results demonstrate an evident reduction of the recovery time due to fast error detection time and selective partial reconfiguration of faulty domains. Moreover, the methodology drastically reduces Cross-Domain Errors in Look-Up Tables and routing resources. The improvements in recovery time and fault tolerance are achieved at an area overhead of a single LUT per majority voter in TMR circuits

5.1 Cross-Domains Errors and Scrub Time

In order to avoid single point of failures in circuits implemented on SRAM-based FPGAs, Triple Modular Redundancy (TMR) is the main methodology selected by designers [64]. The logical fault masking ability of TMR enables the systems to operate fault-free if an SEU creates a bit flip in one of the TMR domains. However, the persistence of SEU effects in the configuration memory can result in failures of two TMR domains and thus the fault masking can break. This requires periodic (e.g., blind scrubbing) [73] or selective (e.g., single frame read-back and correction [99]) reconfiguration of the bit-stream to stop the accumulation of SEUs in the configuration memory. While the growing device densities enable the implementation of more functionality per unit area, it has two important consequences for TMR circuits implemented on SRAMbased FPGAs. First, the close proximity of SRAM-cells increases the probability of a single particle to effect two neighboring cells. These single event multiple bit upsets can become critical for TMR circuits when the two neighboring cells are storing the configuration bits of two different TMR domains. These multiple bit upset increases Cross-Domain Errors (CDEs) that result in TMR failures [66]. Second, the growing size of configuration memories and the unimproved speeds of reconfiguration interfaces are causing the reconfiguration times to increase with each new generation of FPGAs. For example, the 28 nm Virtex-7 FPGA has a configuration time of 125 milliseconds [54]. This system downtime can be unaffordable for safety-critical applications with strict real-time requirements. Nowadays, FPGA implementation tools usually tend to optimize the overall circuit performances, however, such kind of optimizations have an opposite effect on dependability of circuits. Since, logic and routing resources tend to have a high congestion, particularly in placing voting resources in the same Configurable Logic Block (CLB), the resultant placement increases the probability of TMR failures. A conservative placement of each domain to separate physical area on the chip may ensure that logic and routing elements from different TMR domains do not increase the TMR failure probability. A preliminary work presented in [100] instruments each TMR domain with error detection logic for errant domain identification and leverages the non-overlapping domain placement for selective reconfiguration. This chapter improves the state-of-the-art techniques with several novelties. The former, consists in an error detection logic introduced within the TMR voting structure. In comparison with the traditional minority-voter based implementation, the proposed method uses less routing segments. This reduces the probability of CDEs in the detection circuitry. Secondly, a new strategy for the individuation of the error location within the FPGA architecture is proposed. The developed scheme adopts carry-chain resources available in each CLB slice improving the SEU detection capability especially considering failures into the FPGAs configuration memory.

The latter contribution of the chapter is a non-intrusive net-list algorithm analyzer that allows the evaluation of CDEs at the early design phase incorporating static analyzer methodologies [101]. The proposed methodology has been evaluated on different benchmark circuits including microprocessor cores such as the MiniMIPS and Leon-II processors. The experimental results demonstrate a huge reduction in recovery times with respect to previously adopted techniques such as scrubbing. The proposed methodology completely eliminates CDEs in LUTs, while greatly reducing them in the routing interconnects. Moreover, the proposed methodology incurs an area overhead of one LUT per internal majority voter. For all the benchmark circuits that we considered in this chapter, the error detection logic did not alter the critical path of the circuits. Interestingly, the custom floor-planning of TMR domains enabled the placer to optimize the wire-length more rigorously resulting in performance improvements in some cases.

5.2 TMR architecture and Layout

The goal of this section is to present a new TMR fault-tolerance methodology that reduces the recovery time and the number of CDEs affecting circuits implemented on SRAM-based FPGAs. The methodology mainly consists of architectural and layout level changes introduced in the Xilinx TMR architecture [64].The architectural-level changes include the insertion of error detection and error flag convergence logic while the layout-level changes consist of non-overlapping domain placement. Error detectors and flag convergence logic are inserted at the granularity of TMR domains. The error detection logic uses gate-level functions for comparison, resulting, in a large number of error flags for each domain. As it is fruitless to identify an errant domain with more than one error flags, the flag convergence logic connects together all the flags from a single domain, generating a unique flag per TMR domain. A novel flag convergence logic is introduced in this section to make the realization of fine-grain error detection feasible for TMR circuits.

A fine-grain error detection logic based on logic gates is used to identify faults in a TMR domain. The implementation and insertion of error detection logic in TMR circuits can be accomplished in three different ways mainly based on minority voter and equivalence function shown in figure 5.1. The minority voter has three input signals consisting of a primary and two secondary inputs. The minority voter circuit generates a logic 0 value on the output when the primary agrees with at least one of the secondary inputs otherwise the result is a logic 1 value [64]. Thus, the output of minority voter can be used as an error flag signal. In figure 5.1a, the minority voter is fed by the outputs of majority voters. In case of a fault in one

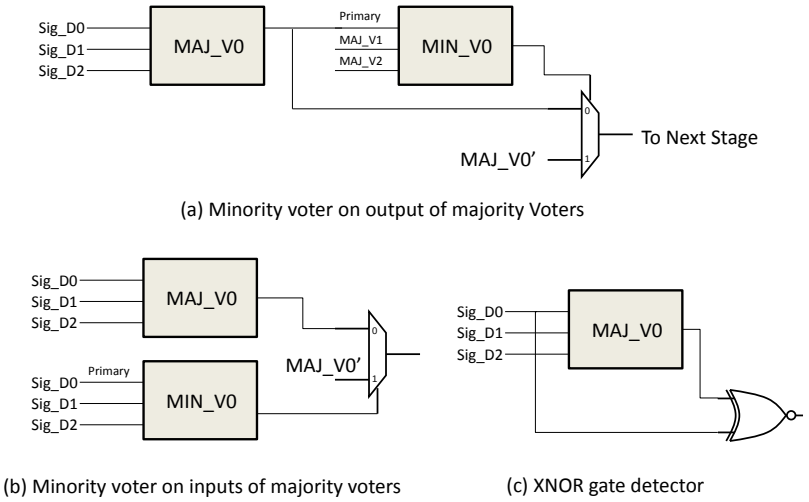

Figure 5.1. Gate-level Error Detection Circuits for TMR Architecture

of the majority voters, the corresponding minority voter will flag an error condition and will disable the propagation of error to the down-stream logic by controlling the select line of the multiplexer. The multiplexer receives inputs from the primary domain and one of the secondary domains. However, a fault in any resource before the majority voter will be masked from reaching the minority voter and it cannot be detected. In 5.1b, the minority voter is placed on the signals that feed the majority voters. In this way, the majority voter will mask faults while the minority voter will detect faults. This type of detector has the ability to detect faults in the resources before the majority voters but a fault in majority voter itself cannot be detected using this method. As the minority voter based detectors require input signals from three different domains of TMR, it is subject to the same reliability consideration during the placement as the majority voter. Furthermore, their usage can lead to routing congestion and error un-detection if they are placed in the same CLB with the majority voters. Considering the propagation of a fault from a domain in a given partition to the next, the majority voters will mask the fault. Similarly, if the next partition is the output stage of the TMR circuit, the minority voters will disable the output to avoid signal contention on the output package pins of the FPGA. Therefore, the steering logic, implemented as multiplexer in minority voter based detectors, can be removed without affecting the functionality of the circuit while the repairing procedure is carried out. To overcome the limitations of minority voting based error detectors, this chapter adopts an XNOR based error detector as

illustrated in figure 5.1c As can be noted from the figure, the XNOR gate is fed by the majority voter and the primary signal that feeds that majority voter. The output of XNOR gate produces a logic 1 value when both the inputs are the same. A fault affecting the logic module before the majority voter or a fault in the majority voter itself will result in a logic 0 value. Therefore, this circuit combines the error detection capabilities of the individual minority voter based detectors. Moreover, as compared to the minority voter based detectors, XNOR detector will be more resilient to routing errors because it receives inputs from the same domain logic. As you can notice from the scheme illustrated in figure 5.1, compared to the minority voter based detectors it uses less number of inputs, thus in a highly congested design this saving can be significant in terms of the routability and congestion avoidance.

5.2.1 Flag Convergence Logic

The use of fine-grain error detection logic generates a large number of error flags for each TMR domain. As this chapter tends to detect errors at the granularity of TMR domains, more than one error flag emerging from each TMR domain is not useful. The purpose of flag convergence logic is to connect them together to have a unique flag per TMR domain. This is achieved by using the built in-slice carry-chains available of fast arithmetic computations in a novel way. As a large number of carry-chains available in each CLB slice on modern FPGAs remains unused because the application at-hand does not require it or because the CAD tools could not infer them, they can be exploited for certain applications [50] [102]. The insertion of error detection and flag convergence logic in a TMR domain is illustrated with a placement-level view in figure 5.2 where SLICE 1 contains the majority voters while SLICE 2 contains the error detection and flag convergence logic. The placement of same domain voters is a mandatory constrain for our methodology. It should be noted that modern FPGAs provide local power supply and ground located near each slice column in the form of the TIEOFF elements, making it possible to connect the inputs of the hardwired multiplexer in the carry-chain to the fixed logic values like VCC and GND. These elements can be configured to VCC or GND by modifying the physical circuit description in the post-map net-list. Although figure 5.2 shows the flag reduction for four XNORs, this method can be extended to connect multiple slices using the dedicated carry-wires [103]. The usage of carry-wires to connect the hardwired carry-chain resources makes the detectors resilient to errors in the detector itself. Although, the proposed solution efficiently detects error in a TMR domain with a unique flag, however, due to the propagation delay along the carry-chain, the method has associated error detection latency. The error detection latency depends upon the number of majority voters at the boundary of a TMR domain in each partition. The formation of partition depends upon the nature of computations that

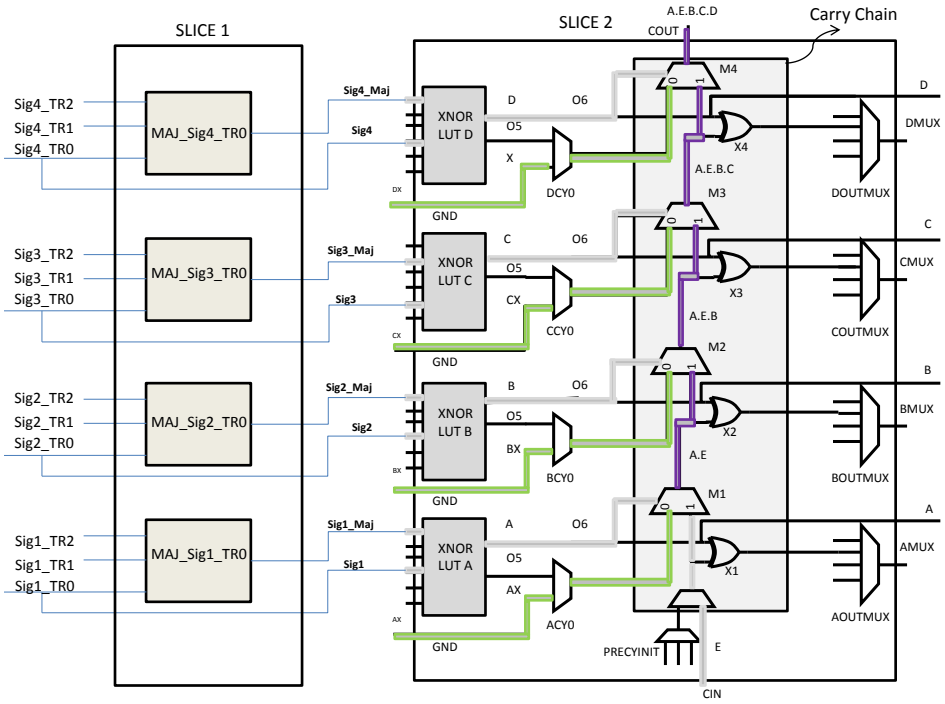

Figure 5.2. A device-level view of the insertion of error detection and flag convergence logic in domain

the circuit implements causing a considerable variance in the size. Consequently, the size of domain within the same partition is constant but it can vary when we move to other partitions. Therefore, the detector length and error detection latency varies in the same TMR circuits for domains of different partitions. This chapter does not optimize the trade-off between the detector length and the number of error flags; instead the goal is to reduce the number of error flags to the minimum possible for every domain.

5.2.2 Non-Overlapping Domain Placement

The non-overlapping domain placement is the layout of TMR domains to separate and non-overlapping areas on chip. As the current CAD tools from the vendors are optimized for timing, the stringent requirements on the timing closure may lead to place elements from different domains in the same slice. This results in a critical voter placement scenario as shown in figure 5.3a. It can be noted that the voters belonging to the same signal group, Voter 1, but different domains are placed in the

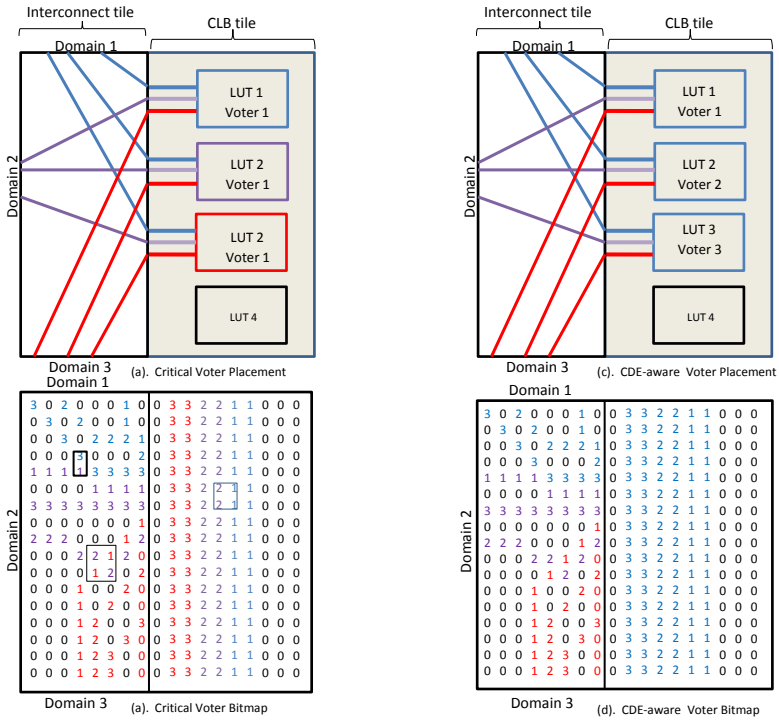

Figure 5.3. Effects of placement on cross-domain errors

same CLB slice. Each domain is represented with a different color. The corresponding critical voter bitmap is shown in figure 5.3b, where 1, 2 and 3 represents the configuration bit for the corresponding LUTs. The configuration bit 0 means that the bit is un-used. The criticality of some of the neighboring bits, responsible for the configuration of two different TMR domains is shown with overlapped squares in figure 5.3b. The close proximity of bit-stream frames in this scenario results in a large number of CDEs in logic as well as in routing. In particular, the interconnect tile contains nine nets from three different domains that feeds combination logic from three different domains. One example of a CDE-aware voter placement is shown in figure 5.3c. where the voters from different signal groups, denoted by Voter 1, Voter 2 and Voter 3, but belonging to the same domain, shown with same color, are packed together in a CLB slice. The corresponding CDE-aware bitmap in figure 5.3d, which follows the same convention, demonstrates a more reliable situation. It can be noted that the number of CDEs in the configuration frames of CLB tile in this case are zero. The constraints on placement not only guarantee that voters are placed in CDE-aware manner but it also places all other logic elements separately from other domain elements. Consequently, most of the routing interconnection reside inside the boundaries of a placed domain and the routing CDEs considerably decreases as well. As each of the TMR domains is already instrumented with logic that identifies

the errant domain, the non-overlapping placement can be used to correct the faulty domain without disturbing the real-time operation of the circuit. This is achievable by exploiting the Dynamic Partial Reconfiguration (DPR) capabilities of modern SRAM-based FPGAs. Therefore, the non-overlapping domains placement accounts for the dramatic decrease in the reconfiguration times of TMR circuits using partial reconfiguration of a faulty domain. The detailed explanation of the implementation of placement, for commercial FPGAs, is presented in the next section.

5.3 XDL-level CAD Flows

This section presents a physical-level CAD flow for implementing the proposed TMR architecture and layout methodology as propose in figure 5.2. The physical-level algorithm inserts the error detection and flag convergence logic in the mapped design at the level of Xilinx Design Language (XDL). The detailed flow is illustrated in figure 5.4. The physical-level flow is based upon the modification of the mapped TMR

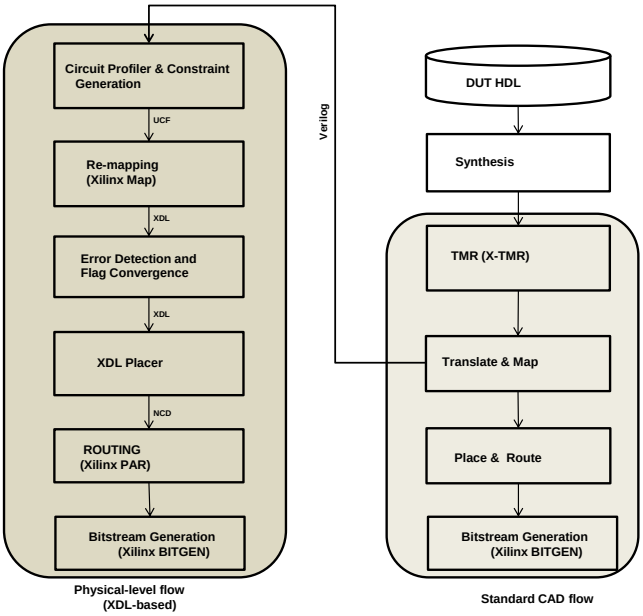

Figure 5.4. The Proposed Physical-level Flow

circuits to the FPGA architecture. The mapping process packs the design LUTs and FFs into slices and assigns them physical locations on the FPGA fabric. The physical flow starts from the mapped design and applies the proposed architectural and layout-level changes presented in figure 5.4. First, a re-mapping of the TMR

circuit with packing constraints is performed to avoid logic elements from different domains to be placed in the same CLB slice. This is followed by insertion of the error detection and flag convergence logic section 5.3.1 and finally the custom placement section 5.3.2 algorithm performs the non-overlapping domain placement.

Algorithm 5.1 Physical-level Manipulator

Input: Voters list file , XDL file

Output: Modified XDL file with error detection and reduction logic

```
1: // ********** Initialization Phase **********
2: PartitionDomainMap[Voter] = extract_partition_domain(Voter);
3: SliceLutMap[Voter] = extract_slicePtr_lutPtr(Voter);
4: VoterNetsMap[Voter] = extract_inputs_for_XNOR(Voter);
5: // ********** XNOR LUT Creation Phase **********
6: foreach Voter v in VoterNetsMap do
7:     XNOR_inputs = VoterNetsMap[v];
8:     NewXNORLUTObjtMap[v] = XNORLUTObjts(XNOR_inputs);
9: end foreach
10: // ********** Slices and Carry-Chains Insertion **********
11: // Notation: p "partition" and d "domain"
12: foreach voters_group with same "p,d" do
13:     XNORs_cluster = NewXNORLUTObjectMap[voters_group];
14:     NewSliceInstace = CreateNewSliceInstance(XNORs_cluster);
15:     InsertCarryChain(NewSliceInstance);
16:     NewSlicesMap.insert(NewSliceInstance);
17:     Connect2PreviousSliceinNewSliceMap(NewSliceInstance);
18:     if XNORs_cluster contains last voter in current "p,d" then
19:         flagNet = extract_flagNet_from_slice(NewSliceInstance);
20:         NewNetName = extract_name_with_partition_domain(flagNet);
21:         XDLNet = CreateNewXDLNet(NewNetName);
22:         XDLNet.source(flagNet);
23:         XDLNetMap.insert(XDLNet);
24:     end if
25: end foreach
26: // ********** Controller Connection **********
27: foreach XDLNet in XDLNetMap do
28:     XDLNet.target(ReturnReconfigurationControllerPort());
29: end foreach
```

The preliminary phase of the flow consists of creating a Directed Acyclic Graph

(DAG) of the TMR circuit by parsing the post-mapped simulation Verilog file (utilizing [60]). The nodes of the graph are modeled as an object with a number of fields such as functional string, instance name, inputs nets, output nets, type of primitive element (LUT or FF) and the corresponding domain and partition. After the net-list is created, each node should be labeled with the corresponding domain, partition and signal group. It should be noted that each node of the graph has a name containing a string referred to as TR0, TR1 and TR2 for the corresponding domain; however, in order to find partitions, majority voters should be identified in the circuit net-list because majority voters are located at the boundary of partitions. Majority voters are identified with the functional strings and three inputs each one from different domains. Furthermore, partitions are created starting from the circuits primary output pins and traversing the graph in Breadth First Search (BFS) manner towards the circuit primary inputs. Starting from a boundary of majority voters, a unique label ID for the partitions is assigned to all the elements of the graph until the next majority voter boundary is encountered. The next majority voter boundary will be assigned an incremented label ID and the partition label ID will be propagated in a similar manner as before. During the process of identification of partitions, signal groups are also formed by finding majority voter triplets which receives the same nets as inputs. It should be noted that the signal group ID is assigned in a manner that they increment linearly in the same partition but are set to zero and incremented again for the next partition. Once this process finishes, all the nodes of the graph will have the corresponding partition and signal group assigned. This information is mandatory for generating the circuit profile which is a statistics of the division of elements in domains, partitions and signal groups. The circuit profiler is used to identify the slices which are in packing violations using the criteria that all the LUT/FFs in a slice are related to the same domain. Once the elements violating the dependability rules are identified, a new constraint file is generated using the packing constraints [104] and fed to the mapper to generate a mapped design. This mapped design will drive the subsequent phases of the flow by identifying the insertion locations for error detection and flag convergence logic.

5.3.1 Error Detection and Flag aggregation

A physical-level tool was developed for inserting error detection and flag convergence logic in the slice-level FPGA design net-list. This tool is based upon the software primitives defined in [58]. The developed physical-level manipulator presented in Algorithm 5.1 requires two input files. The former, a voters list file generated by the circuit profiler which has the partition, domain and signal group information for every majority voter in the design. The latter, a physical-level file that describes the mapped design constrained as presented in section **??**. During the initialization

phase, the voters list file is read and the corresponding domain and partition for each voter is extracted and stored in *PartitionDomainMap*. Using the names of each voter, the corresponding slice and LUT pointers are extracted and stored in a *SliceLUTMap*. After identifying the voters in the slice-level net-list, the nets that will feed the XNOR gates need to be identified. For each voter, the nets on the primary inputs and outputs of the majority voters are extracted and stored in a *VoterNetsMap* data-structure. It is worth mentioning that as the physical-level description is a slice-level net-list while the error detection logic is in form of LUTs, therefore, it needs to be packed together into slices. This is achieved by creating abstract LUTs objects and connecting them to every voter in the *VoterNetsMap*. In order to pack these created LUT objects, the slice and carry-chain insertion phase combines from the same domain in a partition four XNOR LUTs into a newly created slice instance. A carry-chain object is then inserted into the newly created slice instance and is connected to the previous slice in the same domain and partition pair. This process of interleaving the chains continues until all the LUT objects of a domain in a partition are connected. The insertion of error detection logic and flag signal in the post-map net-list follows a specific naming convention, identifying each flag with its corresponding domain and partition. When all the LUT objects of a certain domain are connected, the output from the final carry-chain is the flag signal. A new net is created at this point and the output of the carry-chain is added as a source. The created nets are stored in *XDLNetsMap* for all the partitions and domains. In the controller connection phase, for every flag net in the *XDLNetsMap*, a target port on the reconfiguration controller is specified. The target port of a flag signal can be an external or internal reconfiguration controller. In case of external controller the destination is an IOB site connecting to a PAD, while in case of internal reconfiguration controller the destination can be a port of a processor or input of a hardware reconfiguration manager. It is important to note that we have added new nets in the mapped design; therefore, it is possible to carry out the routing phase using the commercial place and route tool.

5.3.2 Placement

The independent non-overlapping placement for domains is carried out on the physical-level net-list, which already packed together LUTs and FFs into slices. The physical-level placer which uses a simulated annealing algorithm [58] modified for independent placement of domains in different physical areas as presented in the Algorithm 5.2. As a pre-processing step, the physical-level description file is analyzed to extract a resource vector representing the computational nodes that will be required for each domain in every partition. An area range on the chip is selected for each resource vector that can house it. More formally, the placement problem

formulation can be laid out in the following manner. Given a TMR partition set represented by P=$\{p_1,p_2,...,p_n\}$ where each partition p_i contains a 3-tuple of domains D=$\{d_1,d_2,d_3\}$. Let for each domain dj belonging to partition p_i denoted by $d_{i,j}$ a 3-tuple $rv_{i,j}$=$\{n_{clb},n_{bram},n_{dsp}\}$ represents a resource vector. Moreover, a set of resource vectors RV=$\{rv_{1,1},rv_{1,2},rv_{1,3},.....,\ rv_{n,1},rv_{n,2},rv_{n,3}\}$ represents the resource requirement for all the partitions and domains. Similarly, a rectangular region denoted by $rr_{i,j}$ is the corresponding area on the FPGA that satisfies the resource vector $rv_{i,j}$ for a domain $d_{i,j}$. Each rectangular region is a 4-tuple $\{x_0,y_0,x_1,y_1\}$ where the bottom left corner is represented by (x_0,y_0) and the top right corner is represented by (x_1,y_1). The floorplan for the whole design can be represented by RR=$\{rr_{1,1},rr_{1,2},rr_{1,3},rr_{n,3}\}$ which is a vector of rectangular regions on the FPGA fabric. Moreover, PL=$\{P_{1,1},P_{1,2},P_{1,3},P_{n,3}\}$ is the random initial placement of all the domains according to the selected floor-plan. The detailed placement algorithm is shown in the following XDL-level simulated annealing placer.

Algorithm 5.2 Simulated-annealing based physical-level placer

Input: Unplaced XDL file
Output: Placed XDL file

1: // ********** Initialization Phase **********
2: $RV\{rv(1,1), rv(1,2), rv(1,3), \ldots, rv(n,3)\} = extract\_resource\_vector(DAGTMR)$;
3: $RR\{rr(1,1), rr(1,2), rr(1,3), \ldots, rr(n,3)\} = floorplan\_area\_on\_chip(RV)$;
4: $PL\{pl(1,1), pl(1,2), pl(1,2), \ldots, pl(n,3)\} = random\_initial\_placement(RV, RR)$;
5: // ********** Simulated annealing engine **********
6: **foreach** pl(i,j) belonging to PL **do**
7: T = Initial Temperature
8: **while** $(T > FinalTemperature)$ **do**
9: $pl(i,j)_{new} = swap\_two\_randomly\_selected\_elements(pl(i,j))$;
10: $\triangle C = Cost(pl(i,j)_{new}) - Cost(pl(i,j))$;
11: **if** $(\triangle C < 0)$ **then**
12: $pl(i,j)) = pl(i,j)_{new}$;
13: **else**If $PickFromUniformRandomDistof(0,1) < exp(- \triangle C/T)$
14: $pl(i,j)) = pl(i,j)_{new}$;
15: **end if**
16: $T = update(T)$;
17: **end while**
18: **end foreach**

In the initialization phase, the TMRs DAG is traversed to divide the logic resources into the corresponding domains and partitions. The information extracted from the circuit profiler phase can also be used to locate the LUTs/FFs according to domains and partitions. Based on the required resource a feasible area for each

domain is selected. It should be noted the floor-planning step is not optimized in any way. This is followed by an initial random placement of domains according to the selected floor-plan. The simulated annealing algorithm starts by initializing the temperature T to a very large value. For each domain, in each iteration, two randomly selected logic elements are swapped to check if the half-perimeter wire-length decrease. Each good move is accepted, however, a bad move can also be selected according to a probability. The probability of accepting a bad move is high in the beginning to enable the algorithm to avoid local minima but decreases with time for convergence towards global minima. The physical-level placement algorithm combined with the un-optimized packing results in a decreased performance for this flow which can be vastly improved by the non-intrusive semi-custom flow presented in the EDIF-based flow in the following section.

5.4 EDIF-level CAD Flows

The net-list described in EDIF grammar is designed with powerful expressive capabilities for all the levels of abstractions related to the electronic design and automation of circuits and systems. It is the industry wide standard for interchanging designs from net-list description to layout phase. Therefore, an alternative net-list based CAD flow is developed to automatically harden circuits according to our proposed methodology to show viability for other vendors. The net-list level CAD flow (based upon [58]) is shown in figure 5.5. The flow starts from the triplicated net-list generated by the TMR tool and applies several modifications including hierarchy formation, error detection and flag convergence logic insertion, floor-planning and placing. The rest of the details are in the following sub-sections.

5.4.1 Flat to hierarchical Netlist Conversion

The input to the hierarchy formation block is a triplicated net-list containing instances from the macro block synthesis library containing cells that are used by TMR tool for hardening the circuits [64]. The functionality of this block is like the circuit profiler block from section 5.3. Like the circuit profiler block the purpose is to label each element with a corresponding domain and partition. The net-list uses elements from a library that have special cells for voter structures and the output buffer elements. In contrast to the post-map simulation Verilog file used in section 5.3, it is easy to identify them and divide the circuit into partitions. In a similar fashion to the circuit profiler, a BFS algorithm starting from the circuit primary output towards the circuit primary inputs labels each element with a corresponding

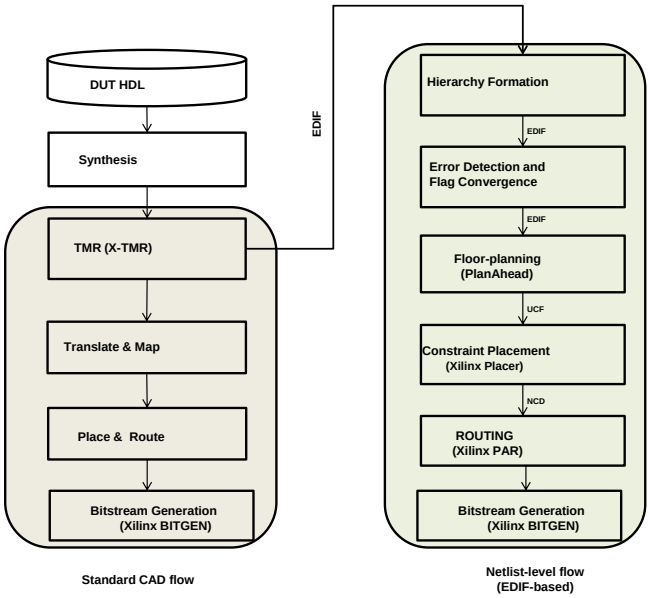

Figure 5.5. The Proposed Netlist-level Flow

domains and partitions. The information extraction from the profiling is used to convert the flat net-list generated by the TMR tool into a domain-level hierarchical net-list. In order to achieve this task, each logic element from every domain and partition is annotated with the corresponding partition and domain. The instances of each cell are renamed by including it in a domain and partition hierarchy. This kind of hierarchy formation is important for the non-overlapping domain placement which is a central theme of this chapter. The hierarchical net-list is used for constraining the floor-planning and placement using commercial tool-chain and will be in described in section 5.4.3.

5.4.2 Error Detection and Flag Convergence Logic Insertion

The physical-level description file used in section 5.3.1 is a slice level circuit description, which already packed LUT/FFs and carry-chains and assigns them physical locations on the FPGA. However, the net-list level flow uses a cell-level net-list utilizing basic elements that comprises slices. Therefore, it is possible to insert the gate-level detectors directly as LUTs and connect them together using instances of carry-chain mux cells. The developed Net-list Manipulator Algorithm 5.3 shows the implementation details for the insertion of error detection and flag convergence logic as illustrated in figure 5.2. In the XNOR gate insertion phase, a new LUT instance

Algorithm 5.3 Netlist Manipulator

Input: EDIF Netlist file
Output: Modified EDIF file

```
// ******************** XNOR gate insertion ************************
foreach voter "v" which is not an output voter  do
    [primary_net, voter_out_net] = getPrimaryNetandOutputNet(v);
    XNORMap[v] = InsertXNORInstance(primary_net, voter_out_net);
end foreach
// ******************** flag convergence logic ************************
foreach partition "p" belonging to TMR partitions do
    foreach domain "d" belonging to partition "p" do
        foreach XNOR gate "x" in XNORMap do
            if domain[x] == d && partition[x] == p then
                if first element in the chain then
                    // Implement MUXCY "M1" in Fig 5.2
                    Y = InsertMuxCyInstance(GND, output(x), VCC);
                else
                    Y = InsertMuxCyInstance(GND, output(x), prv_O);
                end if
                prv_O = output(Y);
                if last element in the chain then
                    FlagNets.insert(prv_O);
                end if
            end if
        end foreach
    end foreach
end foreach
// ******************** Adding Top-Level EDIF flag Ports ***************
FlagPort = CreateTopLevelEDIFPort(FlagNets.size());
foreach flag belonging to FlagNets do
    connectflag2portindex(flag, port, index);
    increment_index;
end foreach
```

is created for the XNOR gate. For each voter, the primary input net and the voter output nets are extracted and are connected to the input of created XNOR LUT object. All the created XNORs are stored in a map data-structure *XNORMap* indexed by the voter instance. It should be noted that the insertion of XNOR gates follows a naming convention that labels them according to the corresponding voters partition and domain. The output from the last XNOR gate is stored in the *FlagNets* vector. In the last phase of top-level port creation, an EDIF vector port is created equal to the size of *FlagNets*. Each of the flag net is connected to a unique bit index of the vector port. This completes the Netlist manipulation algorithm. This methodology is completely compatible with the commercial tool-chain and can be easily adopted for external or internal reconfiguration controller.

5.4.3 Floorplanning and Placement

The modified hierarchical net-list is used for floorplanning and placement using commercial tools. The floor-planning and placement tools are directed using constraints described in [104] to respect the non-overlapping domain placement required for improving the dependability and recovery time. These constraints are specified for each partition and domain in the design. This helps the floor-planning tool to correctly recognize the hierarchy and extracts the corresponding modules from it. Then, each module is manually floorplanned by selecting a feasible rectangular region on the FPGA fabric. It is also important that during the floor-planning process, the resource requirements for each domain should be satisfied. The use of special resource type like hardwired processors and memories should also be planned accordingly for each domain. For very large designs that may require a large percentage of the device resources, the use of very long detectors can result in infeasible floor-planning, which can be elevated by increasing the number of flag signals per domain. The completed floor-planning information is added as new constraints into the previously created constraint file which is then used by the placer. The placed design is then routed using the commercial router. The usage of commercial tools for placing the design in the net-list level flow as compared to custom placer in the physical-level flow, results in better performance as presented in the following experimental results section.

5.5 Results and Discussions

To evaluate the proposed approach, the whole system was implemented for Xilinx Virtex-5 LX110T FPGA. A number of test-bench circuits from the processors category of OpenCores [105] repository were used for experiments. All the circuits were

hardened using the X-TMR tool and modified according to the methodology presented in section 5.3 and 5.4. The following sub-sections emphasize the main results comparing Xilinx TMR with minority-voter [100] and XNOR detectors based TMR methodologies.

The circuit profile statistics is the division of resource in the TMR circuits according to partitions as shown in Table 5.1. Partitions are formed by the TMR tools according to the nature of computations the circuit implements. Majority voters in the circuit mark the boundary of TMR partitions. In general, the greater the number of feedback registers in design the more will be the partitions formed. It is important to note that there is a considerable variance among the sizes of TMR partitions. The number of signal groups refers to the triplets of voters in the design and depends upon the number of primary outputs of the circuit. The number of LUTs and FFs shown here is the sum of all the resources that are spread in the all the TMR partitions. Table 5.1 also shows the special type of resources in form of DSPs and BRAMs that are required by each circuit. The circuit profile statistics drives the variance in area overhead of the proposed methodology and will be explained in more details latter in this section. The recovery time comparison of the proposed

Table 5.1. Circuit Profiling Results

Circuit	Partitions	Signal Groups	LUTs	FFs	DSPs	BRAMs
RISC5x	3	192	1,805	495	0	0
CORDICR2P	1	40	4,311	3,387	0	0
CORDICP2R	1	32	2,890	2,259	0	0
MINIMIPS	3	528	14,490	5,649	12	0
LEON II	4	748	32,290	8,046	3	39
L80SOC	10	137	1,548	726	0	6
HP16	2	134	3,533	876	0	0
RSDEC	5	376	3,919	2,130	0	12
FPU	2	136	26,702	14,727	36	0
DCT	3	40	6,576	4,755	0	126

approach with the standard X-TMR is presented in Table 5.2. The detector length expressed in terms of CLBs is the length of the flag convergence logic implemented with cascaded carry-chains along a column of CLBs. It directly depends upon the number of majority voters at the boundary of a partition. The detector length reported in Table 5.2 is for the partition with maximum number of majority voters

at the boundary. Nevertheless, for other partitions, the detector length and corresponding delay can be less than those maximum values shown in Table 5.2. The detector delay is the corresponding maximum delay. It is interesting to note that the delay of the longest detector of 192 CLBs, for LEON II, is 27.625 ns. Consider that the device we used in these experiments has 160 CLBs from top to bottom along the CLB column. This means that even detectors which span more than one device resource columns have extremely fast error detection times. This is due to the dedicated communication channel connecting the CLBs along a column, especially designed for fast arithmetic additions. However, such long detector lengths have to be floorplanned by partitioning the chip vertically to accommodate the design constraints imposed by our methodology. The recovery time of XNOR based detector is the sum of error detection latency and the reconfiguration time. The comparison clearly shows that the methodologies based on non-overlapping domain placement hugely reduce the recovery time. The difference between recovery times of minority voter based TMR and XNOR based TMR is not significant. The reasons are the same floor-plan and the fast error detection latency. For Xilinx TMR, the partial reconfiguration of a faulty domain is not possible and therefore in case of a fault the whole FPGA bit-stream should be written back. For a Virtex-5 device that is used in this experimental study the total bit-stream download time is 25 milliseconds. However, the Table 5.2 represents the values for Xilinx TMR considering the used frames only, in order to have a fair comparison with our approach. Although, the probability of un-detection is quite low due to the low cross-sectional area of the detector and the hardwired nature of the carry-chains, however, in order to avoid the accumulation of upsets in the configuration memory, the proposed methodology can be augmented with a full scrubbing at a considerably low scrub rate compared to the traditional TMR methodologies.

The reduction in the number of CDEs is a consequence of the non-overlapping placement of TMR domains in different reconfigurable regions. These results were obtained by using the STAR tool [101] on the physical-level description format. The STAR tool uses an analytical approach to estimate the effects of SEUs based upon a cross-correlation of the bitstream to its resource mapping. Table 5.3 represents the SEU sensitivity analysis for the standard Xilinx TMR with non-overlapping domain placement based methodologies. It can be noted that the number of CDEs are significantly reduced by our methodologies compared to the X-TMR approach. Mainly, two reasons contribute towards the significant reduction of the number of CDEs. First, the non-overlapping domain placement and second the frequent run-time partial repairing. The non-overlapping domain placement nullifies the CDEs for the combination logic in form of LUTs as discussed in section 5.2.2. The frequent selective run-time repairing greatly reduces the probability of accumulation of SEUs in the configuration memory. It can be noted that the CDEs induced by routing

Table 5.2. Recovery Time Comparison

Circuits	XNOR Detector based TMR			Min-TMR Recovery Time (μsecs)	X-TMR Recovery Time (μsecs)
	Detector Length (CLBs)	Detector Delay (nanosecs)	Recovery Time (μsec)		
RISC5x	22	11.032	74.061	74.05	666.05
CORDICR2P	0	0	530.25	530.25	1590.76
CORDICP2R	0	0	355.47	355.47	1066.41
MINIMIPS	72	18.402	594.11	594.09	5346.81
LEON II	192	27.625	992.95	992.92	11915.01
L80SOC	14	9.928	17.32	17.31	571.21
HP16	47	15.693	217.30	217.27	1303.67
RS_DEC	78	23.479	96.43	96.41	1446.11
FPU	17	12.274	1642.19	1642.17	9853.04
DCT	6	9.402	269.63	269.61	2426.54

failures for the minority voter based TMR are more than that for the XNOR-based TMR. This is due to the nature of minority voter which requires three inputs from three different domains. In fact this strong likelihood in terms of interconnections between the majority voters and the minority voters can result in a placement in the same CLB. Although, we constrained the placement in such a way that logic from different domains do not mix up, however, in a domain the placement of majority voter and minority voter is not constrained. This increases the probability that a CDE in the routing can result in an error being undetected when the minority voter is affected. This can cause a fault to linger longer in the TMR circuit resulting in the accumulation of faults and can lead to a functional failure. The use of XNOR gate reduces the probability of a fault being un-detected due to the fact that it uses two inputs from the same domain. Moreover, the carry-chain based detectors are non-programmable and are not controlled by SRAM cells thus enhancing the dependability.

The area comparison of the proposed approaches with X-TMR is shown in figure 5.6. For each benchmark circuit shown on the horizontal axis, the vertical axis plots the number of LUTs utilized by the design using a logarithmic scale. Consider the case of the CORDIC cores: both circuits have zero area overhead, while the percentage timing improvement is 67 percent. This is because both circuits have

Table 5.3. Critical SEUs-induced cross domain errors

Circuit	X-TMR		Minority-Voter based TMR		XNOR-based TMR (proposed scheme)	
	CLB	Routing	CLB	Routing	CLB	Routing
RISC5x	22	32	0	13	0	6
CORDICR2P	20	84	0	22	0	12
CORDICP2R	32	932	0	76	0	42
MINIMIPS	12	342	0	24	0	12
LEON II	52	1,230	0	88	0	47
L80SOC	31	86	0	10	0	0
HP16	28	109	0	21	0	14
RS_DEC	32	118	0	30	0	18
FPU	19	273	0	54	0	32
DCT	10	120	0	19	0	12

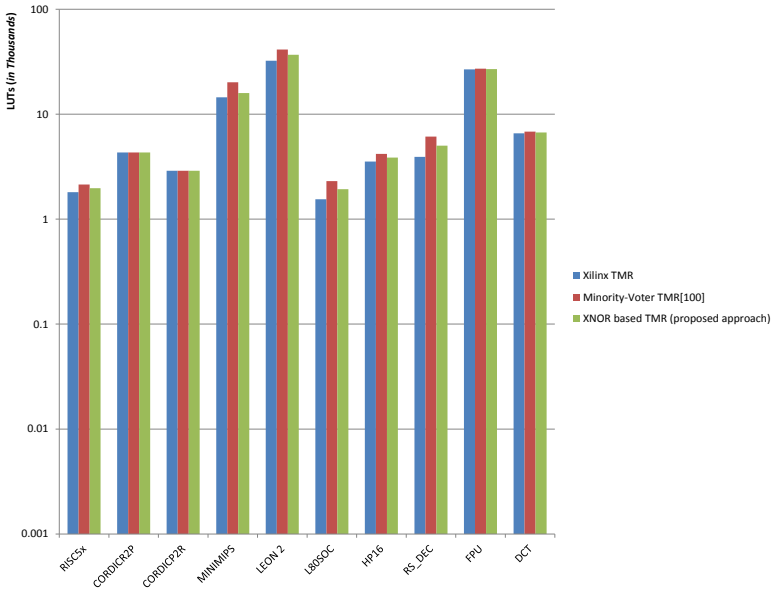

Figure 5.6. Area comparison of proposed TMR architecture with X-TMR

a single partition and the overhead in this case is just the one stemming from

the flag signals taken from the output of minority voters. For the case of two partitions, HP16 has more area overhead compared to FPU because the number of signals is comparable; however, the logic resources in FPU are far more than the logic resources of HP16. This means that for the same number of signal groups and partitions, for any two different designs our method will result in less area overhead for the larger design. Moreover, for two designs with roughly the same area (e.g., RISC5x and L80SOC) our approach will result in a larger area overhead for the design with the larger number of partitions. Concretely, the area overhead is dependent directly on the number of signal groups and on the number of partitions. Also the logic distribution of resources and area of the original circuits have impact on the area overhead of our method. As discussed in detail in section 5.3.1, the single SEU fault assumption, enable us to remove the steering logic in the minority voter based schemes, therefore, the proposed approach has a less area overhead than the approach proposed in [100]. The effects of the proposed methodology on the operating frequency of the benchmark circuits are shown in figure 5.7. These results

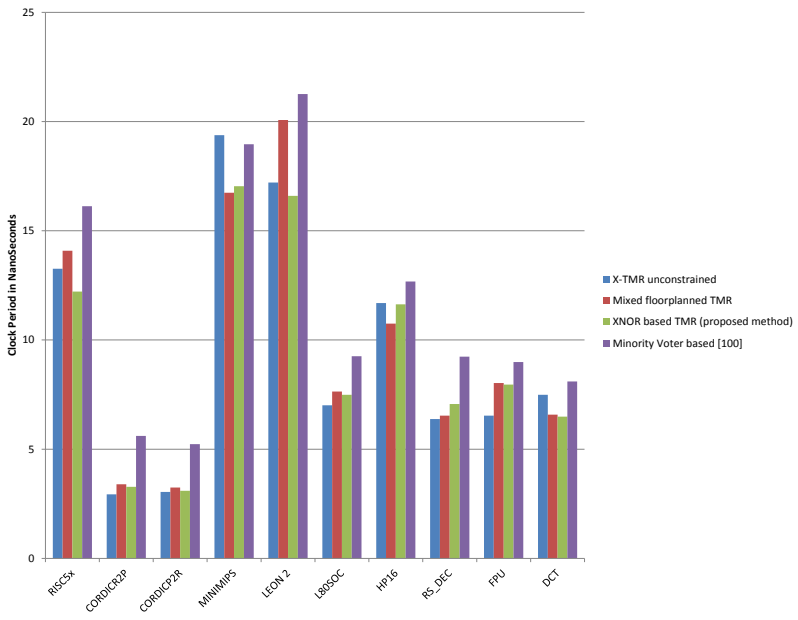

Figure 5.7. Effects of proposed methodology on clock period

were obtained by analyzing the post place and route net-list with the Xilinx timing analyzer tool. For each benchmark circuit the figure shows the clock period for Xilinx TMR, XNOR based mixed floor-planned, XNOR based non-overlapping and minority voter based non-overlapping versions. The plot shows, on the vertical axis the clock period (in nanoseconds) and the corresponding benchmark circuit on the horizontal axis. The Xilinx TMR version is implemented with standard tool-flow

and is not constrained in any way. The mixed floor-planned version has the XNOR based error detectors inside but the domains overlap. The minority voter version utilized the physical-level flow and uses a simulated annealing placement algorithm. The custom simulated annealing placer has a poor performance compared to the Xilinx placer utilized by the other three versions. In contrary to our intuition, the floor-planning of domains has enhanced the performance of some of the benchmark circuits. The reason for this performance improvement can be contributed to the fact that the manual floorplan, done in an optimal manner for most benchmark circuits, enables the placer to more rigorously optimize the wire-length due to compact area constraints. It is also very interesting to note that the insertion of very long error detectors in the TMR circuit does not affect the circuit operating frequency. This is due to the usage of dedicated hardwired carry-chains connections, designed for fast arithmetic computations, that never changed the critical path of the circuits.

Chapter 6

ASIC Fault Emulation

Hardware fault emulation for Application Specific Integrated Circuits (ASICs) on FPGAs can considerably reduce the time required for the fault simulation. This chapter presents a methodology to emulate ASIC faults on state-of-the-art FPGAs. The fault emulation is achieved by following a fully automated process consisting of: constrained technology mapping of ASIC net-list; creation of fault dictionary, generation of faulty partial bit-streams and fault emulation. The proposed approach exploits run-time partial reconfiguration techniques for fault injection and avoids full net-list re-compilations. The methods feasibility is assessed through carefully selected circuits and overhead in terms of area and timing is reported.

6.1 Motivation

As discussed in previous chapters that the circuit instrumentation approaches for fault emulation modifies the original net-list by inserting extra- hardware elements for fault injection purposes. In contrast, the reconfiguration-based techniques change the configuration bit-stream to inject faults and therefore do not incur hardware overhead making reconfiguration based approaches are very attractive. The challenge is how to fault emulate every ASIC fault with a corresponding faulty configuration bit-stream for the FPGA. For fault emulation of ASICs on FPGA, it is necessary to consider the types of technology nodes both utilize. ASIC net-lists are composed of gates in a specific standard cell library while FPGA net-lists comprise of Look Up Tables (LUTs). LUTs can implement any combinational Boolean function of k inputs where k is the maximum number of inputs to the LUT. To optimize area (usually measured in number of LUTs) and delay, during mapping of the ASIC net-list on FPGA, the FPGA CAD tools changes the structure of the original ASIC

net-list; clearly, this process is not affordable for fault emulation purposes, since it may lead to structural differences in the considered circuits, modifying the actual faults to be emulated. A possible solution may rely on partitioning the ASIC net-list to a set of circuit chunks suitable to be placed into the FPGA LUTs; however, this division process is hard to obtain by using commercial FPGA tools. Therefore, a known technology mapping from the ASIC net-list to FPGA net-list is mandatory to maintain the same circuit structure, that guarantees the actual emulation of all the ASIC faults. The authors in [88] used a constrained technology mapping exploiting a commercial fault emulation platform for serial fault emulation. This technique results in a known logic mapping for each gate of the original ASIC gates. Resultantly, the fault emulation for each ASIC fault with a corresponding LUT configuration string is made possible. However, the technique was applied using a commercial fault emulation platform requiring vendor specific tool-chain. This chapter presents an efficient methodology for the stuck-at fault emulation of ASICs on commercial FPGAs. The methodology starts from an ASIC synthesized net-list and applies a constraint technology mapping to get the circuit without any modification into an FPGA net-list comprising of LUTs. Then, a fault dictionary generator automatically computes all the faulty LUT strings. Consequently, a partial bit-stream is generated for the every fault in the obtained dictionary, which is used later for fault emulation on FPGA. Our methodology enables the run-time injection of faults using dynamic partial reconfiguration and avoids lengthy recompilation times. Experimental results gathered on a set of representing circuits show that the approach is able to achieve a fault emulation speedup compared with software fault simulation while guaranteeing the emulation of every ASIC fault with a corresponding FPGA configuration string.

6.2 Fault Emulation on FPGAs

The emulation of ASIC faults on FPGA requires the knowledge of the FPGA's resource that can be exploited for fault injection. This section outlines the generic architecture of FPGA devices that may be exploited during fault emulation. In particular, combinatorial logic fault injection based on LUT reconfiguration, as well as, flip-flop's fault injection techniques are summarized. In addition, some considerations about the relationship between technology mapping and its effects on fault list are described in details.

6.2.1 Fault Emulaiton in LUTs

FPGA consists of tiles of different primitive types. For example, a tile can be made of Configurable Logic Blocks (CLBs), DSPs, BRAMs, IOBs and Interconnect Tile. The CLBs are the workhorse of FPGAs for implementing combinational and sequential elements of a circuit. Each CLB consists of a number of function generators in form of LUTs. Within each CLB, flip-flops reside in close proximity to LUTs. Each CLB can connect to global horizontal and vertical interconnect lines using the Interconnect Tile located near to it. For a LUT with k inputs the maximum number of configurations is 2^{2^k}. Figure 6.1 shows an abstract architectural view of a part of CLB and its Interconnect Tile. For illustration purposes only one LUT and FF is shown in the figure 6.1. It can be noted that the LUT implements a six input

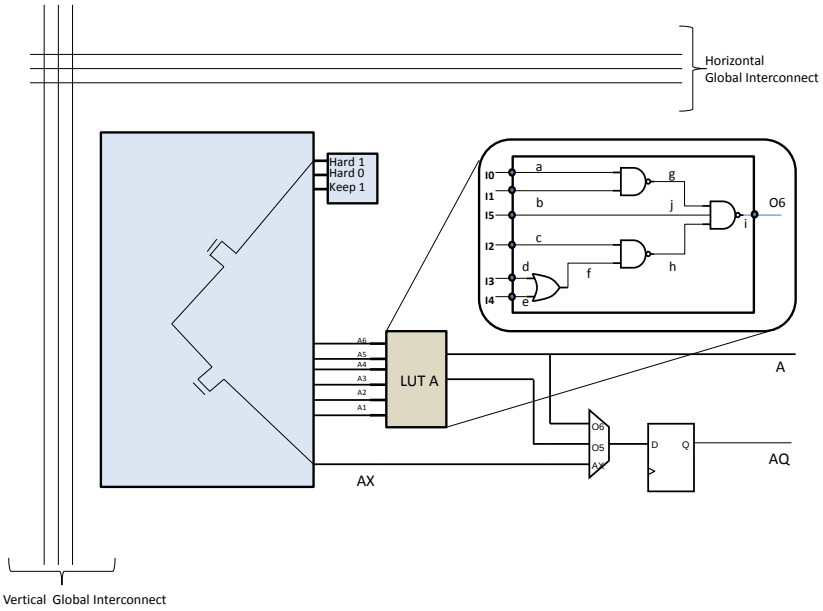

Figure 6.1. Fault emulation in LUTs and FFs

function as shown in equation (6.1)

$$O6 = ((I0\neg \wedge I1)\neg \wedge I5)\neg \wedge (I2\neg \wedge (I3 \vee I4)) \tag{6.1}$$

If this equation represents a logic cone that belongs to the original ASIC net-list then, it is possible to emulate ASIC equivalent faults on the structure through a corresponding LUT equation modification. For example, for net f in figure 6.1, a

stuck-at zero corresponds to LUT equation (6.2)

$$O6 = ((I0\neg \wedge I1)\neg \wedge I5)\neg \wedge (1) \tag{6.2}$$

6.2.2 Fault Emulation in FFs

Flip-flop faults emulation, on the other hand, may be handled resorting to multiple scenarios according to the circuit structure. Figure 6.1 shows a FF that can be fed by the *O6* output of a LUT, *O5* output of a LUT or with the auxiliary input line *AX*. For emulating a fault at the input of the FF, a LUT or an Interconnect Tile can be used. When the FF is connected to a single LUT within the same CLB, the LUT equation can be modified accordingly to the considered fault model to emulate the fault at the input of the FF. However, when the LUT output feeds multiple sinks including a FF, this strategy is not suitable. In this case, it is possible to use the *GROUND* and *VCC* resources present near the Interconnect Tile of each CLB, by connecting the *AX* to the input of the FF using the Interconnect Tile and the configurable multiplexer available within each CLB.

6.2.3 Technology Mapping

FPGA CAD tools significantly differ from ASIC tools because the underlying technology both use is very different. Emulating ASIC faults using a FPGA based method requires maintaining the structure of the ASIC net-list. However, resorting to standard FPGA tools may result in different circuit structure since these tools usually make several logic optimizations modifying the ASIC net-list. The mapping of ASIC gates to LUTs, is usually referred to as K-LUT technology mapping. This process may duplicate part of the original ASIC gates for reducing the number of LUTs or reducing delay or both [106]. Another optimization uses functional decomposition of ASIC gates to smaller gates for reducing the number of LUTs [107]. However, it is not possible to know how the FPGA tools partitioned the ASIC net-list for mapping to LUTs. In any case, the different approaches for K-LUT technology mapping combine multiple ASIC gates into one FPGA LUT, while maintaining the circuit flip-flops. Thus, it is possible to exploit the similarity between the two net-lists for fault emulation purposes. However, for guaranteed fault emulation (i.e., every ASIC fault has a corresponding LUT configuration) the behavior of both the ASIC circuit and the FPGA model must be the same for all possible inputs. Using this criteria for finding the equivalent LUT configuration for each ASIC fault using

commercial software CAD tools for FPGA is hugely time consuming. Consider the case when multiple LUTs reside in a combinational logic between two flip-flops. For finding the equivalent LUT configuration for an ASIC fault the size of search space is equation (6.3).

$$N_{PIs} * (N_{LUTs} * 2^{2^k}) \tag{6.3}$$

Where N_{LUTs} represents the number of LUTs that resides in the combination logic between two flip-flops, N_{PIs} is the k number of primary inputs and 2^{2^k} is the number of possible configuration for each LUT. Obviously, for guaranteeing fault emulation this is prohibitively time consuming task, and diminishes the returns of hardware emulation when all faults are considered. The search space for this problem can be limited by using the active number of inputs to the LUT as presented in [93] [94]. However, for guaranteed fault emulation it is still excessively large. Therefore, it is necessary to use a custom technology mapping that avoids changes to net-list and results in a known mapping of ASIC gates to FPGA LUTs.

6.3 Proposed Methadology

In this section, we discuss in detail the proposed methodology for ASIC fault emulation on state of the art Xilinx FPGAs. The methodology consists of two main phases: a custom technology mapping of the ASIC net-list to LUT-level FPGA net-list, and the creation of a fault dictionary as shown in figure 6.2. The developed tools use Boost C++ libraries [60], and builds upon the framework presented in [58]. The following paragraphs explain in details all the phases involved.

6.3.1 Custom technology mapping

The goal of this step is to translate an ASIC gate-level net- list to a LUT-level FPGA net-list suitable for fault emulation. This problem of converting the ASIC gates to LUTs is usually called K-LUT technology mapping. The existence of multiple fan-out nodes makes the problem of optimal technology mapping very challenging [108]. Figure 6.3 shows an ASIC net-list with a couple of multiple fan-out gates. The circuit is then mapped to three FPGA LUTs, represented in figure 6.3 by the boxes labeled as *LUT1*, *LUT2* and *LUT3*. This technology mapping uses the concept of Maximum Fan-out Free Cones (MFFCs) [109]. The MFFC of a node v represents the maximum number of nodes in the transitive fan-in of a gate in such a way that

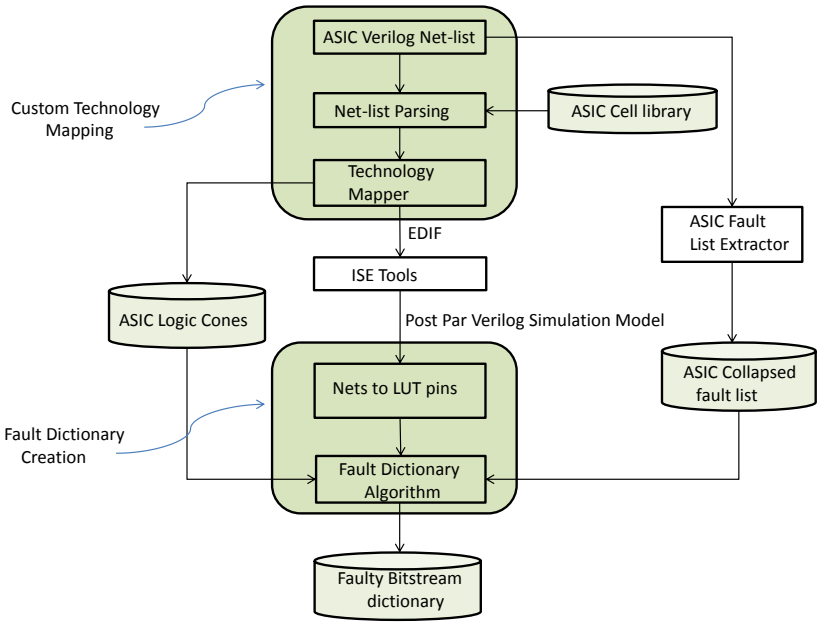

Figure 6.2. Constrained Technology Mapping and Fault Dictionary Creation Flow

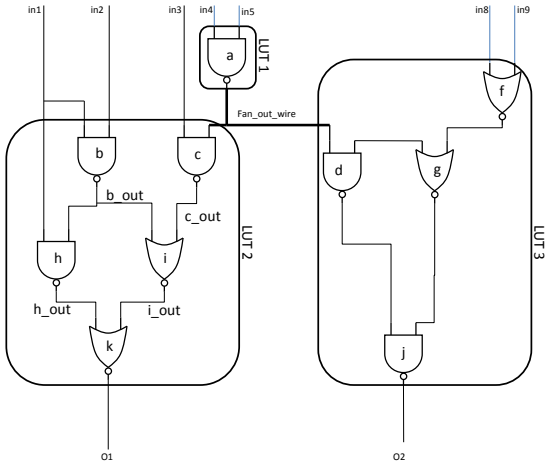

Figure 6.3. Mapping without duplication

the fan-out of every node in the MFFC except the node v is inside the MFFC. For example, the MFFC of node k represented by MFFC_k is composed of all the gates in the transitive fan-in h,i,b,c except the node a because it fan-outs to MFFC_j. It is interesting to note that the gate b has also multiple fan-out but they re-converge on gate k and therefore are a part of MFFC_k. If the MFFC is k-feasible (i.e., the number of inputs are less than or equal to k, the maximum number of inputs to a

LUT) it can be collapsed into a LUT. All the MFFCs in figure 6.4 are k-feasible and therefore are a candidate for a LUT. The advantage of MFFC based mapping is that emulating a fault on a multiple fan-out node requires reconfiguring only a single LUT and thus can exploit collapsed ASIC fault list to reduce the fault emulation time. However, the required number of LUTs can be reduced from three to two if duplication is allowed as shown in figure 6.4. It can be noted that node *a* has been duplicated. For fault emulation purposes every ASIC fault on the structure of node *a* should be emulated in both LUTs. With state of the art ASIC synthesis tools it is often the case that a gate fan-outs to more than two gates. Resultantly, allowing duplication during technology mapping would mean that a single ASIC fault is converted to multiple FPGA faults. In the proposed approach, the presented mapping process uses MFFC duplication-free mapping of the ASIC net-list avoiding nodes duplication in the resulting LUTs. In order to create the actual LUT-level net-list, it is required to identify the initial values every LUT is going to assume. Let us describe how this process is performed for the *LUT 2* in figure 6.3. The input nets *in1*, *in2*, *in3* and *fanoutwire* are arbitrarily bound to the LUT address lines *I0*, *I1*,*I2* and *I3*. For a 6-input LUT, the address lines can be represented as 64-bit vectors formed by concatenating the bits along the truth table from top to bottom. These address lines can be used to calculate the LUT function (often called LUT INIT value) by walking the LUT's DAG and applying the corresponding Boolean operations in the following as show in equation from (6.4) to (6.8) .

$$\begin{aligned} \texttt{I0} &= \texttt{AAAAAAAAAAAAAAAA} \\ & \qquad\qquad (\text{nand}) \\ \underline{\texttt{I1} &= \texttt{CCCCCCCCCCCCCCCC}} \\ \texttt{bout} &= \texttt{7777777777777777} \end{aligned} \tag{6.4}$$

$$\begin{aligned} \texttt{I2} &= \texttt{F0F0F0F0F0F0F0F0} \\ & \qquad\qquad (\text{nand}) \\ \underline{\texttt{I4} &= \texttt{FFFF0000FFFF0000}} \\ \texttt{cout} &= \texttt{0F0FFFFF0F0FFFFF} \end{aligned} \tag{6.5}$$

Now for calculating the values of next level of gates, we can utilize equation (6.4) and *I0*

$$\begin{aligned} \texttt{bout} &= \texttt{7777777777777777} \\ & \qquad\qquad (\text{nand}) \\ \underline{\texttt{I0} &= \texttt{AAAAAAAAAAAAAAAA}} \\ \texttt{hout} &= \texttt{DDDDDDDDDDDDDDDD} \end{aligned} \tag{6.6}$$

For gate i using equations (6.4) and (6.5)

$$\begin{aligned} &\texttt{cout = 0F0FFFF0F0FFFFF} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\text{(or)} \\ &\underline{\texttt{bout = 7777777777777777}} \\ &\texttt{iout= 7F7FFFFF7F7FFFFF} \end{aligned} \tag{6.7}$$

The LUT INIT value is the following

$$\begin{aligned} &\texttt{iout = 7F7FFFFF7F7FFFFF} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\text{(nand)} \\ &\underline{\texttt{hout = DDDDDDDDDDDDDDDD}} \\ &\texttt{INIT = A2A22222A2A22222} \end{aligned} \tag{6.8}$$

Using this approach a LUT initialization value is calculated and the resultant description is translated to the required format to be mapped into the FPGA. The standard FPGA CAD steps are performed for the implementation of the golden version, followed by bitstream generation phase.

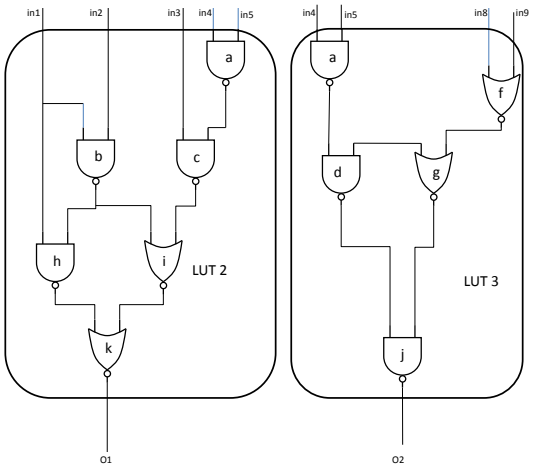

Figure 6.4. Mapping with duplication

6.3.2 Fault Dictionary Creation

The purpose of the fault dictionary creation phase is to generate partial bitstreams for all the faults in the collapsed ASIC fault list. The ASIC fault list may be derived

using a commercial fault simulator. The proposed algorithm, called Fault Dictionary Algorithm (FDA) requires a post placed and routed FPGA simulation model, as well as the ASIC logic cones data-base generated during custom technology mapping and the collapsed ASIC fault list. For every ASIC logic cone represented as a Directed Acyclic Graph (DAG), the FDA creates a corresponding faulty LUT equation as shown in Algorithm 6.1. The algorithm utilizes a map data-structure called *VMap*

Algorithm 6.1 Fault Dictionary Creation (FDA) algorithm

Input: ASIC Logic Cones and ASIC Collapsed Fault List
Output: Faulty Bitstream Dictionary

```
1: //******* Nets to LUT pins binding phase ****************
2: while all PIs for current LUT not assigned do
3:     AddressLineHexVal = RouterUsedAddressLine(PI);
4:     VMap[PI] = AddressLineHexVal;
5: end while
6: //*** Generating Faulty Equations for a LUT DAG *********
7: foreach NET ϵ LUT DAG do
8:     foreach faultCondition on NET do
9:         // Assign the fault value to faulty net
10:        if (faultCondition on NET ϵ ASIC faultlist) then
11:            VMap[NET] = FaultyHexValue;
12:        end if
13:        // Walk the LUT DAG to calculate faulty equation
14:        foreach v ϵ TopologicalSort of DAG Nodes do
15:            foreach INet ϵ inputNet(v) do
16:                VMap[ONet(v)] = VMap[ONet(v)] Opr VMap[INet];
17:            end foreach
18:        end foreach
19:        // Saving the mapping from ASIC to FPGA faults
20:        Write_to_FD[ASICfault] = VMap[LUTDAG_ONet];
21:    end foreach
22: end foreach
```

which is indexed by a net and the value stored is the current Boolean equation for that net during the topological graph traversal. The Boolean equation is generated by traversing the graph in forward topological order from primary inputs to primary outputs and at the output of each logic gate, the type of operation represented by the gate is applied to the values indexed by the input nets. The resulting value is stored in the map data-structure indexed by the output net. For nets which are currently at a stuck-at for fault emulation purposes the computed values during graph traversal is ignored and the stuck-at fault equivalent value is stored in the

map data-structure. The resulting equation is generated in a format compatible with Xilinx Design Language (XDL). It is important to retain LUT address lines even in cases where a fault may make the resultant equation independent on some particular inputs. For example, in figure 6.1, a stuck-at one fault on g net makes the whole LUT equation independent on *I0* and *I1* address lines; however, it is important to keep these signals in the equation so that FPGA routing is kept intact and drastic changes to the net-list are avoided. The resultant equation for *gstuckat1* is shown in equation (6.9)

$$O6 = (((I3 \vee I4)\neg \wedge I2)\neg \wedge (I5)\neg \wedge ((I0 \vee \neg(I0)) \wedge (I1 \vee \neg(I1)))) \tag{6.9}$$

Note that, in the equation, *I0* and *I1* are there only to avoid re-routing, otherwise the function is independent on these due to the faulty condition on g. Following this methodology for each fault, a collection of partial bitstream reconfiguration files are generated, which are used later in the actual fault injection process. Finally, the golden circuit, as well as the faulty one is mapped in the same FPGA and the test patterns may be applied and the behavior on both of the circuits compared considering every fault.

6.4 Experimental Results

The experimental results were collected on a Xilinx Virtex- 5 VLX110T FPGA emulating the stuck-at faults for every ASIC benchmark circuit. The methodology is applied to 5 circuits including 3 arithmetic circuits and 2 DSP circuits. A Micro-blaze based version of the platform presented in [110] was adopted for test pattern application to the Design Under Test (DUT). The DUT has the custom technology mapped ASIC net-lists in two version i-e the golden copy and the faulty copy. The DUT also contains a hardware comparator that compares the output from both copies to detect a faulty condition. The DUT is attached as an IP core to the Micro-blaze processor for test pattern application and monitoring. The FDA algorithm's generated faulty LUT equations are inserted into XDL file of the whole Micro-blaze system to generate valid partial bit-streams. The stored partial bit-streams are downloaded one by one from a personal computer to the FPGA. The processor then starts to apply the test patterns that were derived from the commercial fault simulator stored in an off-chip memory. The results of fault detection are communicated back to the host PC which proceeds with the injection of another partial faulty bit-stream. Another set of experiment utilizing the post-Placed and Routed Simulation model used ModelSim for fault simulation of the ASIC net-list,

Xilinx mapped net-list and custom mapped net-list to show the relative difference in speeds. It is important to note that the test patterns generated with commercial fault simulator stored in STIL format are used with ModelSim. The fault injection on the ASIC net-list is achieved with ModelSim *force freeze* command while for injecting LUT faults in FPGA net-list, re-compilation with ModelSim -*G* switch is performed. Therefore, the simulation timing required for FPGA is more than that of ASIC fault injection due to re-compilation requirements. The fault simulation time $T_{modelsim}$ is calculated by using ModelSim *simstats* command with an automated TCL script. The actual timing requirements of our custom mapper using hardware emulation are given by T_{fpga} which is calculated in the following way. Let T_{comp} be the time required for compilation of the golden version of the circuit, T_{config} be the bit-stream download time, N_{cf} be the number of collapsed faults required for ASIC fault simulation, $T_{reconfig}$ be the fault injection or fault removal time, T_{pd} is the propagation delay of the critical path in case of combination circuit or the clock period length in case of sequential circuit and N_p are the average number of patterns required for detection of faults in a benchmark circuit. Then we can write the total fault emulation time considering serial fault emulation in show in equation (6.10).

$$T_{fpga} = T_{config} + T_{comp} + N_{cf} * (T_{reconfig} + T_{pd} * N_p) + T_{reconfig} * N_{LUTs} \quad (6.10)$$

It can be noted that the compilation time and configuration time are required only once and therefore they can be neglected. The reconfiguration time depends on the number of LUTs because for emulating faults in a single LUT there is no need to remove the fault before injecting another one but if the next fault is to be injected in another LUT then the previous faulty LUT should be restored to the golden LUT configuration value. The product of T_{pd}*Np is the time required for fault emulation of each fault. Table 6.1 shows the comparison between the fault simulation and fault emulation times for ASIC and FPGA netlists. Table 6.2 presents the fault statistics from the original ASIC net-list using commercial fault simulator Tetra-Max. The original ASIC design was synthesized with Design Vision for pdt2002 library. It should be noted that because of the known technology mapping we are able to directly use the values from Table 6.2 and we achieve the same coverage for the stuck-at fault model given by the Tetra-Max on the given benchmark circuits. Although our custom technology mapping based fault emulation flow enables guaranteed emulation of every ASIC fault in the ASIC fault list with a reduction in fault simulation timing requirements, however, this is achieved at an overhead in terms of area and delay of the circuit as reported in Table 6.3. It can be noted that due to MFFC based mapping our mapper utilizes more LUTs and runs more slowly than the Xilinx mapper.

Table 6.1. Fault Simulation and Emulation Time Comparison

Circuit	ASIC $T_{modelsim}$ (sec)	FPGA		
		Xilinx Mapper $T_{modelsim}$ (sec)	Custom Mapper	
			$T_{modelsim}$ (sec)	T_{fpga} (secs)
Adder64	7.136	120.364	21.503	0.004
Multiplier32	158.87	10,124.09	351.922	0.271
Multiplier64	618.71	1506,061.21	1,418.995	3.732
FIR_filter	120.12	24,702.26	418.116	0.058
Hilbert_trans	29.15	796.314	92.687	0.012

Table 6.2. Fault Statistics for ASIC netlist

Circuit	Total Faults	N_{cf}	N_p	Coverage %
Adder64	2,436	1,410	34	100.00
Multiplier32	42,304	23,431	137	99.81
Multiplier64	170,612	94,125	288	99.87
FIR_filter	46,696	27,873	14	98.00
Hilbert_trans	11,168	6,179	40	97.20

Table 6.3. Area and Overhead delay

Circuit	Xilinx Mapper			Custom Mapper		
	LUTs	FFs	Delay	LUTs	FFs	Delay
Adder64	96	0	29.81	111	0	34.86
Multiplier32	1,624	0	43.40	2,679	0	71.19
Multiplier64	7,110	0	87.68	11,114	0	131.32
FIR_filter	2,052	971	18.17	2,396	971	26.33
Hilbert_trans	288	225	3.07	387	225	5.99

Chapter 7

Conclusions

This dissertation focus on the usage of SRAM-based FPGAs for design, implementation and evaluation of dependable systems. Reducing the time required for fault injection and fault correction along with effective mitigation of MBUs in the configuration memory are the core challenges that this thesis undertook. These problems are particularly relevant as there is an increasing trend in reconfiguration times and multiple bit errors with growing device densities and dimensional scaling in the latest generation of FPGAs. The potential approach for tacking the research challenges mentioned before are based on granularity of redundancy and reconfiguration.

As the first case study a self-repairing system is investigated which contains a static and a dynamic region. The static region contains a processor and fixed resources while the dynamic region contains the Design Under Test (DUT). A fine-grain fault detection mechanism is used in DUT region while fine-grain correction mechanism is carried out by the static region in the proposed system. The approach is characterized by the capabilities of detecting and correcting errors induced by single and multiple upsets affecting the FPGA configuration memory. The approach exploits the available carry chains and the hardwired extra logic to perform the error detection with error detection latency in order of nanoseconds. For example, detector employed in single bit error region has a detection latency of approximately 60 nanoseconds for a detector length of 20 CLBs while detectors in multiple bit region have delay latency of less than 5 nanoseconds. It has been observed that there is a considerable variable in the error detection latency due to routing congestion in the local vicinity and due to the design complexity. The effectiveness of our approach has been evaluated with fault injection campaigns demonstrating that our approach is able to detect and correct more than 98% of the bit-flips, showing an improvement of more than one order of magnitude in terms of recovery time with respect

to traditional redundancy-based approaches. Moreover, our approach has a more limited area overhead in terms of required circuit resources.

The fault masking capabilities of TMR-based scheme enables the systems to operate without interruption, however, they require a periodic scrubbing to avoid the accumulation of upsets in the configuration memory of FPGA. With the dimensional scaling, low voltage thresholds and densely packed SRAM-cells, not only the probability of MBUs in the configuration memory is rising but also the scrubbing time is increasing as well. MBUs can create simultaneous faults in more than two TMR domains and thus break fault masking abilities, errors arising from this scenario are termed as CDEs. To simultaneously cope with theses problems a modified TMR architecture is presented in this thesis in order to improve the recovery time and fault tolerance of TMR circuits implemented on SRAM-based FPGAs. Two key modifications are proposed in the TMR architecture and in the placement phase. First, the TMR architecture is instrumented with custom logic at the granularity of individual domains to detect, localize and repair faulty domains. The huge number of comparator check flags generated are merged using the on-chip in-slice carry-chain resources. The carry-chain based detectors utilized in this case functions as an AND-tree and uses the hardwired internal routing therefore has significantly higher speed for fault propagation. As an example, a detector of length 192 CLBs has an error detection latency of 28 nanoseconds. Second, a novel CAD flow is used for insertion of custom logic and non-overlapping placement of TMR domains. The results shows more than 80% reduction in recovery time compared to X-TMR. Similarly, the number of CDEs in LUTs reduce to zero while a significant reduction in routing errors is observed. The recovery time is bounded by the size of maximum partition in a TMR circuit and is orders of magnitude less that the full scrubbing. The area overhead depends on the number of signals crossing the boundaries between any two partitions, on the number of partitions, on the number of logic resources in the circuit, and on its distribution across partitions. Interestingly, the method does not incur performance overheads due to the insertion of error detection logic. An analytical reliability evaluation method is employed for early design phase evaluation in this thesis; however, in the future it would be interesting to investigate radiation effects under particle accelerator.

After considering dependable system design techniques which strives to improve fault tolerance and availability capabilities for circuit mapped on SRAM-based FPGAs, fault simulation of custom ICs is investigated which is a dependability evaluation technique. The main motivation of this task is to reduce the fault simulation times due to at-hardware speed processing on FPGAs. This thesis presented a methodology and a custom CAD flow for guaranteed emulation of ASIC permanent

faults using SRAM-based FPGAs. The flow utilizes a custom technology mapping for directly converting the post layout gate-level net-list into a LUT level net-list. This known mapping of gates to LUTs enables us to develop a fault dictionary from the ASIC fault list to FPGA partial bitstreams. The methodology avoids drastic changes to the net-list, therefore, does not need lengthy re-compilation times during the fault emulation process. Furthermore, our experimental results demonstrate a significant speed up for fault emulation compared to software based fault simulation. In future, we would like to extend the methodology to encompass a wide set of other fault models and to extend the approach to automatic test pattern generation. It is also possible to optimize the LUT-mapping algorithm presented in this thesis to reduce the number of frames required for fault injection thereby improving the speed of fault injection further.

Bibliography

[1] G.E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, Jan 1998.

[2] P. Alfke, I. Bolsens, B. Carter, M. Santarini, and S. Trimberger. Its an fpga! *Solid-State Circuits Magazine, IEEE*, 3(4):15–20, Fall 2011.

[3] Steve Leibson and Nick Mehta. Xilinx ultrascale:the next-generation architecture for your next-generation architecture. Technical Report WP435(v1.1), 2014.

[4] Altera. The breakthrough advantage for fpgas with tri-gate technology. Technical Report WP-01201-1.0, 2013.

[5] V.S.V. Prabhakar K. Lal Kishore. *VLSI Design*. I K International Publishing House, 2011.

[6] K. Johansson, P. Dyreklev, B. Granbom, M.C. Calver, S. Fourtine, and O. Feuillatre. In-flight and ground testing of single event upset sensitivity in static rams. *Nuclear Science, IEEE Transactions on*, 45(3):1628–1632, Jun 1998.

[7] J. Olsen, P.E. Becher, P.B. Fynbo, P. Raaby, and J. Schultz. Neutron-induced single event upsets in static rams observed a 10 km flight attitude. *Nuclear Science, IEEE Transactions on*, 40(2):74–77, Apr 1993.

[8] R. Katz, K. LaBel, J.J. Wang, B. Cronquist, R. Koga, S. Penzin, and O. Swift. Radiation effects on current field programmable technologies. *Nuclear Science, IEEE Transactions on*, 44(6):1945–1956, Dec 1997.

[9] Radiation effects on sram-based fpgas. In *Electronics System Design Techniques for Safety Critical Applications*, volume 26 of *Lecture Notes in Electrical Engineering*, pages 17–45. Springer Netherlands, 2009.

[10] Dagan White. Considerations surrounding single event effects in fpgas, asics, and processors. Technical Report WP402 (v1.0.1), 2012.

[11] J.C. Knight. Safety critical systems: challenges and directions. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 547–550, May 2002.

[12] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts

and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004.

[13] J. Borecky, P. Kubalik, and H. Kubatova. Reliable railway station system based on regular structure implemented in fpga. In *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, pages 348–354, Aug 2009.

[14] Jinkge She. *Investigation on the Benefits of Safety Margin Improvement in CANDU Nuclear Power Plant Using an FPGA-based Shutdown System* . PhD thesis, School of Graduate and Postdoctoral Studies, The University of Western Ontario London, Ontario, 2012.

[15] J. Henaut, D. Dragomirescu, and R. Plana. Fpga based high date rate radio interfaces for aerospace wireless sensor systems. In *Systems, 2009. ICONS '09. Fourth International Conference on*, pages 173–178, March 2009.

[16] M. Lanuzza, P. Zicari, F. Frustaci, S. Perri, and P. Corsonello. Exploiting self-reconfiguration capability to improve sram-based fpga robustness in space and avionics applications. *ACM Trans. Reconfigurable Technol. Syst.*, 4(1):8:1–8:22, December 2010.

[17] Xilinx Inc. *Partial reconfiguration user guide.*

[18] Umer Farooq, Zied Marrakchi, and Habib Mehrez. Fpga architectures: An overview. In *Tree-based Heterogeneous FPGA Architectures*, pages 7–48. Springer New York, 2012.

[19] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, editors. *Architecture and CAD for Deep-Submicron FPGAs.* Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[20] Katherine Compton and Scott Hauck. Reconfigurable computing: A survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, June 2002.

[21] D. Dickin and L. Shannon. Exploring fpga technology mapping for fracturable lut minimization. In *Field-Programmable Technology (FPT), 2011 International Conference on*, pages 1–8, Dec 2011.

[22] Ju-Yueh Lee, Yu Hu, R. Majumdar, Lei He, and Minming Li. Fault-tolerant resynthesis with dual-output luts. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 325–330, Jan 2010.

[23] A. Das, S. Venkataraman, and A. Kumar. Improving autonomous soft-error tolerance of fpga through lut configuration bit manipulation. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–8, Sept 2013.

[24] A. Singh and K. Chaudhary. Pin reordering during placement of circuit designs, June 6 2006. US Patent 7,058,915.

[25] Xilinx Inc. Xcell journal virtex-5 special edition. Technical report, 2006.

[26] M.B. Tahoori and S. Mitra. Automatic configuration generation for fpga interconnect testing. In *VLSI Test Symposium, 2003. Proceedings. 21st*, pages

134–139, April 2003.

[27] X. Iturbe, K. Benkrid, R. Torrego, A. Ebrahim, and T. Arslan. Online clock routing in xilinx fpgas for high-performance and reliability. In *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*, pages 85–91, June 2012.

[28] Ritesh Kumar Soni, Neil Steiner, and Matthew French. Open-source bitstream generation. In *Proceedings of the 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '13, pages 105–112, Washington, DC, USA, 2013. IEEE Computer Society.

[29] A library for low-level manipulation of partially placed-and-routed fpga designs. Technical report, 2014.

[30] Xilinx Inc. *Virtex 5 FPGA configuration user guide*. Xilinx.

[31] Luka Daoud, Dawid Zydek, and Henry Selvaraj. A survey of high level synthesis languages, tools, and compilers for reconfigurable high performance computing. In Jerzy SwiÄ?tek, Adam Grzech, PaweÅ? SwiÄ?tek, and Jakub M. Tomczak, editors, *Advances in Systems Science*, volume 240 of *Advances in Intelligent Systems and Computing*, pages 483–492. Springer International Publishing, 2014.

[32] Giovanni De Micheli. *Synthesis and Optimzation of Digital Circuits*. McGraw-Hill Science/Engineering/Math; 1 edition (January 1, 1994), 1994.

[33] Russell Tessier. Fast placement approaches for fpgas. *ACM Trans. Des. Autom. Electron. Syst.*, 7(2):284–305, April 2002.

[34] Keheng Huang, Yu Hu, and Xiaowei Li. Reliability-oriented placement and routing algorithm for sram-based fpgas. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22(2):256–269, Feb 2014.

[35] Luca Sterpone. *Electronics System Design Techniques for Safety Critical Applications*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[36] Larry McMurchie and Carl Ebeling. Pathfinder a negotiation based performance driven router for fpgas. In *Proceedings of the 1995 ACM Third International Symposium on Field programmable Gate Arrays*, FPGA 95, pages 111–117, New York, NY, USA, 1995. ACM.

[37] L. Sterpone and M. Violante. A new reliability-oriented place and route algorithm for sram-based fpgas. *Computers, IEEE Transactions on*, 55(6):732–744, June 2006.

[38] Cristian Constantinescu. Impact of deep submicron technology on dependability of vlsi circuits. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, pages 205–209, Washington, DC, USA, 2002. IEEE Computer Society.

[39] M.A. Alam, K. Roy, and C. Augustine. Reliability- and process-variation aware design of integrated circuits; a broader perspective. In *Reliability*

Physics Symposium (IRPS), 2011 IEEE International, pages 4A.1.1–4A.1.11, April 2011.

[40] Alvin W. Strong, Ernest Y. Wu, Rolf-Peter Vollertsen, Jordi Sune, Giuseppe La Rosa, and Timothy D. Sullivan. *Reliability Wearout Mechanisms in Advanced CMOS Technologies*. John Wiley Sons, 2006.

[41] Nicolaidis Michael. *Soft Errors in Modern Electronic Systems*. Springer, 2011.

[42] K. Ramakrishnan, S. Suresh, N. Vijaykrishnan, M.J. Irwin, and Vijay Degalahal. Impact of nbti on fpgas. In *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, pages 717–722, Jan 2007.

[43] P. Blain C. Carmichael N. Khalsa E. Fuller, M. Caffrey and A. Salazar. Radiation test results of the virtex fpga and zbt sram for space based reconfigurable computing. In *Military & Aerospace Applications of Programmable Logic Devices,Laurel MD*, 1999.

[44] M. Ceschia, M. Bellato, A. Paccagnella, and A. Kaminski. Ion beam testing of altera apex fpgas. In *Radiation Effects Data Workshop, 2002 IEEE*, pages 45–50, 2002.

[45] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham. The reliability of fpga circuit designs in the presence of radiation induced configuration upsets. In *Field-Programmable Custom Computing Machines, 2003. FCCM 2003. 11th Annual IEEE Symposium on*, pages 133–142, April 2003.

[46] M. Abramovici, C.E. Stroud, and J.M. Emmert. Online bist and bist-based diagnosis of fpga logic blocks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(12):1284–1294, Dec 2004.

[47] C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici. Bist-based diagnosis of fpga interconnect. In *Test Conference, 2002. Proceedings. International*, pages 618–627, 2002.

[48] M. Ceschia, M. Violante, M.S. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori. Identification and classification of single-event upsets in the configuration memory of sram-based fpgas. *Nuclear Science, IEEE Transactions on*, 50(6):2088–2094, Dec 2003.

[49] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui. Radiation-induced multi-bit upsets in sram-based fpgas. *Nuclear Science, IEEE Transactions on*, 52(6):2455–2461, Dec 2005.

[50] G.L. Nazar and L. Carro. Exploiting modified placement and hardwired resources to provide high reliability in fpgas. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 149–152, April 2012.

[51] L. Sterpone, M. Porrmann, and J. Hagemeyer. A novel fault tolerant and runtime reconfigurable platform for satellite payload processing. *Computers,*

IEEE Transactions on, 62(8):1508–1525, Aug 2013.

[52] David Dye. Partial reconfiguration of xilinx fpgas using ise design suite. Technical Report WP374 (v1.2), 2012.

[53] J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Enhanced fpga reliability through efficient run-time fault reconfiguration. *Reliability, IEEE Transactions on*, 49(3):296–304, Sep 2000.

[54] Nazar GL. *Fine-Grain Error Detection Techniques for Fast Repair of FPGAs*. PhD thesis, Universidade Federal do Rio Grande do Sul, 2013.

[55] openfpga.

[56] Jason Luu, Jeff Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Norrudin Ahmed, Kenneth B. Kent, Jason Anderson, Jonathan Rose, and Vaughn Betz. VTR 7.0: Next Generation Architecture and CAD System for FPGAs. volume 7, pages 6:1–6:30, June 2014.

[57] Christopher Lavin, Marc Padilla, Jaren Lamprecht, Philip Lundrigan, Brent Nelson, and Brad Hutchings. RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs. In *Proceedings of the 21th International Workshop on Field-Programmable Logic and Applications (FPL'11)*, September 2011.

[58] Neil Steiner, Aaron Wood, Hamid Shojaei, Jacob Couch, Peter Athanas, and Matthew French. Torc: Towards an open-source tool flow. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, pages 41–44, New York, NY, USA, 2011. ACM.

[59] C. Beckhoff, D. Koch, and J. Torresen. The xilinx design language (xdl): Tutorial and use cases. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on*, pages 1–8, June 2011.

[60] Boost c++ libraries.

[61] Optimizing mitigation strategies for fpga critical applications. Technical report, 2010.

[62] Mahtab Niknahad. *Using Fine Grain Approaches for highly reliable Design of FPGA-based Systems in Space*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2013.

[63] J.R. Azambuja, F. Sousa, L. Rosa, and F.L. Kastensmidt. Evaluating large grain tmr and selective partial reconfiguration for soft error mitigation in sram-based fpgas. In *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*, pages 101–106, June 2009.

[64] Triple module redundancy design techniques for virtex fpgas. Technical Report xapp197, 2001.

[65] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis. A fault injection analysis of virtex fpga tmr design methodology. In *Radiation and Its Effects on*

Components and Systems, 2001. 6th European Conference on, pages 275–282, Sept 2001.

[66] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen. Domain crossing errors: Limitations on single device triple-modular redundancy circuits in xilinx fpgas. *IEEE Transactions on Nuclear Science*, 54(6):2037–2043, Dec 2007.

[67] F.L. Kastensmidt, C.K. Filho, and L. Carro. Improving reliability of sram-based fpgas by inserting redundant routing. *Nuclear Science, IEEE Transactions on*, 53(4):2060–2068, Aug 2006.

[68] F.L. Kastensmidt, L. Sterpone, L. Carro, and M.S. Reorda. On the optimal design of triple modular redundancy logic for sram-based fpgas. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1290–1295 Vol. 2, March 2005.

[69] K. Kyriakoulakos and D. Pnevmatikatos. A novel sram-based fpga architecture for efficient tmr fault tolerance support. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 193–198, Aug 2009.

[70] J.H. Lala and R.E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, Jan 1994.

[71] L.A. Tambara, F.L. Kastensmidt, J.R. Azambuja, E. Chielle, F. Almeida, G. Nazar, P. Rech, C. Frost, and M.S. Lubaszewski. Evaluating the effectiveness of a diversity tmr scheme under neutrons. In *Radiation and Its Effects on Components and Systems (RADECS), 2013 14th European Conference on*, pages 1–5, Sept 2013.

[72] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin. Improving fpga design robustness with partial tmr. In *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, pages 226–232, March 2006.

[73] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K.A. LaBel, M. Friendlich, H. Kim, and A. Phan. Effectiveness of internal versus external seu scrubbing mitigation strategies in a xilinx fpga: Design, test, and analysis. *IEEE Transactions on Nuclear Science*, 55(4):2259–2266, Aug 2008.

[74] Salazar A Carmichael C, Caffrey M. Correcting single event upsets through virtex partial reconfiguration. Technical Report xapp216, 2000.

[75] D.P. Schultz, L.C. Hung, and F.E. Goetting. Fpga configuration circuit including bus-based crc register, February 20 2001. US Patent 6,191,614.

[76] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin. Dynamic reconfiguration for management of radiation-induced faults in fpgas. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 145–, April 2004.

[77] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb. Fpga partial reconfiguration

via configuration scrubbing. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 99–104, Aug 2009.

[78] M.G. Gericota, L.F. Lemos, G.R. Alves, and J.M. Ferreira. On-line self-healing of circuits implemented on reconfigurable fpgas. In *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International*, pages 217–222, July 2007.

[79] Conrado Pilotto, José Rodrigo Azambuja, and Fernanda Lima Kastensmidt. Synchronizing triple modular redundant designs in dynamic partial reconfiguration applications. In *Proceedings of the 21st Annual Symposium on Integrated Circuits and System Design*, SBCCI '08, pages 199–204, New York, NY, USA, 2008. ACM.

[80] Diessel O Cetin E. Guaranteed fault recovery time for fpga based tmr circuits employing partial reconfiguration. In *Proceesings of 2nd International Workshop on Computing in Heterogeneous Autonomous N Goal-Oriented Environments*, 2012.

[81] Kwang-Ting Cheng, Shi-Yu Huang, and Wei-Jin Dai. Fault emulation: a new approach to fault grading. In *Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on*, pages 681–686, Nov 1995.

[82] Kwang-Ting Cheng, Shi-Yu Huang, and Wei-Jin Dai. Fault emulation: A new methodology for fault grading. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(10):1487–1495, Oct 1999.

[83] Jin-Hua Hong, Shih-Arn Hwang, and Cheng-Wen Wu. An fpga-based hardware emulator for fast fault emulation. In *Circuits and Systems, 1996., IEEE 39th Midwest symposium on*, volume 1, pages 345–348 vol.1, Aug 1996.

[84] Shih-Arn Hwang, Jin-Hua Hong, and Cheng-Wen Wu. Sequential circuit fault simulation using logic emulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(8):724–736, Aug 1998.

[85] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante. An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits. *J. Electron. Test.*, 18(3):261–271, June 2002.

[86] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena-Arrontes. An autonomous fpga-based emulation system for fast fault tolerant evaluation. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 397–402, Aug 2005.

[87] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena. Autonomous fault emulation: A new fpga-based acceleration system for hardness evaluation. *Nuclear Science, IEEE Transactions on*, 54(1):252–261, Feb 2007.

[88] L. Burgun, F. Reblewski, G. Fenelon, J. Barbier, and O. Lepape. Serial fault emulation. In *Design Automation Conference Proceedings 1996, 33rd*, pages 801–806, Jun 1996.

[89] Xilinx Inc. *JBits SDK 2.8 for Virtex*.

[90] L. Antoni, R. Leveugle, and B. Feher. Using run-time reconfiguration for fault injection applications. In *Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE*, volume 3, pages 1773–1777 vol.3, 2001.

[91] L. Antoni, R. Leveugle, and B. Feher. Using run-time reconfiguration for fault injection in hardware prototypes. In *Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings. 17th IEEE International Symposium on*, pages 245–253, 2002.

[92] L. Antoni, R. Leveugle, and B. Feher. Using run-time reconfiguration for fault injection applications. *Instrumentation and Measurement, IEEE Transactions on*, 52(5):1468–1473, Oct 2003.

[93] A. Parreira, J.P. Teixeira, A. Pantelimon, M.B. Santos, and J.T. de Sousa. Fault simulation using partially reconfigurable hardware. In Peter Y. K. Cheung and GeorgeA. Constantinides, editors, *Field Programmable Logic and Application*, volume 2778 of *Lecture Notes in Computer Science*, pages 839–848. Springer Berlin Heidelberg, 2003.

[94] Abílio Parreira, J. P. Teixeira, and Marcelino Santos. A novel approach to fpga-based hardware fault modeling and simulation. In *Proc. of the Design and Diagnostics of Electronic Circuits and Syst. Workshop*, pages 17–24, 2003.

[95] D. de Andres, J.C. Ruiz, D. Gil, and P. Gil. Fast emulation of permanent faults in vlsi systems. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–6, Aug 2006.

[96] M. Grosso, H. Guzman-Miranda, and M.A. Aguirre. Exploiting fault model correlations to accelerate seu sensitivity assessment. *Industrial Informatics, IEEE Transactions on*, 9(1):142–148, Feb 2013.

[97] M. Niknahad, O. Sander, and J. Becker. Fgtmr - fine grain redundancy method for reconfigurable architectures under high failure rates. In *Nano, Information Technology and Reliability (NASNIT), 2011 15th North-East Asia Symposium on*, pages 186–191, Oct 2011.

[98] Gabriel L. Nazar and Luigi Carro. Exploiting modified placement and hardwired resources to provide high reliability in fpgas. In *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, FCCM '12, pages 149–152, Washington, DC, USA, 2012. IEEE Computer Society.

[99] Jones L champman K. Seu strategies for virtex-5 devices. Technical Report xapp864, 2009.

[100] L. Sterpone and A. Ullah. On the optimal reconfiguration times for tmr circuits on sram based fpgas. In *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 9–14, June 2013.

[101] L. Sterpone and M. Violante. A new analytical approach to estimate the effects of seus in tmr architectures implemented through sram-based fpgas.

IEEE Transactions on Nuclear Science, 52(6):2217–2223, Dec 2005.

[102] M.S. Reorda, L. Sterpone, and A. Ullah. An error-detection and self-repairing method for dynamically and partially reconfigurable systems. In *Test Symposium (ETS), 2013 18th IEEE European*, pages 1–7, May 2013.

[103] Xilinx Inc. *Virtex 5 FPGA User Guide*. Xilinx.

[104] Xilinx Inc. *Constraints guide*.

[105] Opencores repositories.

[106] Jason Cong, Chang Wu, and Yuzheng Ding. Cut ranking and pruning: Enabling a general and efficient fpga mapping solution. In *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, FPGA '99, pages 29–35, New York, NY, USA, 1999. ACM.

[107] R. Francis, J. Rose, and Z. Vranesic. Chortle-crf: fast technology mapping for lookup table-based fpgas. In *Design Automation Conference, 1991. 28th ACM/IEEE*, pages 227–233, June 1991.

[108] A.H. Farrahi and M. Sarrafzadeh. Complexity of the lookup-table minimization problem for fpga technology mapping. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(11):1319–1332, Nov 1994.

[109] J. Cong and Y. Ding. On area/depth trade-off in lut-based fpga technology mapping. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(2):137–148, June 1994.

[110] L. Sterpone and M. Violante. A new partial reconfiguration-based fault-injection system to evaluate seu effects in sram-based fpgas. *Nuclear Science, IEEE Transactions on*, 54(4):965–970, Aug 2007.