Czech Technical University in Prague

Faculty of Electrical Engineering

Doctoral Thesis

March 2007

Pavel Kubalík

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science and Engineering

Design of Self Checking Circuits Based on FPGAs

Doctoral Thesis

Pavel Kubalík

Prague, March 2007

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Supervisor: ***Hana Kubátová***

Thesis Supervisor:

Hana Kubátová

Department of Computer Science and Engineering

Faculty of Electrical Engineering

Czech Technical University in Prague

Abstract

This thesis presents the structure and design principles of a new highly reliable fault tolerant system. The proposed system structure is based on cooperation between two identical FPGA systems each including self-checking circuits. The main idea is based on a standard duplex system combined with fault-tolerant design principles and reconfiguration (static or dynamic).

The system combines two individual FPGA circuits with identical design, where each FPGA is a totally self-checking circuit that enables fault detection. Because the hardware redundancy techniques obviously lower the dependability parameters, this system is designed as reconfigurable. The correct design is repeatedly loaded into a faulty FPGA, when the fault is detected. A totally self-checking circuit is based on error detection codes for which one hundred percent faulty coverage and low area overhead are difficult to achieve. The type of code is discussed, and it is experimentally verified that in many cases single parity is enough for our purpose. Single parity keeps a low area overhead with relatively high fault coverage. The whole system contains two totally self-checking comparators for cases, when the fault is not detected. One totally self-checking comparator is used for each FPGA. The comparators compare the primary outputs. When the outputs are different and no fail signal from the totally self-checking circuit is detected, both FPGAs are reconfigured.

The design methodology for this system is also presented in this thesis. The new fault classification was proposed to obtain better view on various fault types. A Markov model is presented to evaluate the dependability parameters of the proposed system. The proposed modified duplex system is more reliable than the original duplex system, and has a lower area overhead than a triple module redundancy system.

Keywords:

on-line testing, self-checking circuits, fail-safe circuits, error detecting codes, FPGA, highly reliable design, fault tolerant design, reconfiguration.

Acknowledgements

First of all, I would like to express my gratitude to my thesis supervisor, Hana Kubátová. She has been a constant source of encouragement and insight during my research. Her continued support is gratefully acknowledged. Her efforts as thesis supervisor contributed substantially to the quality and completeness of the thesis. Together with Prof. Ondřej Novák, she has provided me with numerous opportunities for professional advancement. I have learned a great deal from both of them.

Many other people have influenced my work. In particular I wish to thank to Petr Fišer, Jan Schmidt and Radek Dobiáš.

The staff of the Department of Computer Science and Engineering has provided a pleasant and flexible environment for my research. Especially, I would like to thank Prof. Pavel Tvrdík, head of the department, for taking care of my financial support. My work has been partially supported by grants from the GACR grant agency.

Finally, my greatest thanks go to my family and friends.

Contents

List of Figures

List of Tables

1. Introduction

VLSI testing has been dominated by the need to achieve high quality manufacture testing with acceptable cost. With the rapidly increasing complexity of VLSI circuits, this goal has become increasingly difficult and has biased the effort of the test community in the direction of manufacturing testing [1].

However, important industrial applications require protection against field failures, and require an on-line testing solution. At first these needs concerned specific products destined for safety critical applications and fault tolerant computing, which were produced in small quantities. At the same time, there were not enough applications to make it attractive for CAD vendors to develop tools specific to the design of on-line testable ICs. The lack of such tools has dramatically increased the effort needed design on-line testable ICs. Low-volume production of such applications often does not justify the high development cost, which will have a dramatic impact on the per product unit cost. In practice, techniques using off-the-shelf components, such as duplication or triplication, are more often adopted, since the development cost is much lower, although the production cost is relatively high.

We can expect this situation to start changing. Various industrial sectors have rapidly increasing needs for on-line testing. Such sectors include railway control, satellites, avionics, telecommunications, control of critical automotive functions, medical electronics, industrial control, etc. We can also expect wider sectors of the electronics industry to demand on-line testing solutions in order to ensure the welfare of the users of electronic products. Some of these applications involve high volume production and should support the standardization of such techniques, in the same way that the increasing needs of VLSI testing have transformed DFT and BIST into standard design techniques, and have supported the development of the specific tools now offered by most CAD vendors.

Since silicon is "cheap", such tools should greatly popularize the design of on-line testable circuits. In addition to these trends, the high complexity of present-day systems requires more efficient solutions. The complex multi-chip systems of yesterday have become present-day single-chip components. Indeed, the fault tolerant and fail-safe system designs of yesterday have to be integrated at chip level, calling for on-line testing techniques for VLSI.

A large variety of on-line testing techniques for VLSI have been developed and are still being enriched by new developments. They can respond efficiently to the needs expressed above, provided that available CAD tools simplify their implementation. Such techniques include self-checking design and signature monitoring.

1.1. Motivation

FPGAs are typically based on SRAMs. This means that the functional bitstream is not permanently saved in FPGA and must be loaded after FPGA is powered on. Because the FPGA configuration is saved in volatile memory, FPGAs are more sensitive to faults. High fault sensitivity is particularly important in aviation and astronautics, due to the long distances from the ground and the low level of the earth’s shield.

Nowadays transistor sizes are decreasing, power voltage is decreasing, and the threshold is decreasing. These properties decrease the noise immunity and increase the SEU sensitivity of FPGA. This is a good reason to focus on SEU detection, localization and correction.

Higher energy particles impacting on an FPGA die can cause various faults. A typical example is a function circuit change due to the impact of high energy particles on the transistor drain. These types of faults are called Single Event Upsets (SEU) [2, 3, 4]. This change occurs after high energy particles impact digital or analog parts. These faults are nondestructive, and after resetting or the correct bitstream load, the functionality of FPGA is restored. The fault can appear as a short pulse or as a change in the flip-flop. The volume of the final errors depends on the structure implemented in FPGA, and on the place where the fault appears. A configuration memory bit change is another example of an SEU effect. These faults are temporary and can be corrected after the new bitstream is loaded into FPGA.

1.2. Related work

Previous approaches benefited from the fact that it was possible to work at functional/logical level, by providing the necessary fault observability properties to each node constituting the functional description of the device under consideration. With ASIC, even when mapping with different technological libraries, commercial tools are able to maintain the functional description of each node constituting the network. Thus a TSC device is produced even when the gates that are used are not exactly those identified by the Boolean equations. With FPGA, the nodes constituting the network are collapsed and merged to better suit the basic CLB elements constituting the FPGA resources, in order to minimize the used area. This operation modifies the observability of each fault, thus potentially not fulfilling the required and previously provided fault-error relation. Hence no assumptions can be made on the observability of each fault on the primary outputs, so that subsequent TSC fault analysis and re-design steps are necessary.

The goal of the investigation proposed here is to explore the suitability of Concurrent Error Detection (CED) techniques based on Error Detection Codes for the FPGA platform. Attention is therefore initially given to Single Event Upsets (SEUs) that may corrupt the internal memory or the LUTs.

The FGPA configuration is stored in SRAM, and any changes to this memory may lead to a malfunction of the implemented circuit. SEUs caused by high-energy particles impacting sensitive parts are one way that configuration memory can be change. Some results of SEU effects on the FPGA configuration memory are described in [2, 3, 29, 57, 64]. These changes are described as soft errors and cannot be detected by an offline test without interrupting the circuit function.

Concurrent Error Detection (CED) techniques are widely used to increase the system dependability parameters. Almost all CED techniques are based on the original circuit contains the primary inputs and outputs, as well as another unit which independently predicts some special characteristic of the primary system outputs for every input sequence. Finally, a checker unit checks whether the special characteristic of the output actually produced by the system in response to the input sequence is the same as the predicted response, and produces an error signal when a mismatch occurs. Some examples of the characteristics can be: single parity, a number of 1s or a number of 0s.

The architecture of a general CED scheme is shown in Figure 1. It is important to note that the general architecture of the CED scheme, as shown in Figure 1, makes use of some form of hardware redundancy (predictor and checker circuits) for error-detection. Time redundancy techniques such as alternate-data-retry and recomputation with shifted operands can also be used for concurrent error detection. Time redundancy directly affects the system performance, but the hardware cost is generally lower than the cost of hardware redundancy.

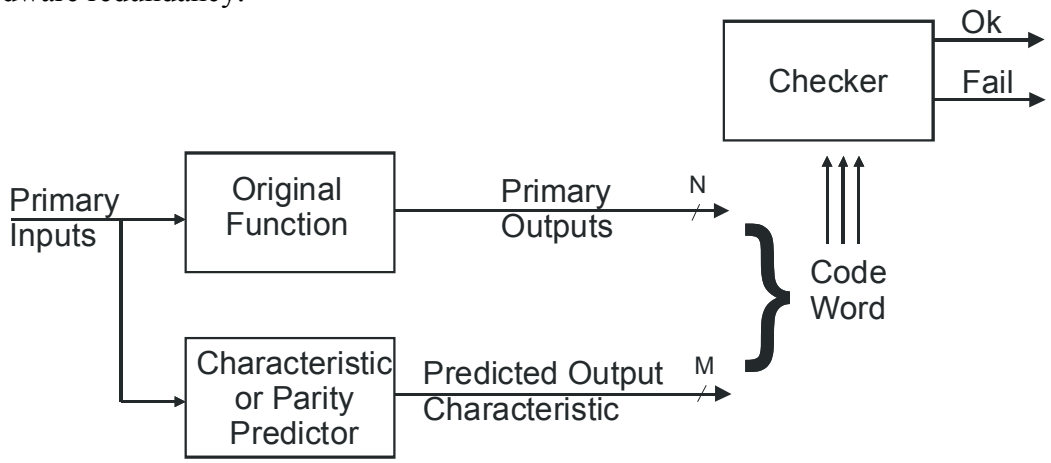


Figure 1. CED scheme

Several CED schemes for reliable computing system design have been proposed and used commercially. These techniques differ mainly in their error-detection capabilities and in the constraints required for the system design. There are many publications on system design with concurrent error detection [5, 6, 9, 20, 27, 28, 32, 33, 56]. They include designs of datapath circuits (e.g., adders, multipliers), and general combinational and sequential logic circuits with concurrent error detection. Almost all publications on CED focus on their area/performance overhead. However, the systems considered here are restricted to those with redundancy through replication. All the above-mentioned CED techniques guarantee system data integrity against single faults. However, these CED schemes are vulnerable to multiple faults and common-mode failures. Common-mode failures are a special and very important cause of multiple faults.

Common-mode failures (CMFs) produce multiple faults, which generally occur due to a single cause; system data integrity is not guaranteed in the presence of CMFs. They include design mistakes and operational failures that may be due to external causes (e.g., EMI, power-supply disturbances and radiation) or internal causes. CMFs in redundant VLSI systems are surveyed in [5, 7].

Another approach focusing on CED techniques using hardware redundancy is presented in [5, 6, 9, 20, 27, 28, 32, 33, 56]. Concurrent error detection (CED) techniques (based on hardware duplication, parity codes, etc.)[16] are widely used to enhance system dependability parameters. All CED techniques introduce some form of redundancy. Redundant systems are subject to common-mode failures (CMFs). While most studies of CED techniques focus on the area overhead, few analyze the CMF vulnerability of these techniques.

The next approach was presented in [8]. This paper addresses the issue of self-checking FPGA design, based on the adoption of error detection codes (e.g., Berger code, Parity code) as an evolution of the traditional approaches developed in past years for the ASIC platform. Research was done on the applicability of design techniques introducing

hardware fault detection properties in a combinational network through information redundancy at functional/gate level. This approach is the starting point for the design of a more complete methodology of dynamically reconfigurable FPGAs in response to a fault, once it has been detected. Furthermore, the original fault-error analysis tool was adapted at the circuit description level. Therefore fault-error relation enforcement can be directly suited for FPGA, due to better control of the effects of manipulations of commercial tools and the presence of unused logic.

The characteristics predictor is not the only unit that is important for realizing the CED scheme. The Checker also plays an important role in the CED scheme. The Checker depends on the characteristics predictor. Many papers have been published on this topic [21, 22].

In many publications the quality of a self-checking circuit is characterized by the number of detected faults. In many cases, however when the fault coverage is high (almost 100%) the area overhead is too high. The high area overhead decreases the fault tolerant properties, and it was important to find some trade-off between the area overhead and the used code. These requirements have been taken into account in this dissertation.

CED techniques based on ED codes are widely used. However many research groups have not evaluated the Totally Self-Checking parameter, fault Secure and Self Testing property of the final circuit. Many publications describe only the TSC parameter. But this parameter provides insufficient information about all faults in a circuit implemented in FPGA. The hidden faults are not taken into account. Therefore a new fault classification has been proposed to describe faults in FPGA caused by SEU.

A fault tolerant system can satisfy fault masking requirements. A fault occurring in such a system is detected and does not lead to an incorrect function. If no techniques for error correcting are used, the system must be stopped after the next fault is detected. A fault tolerant system protected against SEU must also be reliable. Additional techniques must be taken into account to increase the reliability parameters. However, CED techniques increase the final area, and techniques to increase the reliability parameters based on a single FPGA are not sufficient. Some publications have focused on reliable systems based on a single FPGA using a TMR structure [25, 58, 63].

The TMR structure is unsuitable when a high area overhead is unacceptable. Some hybrid architecture must be used. TMR architecture and a hybrid system, e.g., the modified duplex system with a comparator and some CED technique are compared in [59, 61]. A technique based on Duplication With Comparison (DWC) and the Concurrent Error Detection (CED) technique are described in [60, 62].

The fault tolerant system proposed in this work is based on DWC-CED with reconfiguration. This thesis is devoted to methods for maximally increasing the dependability parameters with maintaining the minimal area overhead. The complex structure implemented in each FPGA is divided into small blocks, where every block satisfies TSC properties. This approach can detect a fault before the fault is detected by the output comparator.

Design methodology plays an important role in fault tolerant systems based on a self-checking circuit. The methodology of self-checking code selection was presented in [51, 65]. Here the methodology assumes that the circuits are described by multilevel logic and are realized by ASICs. The synthesis process of this self-checking circuit is different from the classical method. Each part of the self-checking circuit is synthesized

individually, due to possible sharing of logic primitives among these blocks. The sharing logic decreases the number of detected faults. Some papers describing methodologies for VHDL automatic modification has been published [53, 54].

A design flow for protecting an FPGA-based system against SEUs is presented in [25]. This paper presents a design flow composed of standard tools and also ad-hoc developed tools, which designers can use fruitfully for developing a circuit resilient to SEUs. Experiments are reported on benchmark circuits and on a realistic circuit to show the capabilities of the proposed design flow.

There is another on-line testing approach that does not take the implemented design into account. The on-line test is processed for a whole FPGA, without disturbing the normal system operation. [11, 55].

1.3. Contribution of this dissertation thesis

The main result of the dissertation thesis is a fault tolerant design methodology based on self-checking circuits implemented with using FPGAs. The methodology describes the design steps of the fault tolerant system realization. The main contribution of this work can be divided into two groups: primary results and secondary results.

The primary results are: design methodology steps, fault classification, self-checking and fault tolerant structures and etc.

- **Fault classification**

This work supports the design process of CED circuits implemented in FPGAs. We have proposed a new fault classification. Briefly, our classification leads to more accurate evaluation of the fault coverage, and we can determine whether the tested circuit satisfies the FS and ST properties. We can also evaluate how many of the faults violate the FS and ST property. The proposed fault classification is used in our experiments. The classification enables us to distinguish which ED code is suitable for the chosen synthesis method for the fault model that is used.

- **Self-checking circuit suitable for a fault tolerant system**

Previous works show that the area overhead depends on the ED codes that are used. To obtain the minimal area overhead and 100% of fault coverage, we have proposed method to select the appropriate ED code, which may increase the dependability parameters.

- **Single parity and parity net grouping**

We propose a very efficient application of on-line BIST design. Here the circuit outputs are joined together by XOR gates, to form a parity predictor. The parity predictor outputs are compared with the outputs of the original circuit, and thus the appropriate circuit function is checked. The proposed method helps to minimize the parity predictor logic overhead.

- **Modified duplex system (MDS)**

CED techniques are not able to increase the dependability parameters sufficiently. A new structure based on the DWC-CED technique has been developed. An appropriate ED code was selected to ensure a trade-off between area overhead and fault coverage. The dependability parameters depend on these two criteria.

- **MDS Implementation with TSC**

Our methodology for fault tolerant design is based on SC circuits. It assumes a combinational circuit with up to 16 primary inputs, because simulation time grows by

the square of the number of inputs. Therefore a compound design architecture has been proposed. The proposed architecture enables combinational circuits and sequential circuits to be combined in compound design.

- **HW emulation of MDS**

Each reaction to an input vector change must be calculated in the SW simulation. Each simulation step takes many processor cycles, especially for circuits with many gates. One simulation step is processed, and the time needed for calculation is equal to one system cycle. However the results need to be compared and evaluated concurrently. We therefore decided to use an HW emulator. The HW emulatorr was programmed with respect to the Atmel FPSLIC FPGA design process.

- **Proof of MDS optimality**

We describe the MDS system using Markov dependability model. This model is used for computing the availability parameters for the SEU fault model. The results of MCNC [36] and ISCAS [30] benchmarks used in our MDS, reconfigurable and on-line testing design method are compared with the result for the standard duplex system and TMR.

- **Design metodology**

A fault tolerant system design methodology is presented with the aim of obtaining results from individual parts of this study. The design methodology enables to use the system in a mission-critical application, where the dependability parameter requirements are very high.

The secondary results are: implementation, modification and other tools.

- **SW simulator**

Because a new fault classification is being presented here, a new fault simulator is needed. This SW simulator has been written in Java programming language.

- **HW emulator**

The new HW emulator was designed to evaluate faults more precisely. The HW emulator was programmed with respect to the Atmel FPSLIC FPGA design process.

- **Tools that add single parity nets**

Some special tools for modifying benchmark circuits need to be used in this work. We programmed some utilities allowing circuit modification where circuit is described by two-level networks and also multilevel networks. We designed some utilities enabling us to simulate and calculate fault coverage.

- **Tools that add multiple parity nets**

We have programmed a tool enabling modification of the combinational circuit and selection of the appropriate ED code. This tool can generate a single event parity

predictor, a multiple parity predictor and a Hamming-like code predictor. The BOOM [35] and Espresso [34] minimization tools are used to evaluate the area overhead and thus select the appropriate ED code.

1.4. Organization of the thesis

Chapter 1 - The motivation and related works are presented in this section. Works related to each part of the design methodology are described with references to publications by other research groups that have deal with related problems.

Chapter 2 - This chapter describes the entire theoretical background and also defines all important terms in this area of research.

Chapter 3 – In this chapter we introduce a new fault classification for use in comparing different techniques for designing TSC circuits.

Chapter 4 - Concurrent checking methods verify circuits during normal operation. Because the outputs delivered by a circuit during its operations as a part of a system are unpredictable, we need to introduce an invariant property that we can check for this invariance. Self-checking (SC) design is used to achieve concurrent error detection (CED) using hardware redundancy. This chapter describes the self-checking circuit design methodology and the results obtained by SW simulation.

Chapter 5 – This chapter focuses on the proposed MDS structure, and deals with issues in the design of a fault tolerant system.

Chapter 6 – To evaluate the influence of a sequence of SEU faults, we need a more precise definition of a "single fault" is needed. We use availability computation for dependability analysis. This chapter describes the results of the dependability computations obtained from the proposed Markov model of our MDS fault tolerant structure. We compare the original duplex system and TMR with our proposed MDS system.

Chapter 7 – A new design methodology is proposed in this chapter. All steps in this methodology are based on the results given in Chapter 3, 4, 5 and 7.

2. Theoretical Background

When speaking about fault tolerant design, there are some important terms that need to be defined. Defect, fault, and error are key terms in the field of testing [1, 17, 18, 19 ,52].

Defects can occur anywhere on the die, on one of multiple layers, packages, boards, etc. Generally, a defect (failure mechanism) in an electronic system is an unintended physical difference between the implemented hardware and its intended design. A defect may cause s deviation from the given device specifications. Defects occur either during manufacture or during the use of devices and systems [52].

Faults are defined as representations of a **defect** at the abstracted function level (electrical, Boolean or functional malfunction). The difference between a defect and a fault is rather subtle. They are imperfections in the hardware and in the function, respectively [52].

Errors are wrong output signals produced by a defective system (an incorrect response in circuit behavior). Therefore an error is any behavioral effect caused by a physical defect [52]. Errors can be divided into two groups: soft errors and hard errors.

Soft errors are errors occurring **temporarily** in a device, and can be corrected. The correction process depends on the fault that generated the soft error. The fault is typically caused by a change in a device under the test parameters (e.g. temperature, power consumption, cosmic rays, etc.)

Hard errors are permanent errors caused by hard faults. When a hard error occurs, the device must be replaced or masked by some fault masking techniques.

A fault-tolerant system is a system that provides some techniques to avoid failure after a fault has caused errors within the system. When the system satisfies fault tolerant requirements, the following four steps must be taken into account:

Fault detection is the process of recognizing that a fault has occurred. This is often required before any recovery procedure can be implemented.

Fault location is the process of determining where a fault has occurred, so that an appropriate recovery can be implemented.

Fault containment is the process of isolating a fault and preventing the effects of that fault from propagation throughout a system.

Fault recovery is the process of repairing the faulty part of a system to put the system back into an operational state.

Functional blocks can be tested by two basic methods: on-line tests and off-line tests.

In an **off-line test**, the tested system is switched to the non-operational state, where the system is tested by an additional logic performing test. The system is inaccessible while it is being tested. This type of test is in many cases performed after the device has been manufactured.

In an **on-line test**, the test is performed when the system is in the operational state. This test is performed continually during normal life of device. When a fault is detected, the system can stop the device and correct the fault. When a fault tolerant system is used, the faults are corrected without changing the functionality of the device. The recovery operation can be processed if the error is caused by a soft fault, or some redundant techniques can be used.

Fault tolerant systems lead designers use some redundant techniques. The most important redundant techniques are described in the following section.

2.1. Redundancy Techniques

The concept of redundancy implies the addition of information, resources, or time beyond what is needed for normal system operation. Redundancy can take one of several forms, including hardware redundancy, software redundancy, information redundancy, and time redundancy. The use of redundancy can provide additional capabilities within a system. In fact, whenever fault tolerance or fault detection is required, some form of redundancy is also required. However, it should be understood that redundancy can have a very important impact on a system in the areas of performance, size, weight, power consumption, reliability, and others [19].

2.1.1. Hardware Redundancy

Physical replication of hardware is perhaps the commonest form of redundancy used in systems. As semiconductor components have become smaller and less expensive, the concept of hardware redundancy has become commoner and more practical. The cost of replicating hardware within a system is decreasing, simply because the cost of hardware is decreasing.

There are three basic forms of hardware redundancy. First, passive techniques use the concept of fault masking to hide the occurrence of faults and prevent the faults from resulting in errors. Passive approaches are designed to achieve fault tolerance without requiring any action on the part of the system or operator. In their most basic form, passive techniques do not provide fault detection. They simply mask the faults.

The second form of hardware redundancy is the active approach, which is sometimes called the dynamic method. Active methods achieve fault tolerance by detecting the existence of faults and performing some action to remove faulty hardware from the

system. In other words, active techniques require that the system perform a reconfiguration to tolerate the faults. Active hardware redundancy uses fault detection, fault location, and fault recovery in an attempt to achieve fault tolerance.

The final form of hardware redundancy is the hybrid approach. Hybrid techniques combine the attractive features of both the passive and active approaches. Fault masking is used in hybrid systems to prevent erroneous results from being generated. Fault detection, fault location, and fault recovery are also used in hybrid approaches to improve fault tolerance by removing faulty hardware and replacing it with spare hardware. Providing spares is one way of providing redundancy in a system. Hybrid methods are most often used in the critical-computation applications, where fault masking is required to prevent momentary errors, and high reliability must be achieved. Hybrid hardware redundancy is usually a very expensive form of redundancy to implement [19].

2.1.2. Information Redundancy

Information redundancy is the addition of redundant information to data to allow fault detection, fault masking, or possibly fault tolerance. Good examples of information redundancy are error-detecting and error-correcting codes, formed by the addition of redundant information to data words, or by mapping data words into new representations containing redundant information [19].

2.1.3. Time Redundancy

The fundamental problem with the forms of redundancy is the penalty paid in extra hardware to implement the various techniques. Both hardware redundancy and information redundancy can require large amounts of extra hardware for their implementation. In an effort to decrease the hardware required to achieve fault detection or fault tolerance, time redundancy has recently received much attention. Time redundancy methods attempt to reduce the amount of extra hardware at the expense of using additional time. In many applications, time is of much less importance than hardware, because hardware is a physical entity that impacts weight, size, power consumption and cost. Time, on the other hand, may be readily available in some applications. It is important to understand that the selection of a particular type of redundancy is very dependent upon the application [19].

2.1.4. Software Redundancy

In applications that use computers, many fault-detection and fault-tolerance techniques can be implement in software. The redundant hardware necessary to implement the capabilities can be minimal, while the redundant software can be substantial. Redundant software can occur in many forms; it is not necessary to replicate complete programs to have redundant software. Software redundancy can appear as several extra lines of code used to check the magnitude of a signal, or as a small routine used to periodically test a memory by writing and reading special locations.

2.2. Fault models classification

It is known that not all faults are identical. The test techniques that are used depend on the assumed fault model. Nowadays, the following fault models are used in the testing process [1, 17, 18, 19, 52].

- **Stuck-at faults** (SAF) i.e., a type of logical faults affecting logical signal states. This is the commonest fault model for digital circuits at the logic level.
- **Bridging faults,** or simply bridges or shorts, occur between two signals. Two types of bridging faults are recognized: inter-gate and intra-gate shorts. Shorts are the dominant cause of faults in present-day CMOS technologies.
- **Open faults** mean that a physical conducting line in a circuit is broken. The resulting unconnected node is not laid to either power source "Vdd" or ground "GND". The occurrence of such defects can provide "memory effect" or delay faults.
- **Delay faults** mean that timing specifications are not fulfilled. Gate or path delay faults have to be investigated.
- **Parametric faults** are defects causing changes in a device under the test parameters (e.g. current, Vdd and GND voltage, power consumption, temperature, etc.)
- **A Single Event Upset (SEU)** is defined by NASA as "radiation-induced errors in microelectronic circuits caused when charged particles (usually from the radiation belts or from cosmic rays) lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs." [Ref: NASA Thesaurus] SEUs are transient soft errors, and are non-destructive.
- **A Single Event Latchup (SEL)** is a condition that causes loss of device functionality due to a single-event induced current state. SELs are hard errors, and are potentially destructive (i.e., may cause permanent damage).
- **A Single Event Burnout (SEB)** is a condition that can destroy a device due to a high current state in a power transistor. SEB causes the device to fail permanently.

2.3. Design of Functional Blocks

The following properties are required for a functional block

- **Fault Secure:** Under each modeled fault the erroneous outputs that are produced do not belong to the output code. The reason for this property is obvious; if an erroneous output belongs to the code, the error is not detected and the TSC goal is not achieved. Thus, the fault secure property is the most important property required for the functional block.
- Self-Testing: For each modeled fault there is an input vector occurring during normal operation that produces an output vector which does not belong to the code. In fact, this property avoids the existence of redundant faults. Such fault remain undetectable and could be combined with new faults occurring later in the circuits, resulting on multiple fault that could destroy the fault secure and the self-testing properties

- Totally Self-Checking: The circuits are both fault secure and self-testing. Totally self-checking properties offer the highest level of protection [1].

The fault secure property guarantees that the first fault always generates detectable errors. Then, assuming that a sufficient time elapses between the occurrence of two faults so that the functional block receives all inputs required to test its faults (i.e., sufficiently long MTBF), the self-testing property guarantees that the first fault is detected before a second fault occurs in the S-C system. In this way the TSC goal is achieved for a TSC functional block.

TSC can be generalized as the strongly fault secure property that defines the largest class of functional circuits achieving the TSC goal.

The fault secure property is the most important property, since it guarantees error detection in the event of any single fault. However, it is also the most difficult property to achieve. The self-testing property can easily be achieved, especially for stuck-at faults, where it is enough to remove the redundant faults by simplifying the circuit. The most straightforward way to achieve the fault secure property is to duplicate the functional block and use a comparator to check the delivered outputs for equality.

Since this solution raises the hardware cost, by at least 100%, sophisticated techniques are being developed to reduce this cost. These techniques use error detection codes that cost less than the duplication code. The codes are described in greater detail in [1, 17].

2.4. Fail-Safe Design

The final stage of an electronic system often drives some actuators that control elements of the external world. Many systems have states that can be considered as safe. That is, they do not involve catastrophic events if they occur erroneously. A typical example of a safe state is a red traffic light. Nobody can go across the crossroad. In safety critical applications, each actuator must be controlled by a fail-safe signal, (i.e., a signal which in the event of failures is either correct or safe. Self-checking systems deliver groups of encoded signals and are not adequate for driving these actuators (since each actuator is controlled by a single line which must be fail-safe individually). Due to this particular requirement, it is not possible to implement fail-safe systems in VLSI. Therefore, existing fail-safe systems are composed of a self-checking or fault tolerant processing system (e.g., using error detecting codes, duplication, triplication, etc.), together with a fail-safe interface implemented using specific discrete components with very low probability of failing in the non-safe direction.

This interface transforms the outputs of the processing system into fail-safe signals. The drawback of these interfaces is that they are very cumbersome and have a high cost. Furthermore, the use of discrete components results in lower MTTF in VLSI implementations, and system availability is reduced. It is therefore necessary to provide more compact fail-safe interfaces. However, few results have been published in this domain.

2.5. Error Detection codes

Concurrent Fault Detection Circuits (CFDCs) are essential components for on-line testing in systems designed to be highly reliable, highly available, and diagnosable to a replaceable unit such as a PCB or chip. CFDCs are also referred to as concurrent error detection (CED) circuits. CFDCs are typically incorporated in VLSI and PCB designs to support system-level fault recovery and maintenance strategies. These circuits are typically applied in an ad hoc manner to ASIC design, and usually only to the data path circuits. Extensive use has been made of CFDCs in many digital systems, e.g., electronic switching systems.

Many types of error detecting codes (EDCs) and error correcting codes (ECCs) are used in the design of CFDCs. Different types of CFDCs require varying degrees of information redundancy (extra bits) in the system circuitry for the EDC/ECC code word. However, not all of these codes are useful in practical system applications, because of the large area and performance penalties associated with their hardware implementation. Therefore, the choice of a CFDC has a considerable impact on the overall area and cost of the final system. Unlike fault tolerant hardware structures that use hardware redundancy such as N-tuple Modular Redundancy (NMR), CFDCs are based on information redundancy using EDCs or ECCs. While there are many types of EDCs and ECCs, not all of these are useful in practical system applications because of the area and performance penalties that result from the circuitry required to generate the code words. Code word generation is performed on the data at the data source before entering the CUT, and the code words are checked (which requires regeneration of the code word) at the output of the CUT. Partitioning a system into sub-circuits and inserting the code word check and regeneration circuits to detect faults at intermediate points facilitates effective fault isolation and diagnosis of replaceable units.

2.5.1. Parity code

Parity code detects all single errors and more generally all errors of odd multiplicity. It is the cheapest code since it adds only one check bit to the information part. This check bit is computed to make constant the parity of each code word. As a matter of fact, we can use an odd parity code (an odd number of 1’s in each code word) or an even parity code (an even number of 1’s in each code word) [1].

2.5.2. Dual-Rail code

This is a variety of duplication code where the check bits are equal to the complements of the information bits. This code has very strong error detection capabilities, since it detects any errors affecting either the information part or its complement. However, it is quite expensive since it duplicates the information [1].

2.5.3. M-out-of-n code

This is a non-separable code (the information and check bits are merged). It is composed of code words that have exactly m 1’s, (e.g., 2-out-of-4 code: 1100, 1010, 1001, etc.) This code is an optimal non-separable unordered code (minimal redundancy for unordered coding) [1].

2.5.4. Berger code

Berger code [17] can detect all multiple unidirectional errors (where the bits in error fail as logic 1s or logic 0s, but not both in the same data word). However, it provides no error correction capability.

The basic idea is to count the number of logic 1s in the data word and use the inverted binary count value as the code word. By inverting the count value for use as the code word, we are able to detect a stuck-at-0 fault on a serial data line since the Berger code bits would be all 1s to indicate an all 0s data word. The number of data bits to be serviced by the Berger code word can be variable but should be less than (2N), where N is the number of Berger code bits, to ensure optimal error detection.

2.5.5. Arithmetic codes

These codes are divided into separable and non-separable. In separable arithmetic codes of base A, the code words are obtained by associating to an information part X a check part X' equal to the modulo A of the information part, that is $X0 = |X|A$ (residue code), or $X0 = A - |X|A$ (inverse residue code).

In non-separable arithmetic codes of base A (or AN codes), the code words are equal to the product of the original (i.e., non-coded) word by base A.

Arithmetic codes [1] are interesting for checking arithmetic operations, because they are preserved under such operations. The most useful arithmetic codes are separable codes, and they are most often implemented as low cost arithmetic codes, where check base A is equal to 2^m-1. In this case an m-bit modulo A adder is realized by an m-bit adder having the carry-out signal feeding back the carry-in signal (carry end-around adder).

Then, the check part generator for low cost arithmetic codes is realized as a modular network using these adders as building blocks. Low cost arithmetic codes detect variable arithmetic errors according to the value of the check base.

2.5.6. Hamming codes

Hamming codes [17] provide not only error detection but also an error correction capability based on an extension of the principles of parity. The Hamming code word is constructed from the parity bits of various combinations of data bits determined by the parity check matrix. Note that the decimal value of each bit position in the parity check matrix corresponds to the binary value of the parity check matrix. Also note that the Hamming code bits, Hi, occupy 2n positions in the parity check matrix and, as a result, have only a single 1 in any position in the column below Hi. It is easy to extend this matrix to accommodate any desired size data word with a new Hamming code bit introduced each time a new 2n value position is encountered. Each Hamming code bit is generated by the exclusive-OR of all the data bits, Di, that have a 1 in the same row at the corresponding Hamming bit.

Hamming circuits are more complex than parity circuits in terms of the number of gates and the number of additional bits required for the code word. However, Hamming code is quite efficient in comparison with other error correcting codes in terms of the area overhead and the performance penalty required for the error correction process. Hamming circuit models are based on single-bit error-correcting parity codes which use M Hamming bits to detect and correct single-bit errors in N data bits. Given N data bits,

the required value of M can be calculated by the relationship 2M > M + N. If a single bit error is detected, the error can be corrected in the output data bus and the presence is indicated by the active error signal.

2.6. Design of Checkers

The task of a checker is to signal the occurrence of a code input (by generating on its output a correct operation indication), and the occurrence of a noncode input (by generating an error indication. The set of output words indicating the correct form of the output code space of the checker and the set of output words indicating error occurrence form the output noncode space. As an implication of this task the checker verifies the code disjoint property [1].

2.6.1. Code-Disjoint

The checker maps code inputs into code outputs and noncode inputs into noncode outputs.

Code-disjointness is not related to the testability of the checker. It simply reflects a functional property. However, a fault occurring in the checker may alter its ability to produce an error indication output under a noncode input. If this fault is not detected, another fault can later occur in the functional block. Then, an erroneous noncode output produced by this block will eventually not be signaled by the checker due to its own fault. To cope with this problem, the checker must verify the self-testing property [1].

2.6.2. Self-Testing

For each modeled fault there is a code input that produces a noncode output.

In the case of functional blocks, provided that there is a long Mean Time Between Failures (MTBF), the self-testing property guarantees that the fault is detected before the occurrence of another fault in the system. In this way the TSC goal is achieved.

Self-testing code-disjoint checkers can be generalized into strongly code-disjoint checkers, which define the largest class of checkers allowing achieve the TSC goal.

Self-testing checkers are difficult to design, because it is necessary to detect all the faults in the checker by applying only code inputs. Fortunately, we need to consider only a limited number of checker classes corresponding to the more useful error detecting codes.

For these checkers, extensive investigations by numerous researches have accomplished this task. Thus, self-testing checkers are now available for all the error detecting codes used in self-checking design.

An important implication of the self-testing property is that a checker must have at least two outputs. In a single-output checker, one output value (e.g., logic 0) must be used to indicate correct operation and the second (e.g., logic 1) for error indication. Then, a stuck-at fault on the value corresponding to the correct operation indication cannot be detected and the checker is non self-testing. Such a fault is very dangerous, since it will mask any subsequent fault occurring in the functional block. Because of this danger, the use of two output checkers is generally. In this case, the dual-rail indication is used for error indication. The error function is signalized by 00 and 11 values. [1].

2.7. Perturbation Tolerant Memories

Complex electronic systems are subject to transient faults provoked by various causes such as electromagnetic interference, cross-talk, alpha particles, cosmic rays, etc. Transients are the main cause of failures in complex electronic systems. In some particular applications, e.g., space application, protection against soft errors (SEUs caused by heavy ion strikes) is mandatory. There are strong requirements for protection against transients in fault tolerant systems and in safety critical applications. The introduction of deep submicron technologies also significantly increases the sensitivity of VLSI circuits to various causes of transients. As a matter of fact, hardware techniques for designing perturbation tolerant circuits may have a considerable impact on the design of a large number of electronics systems [1].

Memory elements are the most sensitive parts of a CMOS circuit, since static CMOS logic is drastically less sensitive than memory cells to various causes of transient faults. Thus, perturbation resistant/tolerant memory design is the key point for designing perturbation tolerant ICs. Perturbation tolerant design for large memory arrays (e.g., large RAMs, caches, etc.) can be efficiently achieved by means of error correcting codes. However, this solution cannot be used in the case of memory elements distributed across the logic of an IC. This is also a very expensive solution for implementing small embedded memories, because the cost of an error correcting code (check bits plus the error correction controller) will be very high. In these situations, the use of perturbation hardened memory cells is the most appropriate option.

2.8. Availability

Fault tolerance is an attribute that is designed into a system to achieve a design goal. Just as a design must meet many functional and performance goals, it must also satisfy numerous other requirements. The most prominent of the additional requirements are dependability, reliability, availability, safety, performability, maintainability, and testability; fault tolerance is a system attribute capable of fulfilling such requirement.

Dependability is a term used to encapsulate the concepts of reliability, availability, safety, maintainability, performability, and testability.

Reliability is a conditional probability, in that it depends on the system being operational at the beginning of the chosen time interval. The reliability of a system is a function of time $R(t)$.

Availability is a function of time $A(t)$, defined as the probability that a system is operating correctly and is available to perform its functions at instant of time t.

Safety is the probability $S(t)$ that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system.

Performability of a system is a function of time *P(L,t)*, defined as the probability that the system performance will be at or above some level *L*, at instance of time *t*. In many cases, it is possible to design systems that can continue to perform correctly after the occurrence of hardware and software faults, but the level of performance is somehow diminished.

Maintainability is a measure if the ease with which a system can be repaired once it has failed. In more quantitative terms, maintainability is the probability *M(t)*, that a failed system will be restored to an operational state within a period of time t. The restoration process includes locating the problem, physically repairing the system, and bringing the system back to its operational condition.

Testability is simply the ability to test for certain attributes within a system. Measures of testability allow us to assess the ease with which certain tests can be performed.
These parameters are described in greater detail in [26].

Applications of fault tolerant computing can be categorized into four primary areas: long-life applications, critical computations, maintenance postponement, and high availability. Each application presents differing design requirements and challenges [26].

- **Long-life applications**: space flight and satellites.
- **Critical-computation applications**: aircraft flight control systems, military systems and certain types of industrial controllers. (Environmental cleanliness or equipment protection)
- **Maintenance postponement applications**: remote processing stations and certain space applications.
- **High availability applications**: banking, railway control and other time-shared systems

2.9. Reliability Modeling

Reliability is one of the most important attributes of systems. Almost all specifications for systems mandate that certain values for reliability be achieved and, in some way, demonstrated. The most popular reliability analysis techniques are analytical approaches. Of the analytic techniques, combinatorial modeling and Markov modeling are the two most commonly used approaches. Here, we consider both combinatorial and Markov modeling.

Combinatorial models

Combinatorial models use probabilistic techniques that enumerate the different ways in which a system can remain operational. The probabilities of events that lead to a system being operational are calculated to form an estimate of the system's reliability. The reliability of a system is generally derived in terms of the reliabilities of the individual

components of the system. The two models of systems that are most common in practice are series models and parallel models. In a series system, each element of the system is required to operate correctly for the system to operate correctly. In a parallel system, on the other hand, only one of several elements must be operational for the system to perform its functions correctly.

Markov models

The primary difficulty with combinational models is that many complex systems cannot be modeled easily in a combinatorial fashion. Reliability expressions are often very complex. In addition, the fault coverage that we have seen to be extremely important in the reliability of a system is sometimes difficult to incorporate into the reliability expression in a combination model. Finally, the process of repair that occurs in many systems is very difficult to model in a combinatorial fashion. For these reasons, we often use Markov models, which are sometimes referred to as Markov chains.

These types of models are described in greater detail in [26].

3. New Fault Classification Scheme

In this chapter we will introduce a new fault classification that can be used for the comparing different techniques for designing TSC circuits. New fault classification was published in [A.5].

The use of ED codes and possibly some special synthesis methods does not necessarily ensure the TSC property. We need to evaluate how many faults violate the FS and ST property to make a comparison of different methods. In common fault classifications, the faults are divided into two groups according to the testability of the faults. This classification is not sufficient for our purpose. It is necessary to distinguish whether the change to an output caused by a fault is detectable by the given ED code.

3.1. The New Proposed Approach

Fault detection can be based on two different approaches – comparison of two values (duplication), and the use of ED codes. In the first case, the outputs of two units are compared. Assuming that one fault can occur at a time; at least one unit will produce correct values. This means that when a fault-free comparator is assumed, each error caused by any fault in a unit will be detectable. Evaluating the error detection capabilities in the second case is more complicated. The correct output is not known during the processing. The fault detection ability depends only on the ED codes that are used. It is not sure that each fault causes a detectable error. It is necessary to use a different approach to fault classification. For each input vector, the responses of a circuit in the presence of a fault can be divided into three groups:

1. **No error** – the fault does not affect the output values. The data is not corrupted, but the presence of a fault is not detected.

2. **Detectable error** – the fault changes the outputs into a non-code word. This is the best case, because the presence of a fault is detected.

3. **Undetectable error** – the output vector is a valid codeword, but is incorrect (incorrect codeword). This is the worst case, because the checker is not able to detect this error.

Each circuit has a set of allowed input vectors. The faults can be divided into four classes, according to the reaction of the circuit to their presence. These classes are:

- **Class A** - hidden faults. These are faults that do not affect the circuit output for any allowed input vector. Faults belonging to this class have no impact on the FS property. However, if this fault can occur, a circuit cannot be ST.

- **Class B** - faults that are detectable by at least one input vector and do not produce an incorrect codeword (a valid codeword, but incorrect) for other input vectors. These faults have no negative impact on the FS and ST property.

- **Class C** - faults that cause an incorrect codeword for at least one input vector and are not detectable by any other input vector. Faults from this class cause undetectable errors. If any fault in the circuit belongs to this class, the circuit is neither FS nor ST.

- **Class D** - faults that cause an undetectable error for at least one vector and a detectable error for at least one other vector. Although these faults are detectable, they do not satisfy the FS property and so they are also undesirable.

In the following text we use the above mentioned class identifiers. The fault classification is shown in Table 1. Each row corresponds to one class. The columns contain the numbers x_i, y_i, z_i of input vectors that cause the corresponding response at the output, $x_i, y_i, z_i > 0, x_i + y_i + z_i \le N, i = B, C, D$, N is the number of allowed input vectors.

Table 1. Classification of faults for SC circuits

Class	No errors	Detectable errors	Undetectable errors
A	N	0	0
B	x_B	y_B	0
C	x_C	0	z_C
D	x_D	y_D	z_D

With regard to the definitions of the FS and ST properties, we define the following theorems:

- A circuit will be FS and ST only if all the faults belong to class B.
- A circuit will be FS only if all the faults belong to class A or B.
- A circuit will be ST only if all the faults belong to class B or D.

These theorems follow directly from the definitions of FS and ST.

The FS property is achieved only when a special method of synthesis is used. However, if a much simpler method is used, the number of faults that violate the FS property may not be high. It may be useful to evaluate this value to compare different methods. We can use this value to evaluate how much the circuit satisfies the FS property.

The evaluation of the FS property (the number of faults that belong to class A or B) is independent of the set of allowed input words. If a fault does not manifest itself as an incorrect codeword for all possible input words, it cannot cause an undetectable error for any subset of input words. So we can use the exhaustive test set for combinational circuits, and a test that uses all transitions for a sequential circuit.

The evaluation of the ST property (the number of faults that belong to class B or D) is more complicated due to the fact that some input words may not appear. For combinational circuits, where the set of input words is not defined, an exhaustive test set is generated. However, in a real situation, some input words may not occur. This means that some faults can be undetectable. This can decrease the final fault coverage. A similar situation can occur for sequential circuits. For the evaluation of ST we use all possible transition edges of the state transition graph. However, some transition edges can be unreachable from any state (e.g., some edges that are used after reset only). For this reason, there is a higher number of faults that can be undetectable. It is not possible to determine whether the circuit is ST without a good knowledge of the allowed input words.

3.2. Experimental Verification

In our experiments, we will use the proposed fault classification to evaluate the FS and ST property of some methods. The main aim is to show the advantages of the proposed fault classification. Our experiments focus on combinational and sequential MCNC benchmarks [36]. The benchmarks were implemented in Xilinx FPGA. The stuck-at-1 and stuck-at-0 fault model was considered. Tables 12, 13, 14, 15 contain the results of the experiments. These tables contain the name of the tested benchmarks "Circuit", the number of input pins "Inputs", the number of output pins, including check bits "Outputs", the number of state bits, including check bits "State bits", the total number of considered faults (All faults), the number of faults that cause a change at the outputs (X) for one input vector at least and the number of faults according to our classification (A, B, C, D) in Table 1. The method of fault class calculation method is performed by scheme shown in Figure 2.

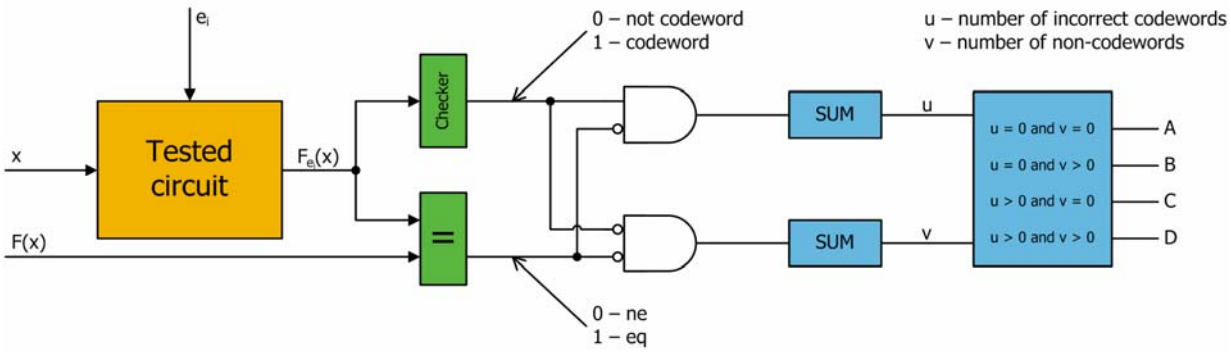


Figure 2. Fault classification scheme

3.2.1. Combinational Circuits

The benchmarks presented here are based on real circuits used in large designs. The exhaustive test set generated for the combinational circuit is limited by the number of inputs of the circuit. The main drawback is that for a big number of inputs the exhaustive test set is too large. For circuits with more than 16 inputs the simulation time increases rapidly (doubles with every added input). Due to this restriction we use only circuits with fewer than 16 inputs for our experiments.

The design technique for these experiments is based on the structure described in Figure x. As the first step we must fully define the outputs. Then we use a copy of the original circuit to create the predictor. The outputs of the copied circuit are replaced by

parity nets calculated from the original outputs. The original circuit and the parity predictor are synthesized individually, because the synthesis process can cause sharing of equivalent logic between the two circuits, and this can lead to lower fault coverage. The lower fault coverage is caused by the logic used for both the original circuit and the predictor. This means that for this shared part the parity bits are not calculated.

Two experiments were carried out for combinational circuits, one with Hamming-like codes (Table 2) and the second with even parity (Table 3).

Table 2. Experiment 1 – combinational circuits and Hamming like code

Circuit	Inputs	Outputs	All faults	X	A	B	C	D
alu1	12	12	1076	1076	0	1076	0	0
apla	10	17	1434	1409	25	1409	0	0
b11	8	37	736	734	2	734	0	0
br1	12	12	1014	994	20	972	0	22
al2	16	54	1180	1166	14	1166	0	0
alu2	10	12	1784	1784	0	1784	0	0
alu3	10	12	1344	1344	0	1344	0	0

Table 3. Experiment 2 – combinational circuits and even parity

Circuit	Inputs	Outputs	All faults	X	A	B	C	D
alu1	12	9	2594	2566	28	2566	0	0
apla	10	13	632	632	0	522	3	107
b11	8	32	418	416	2	321	42	53
br1	12	9	594	594	0	369	78	147
al2	16	48	628	627	1	576	17	34
alu2	10	9	830	819	11	757	0	62
alu3	10	9	622	622	0	572	0	50

More than 99% of the considered faults (X) cause a change of outputs for at least one input vector in both experiments. These results correspond to the common fault classification. In other words, we can say that these faults are testable when we use an external tester. The advantage of our classification is evident from the results of experiment 2. Although more than 99% of all the faults change the output for at least one input vector (X), only 85% of all faults satisfy the FS property (A + B). About 96% of all faults satisfy the ST property (B + D). When our classification is used, we can say that the method used in Experiment 1 is better than the method used in Experiment 2 from the FS and ST point of view.

3.2.2. Sequential Circuits

In the experiments with sequential circuits, the state variables and outputs are coded by ED codes. The faults at the primary inputs are not considered. In Experiment 3 we use an even parity code and in Experiment 4 the M-out-of-N code (1 out of N for state variables and reduced M-out-of-N code for outputs). The KISS2 description was

modified according to the ED code that was used. KISS2 was translated to the VHDL code using our tool. The standard synthesis process was used. We used our fault simulator, as mentioned above. The results are shown in Table 4 and Table 5.

Table 4. Results of Experiment 3 – sequential circuits and even parity

Circuit	State bits	Outputs	All faults	X	A	B	C	D
mc	3	6	174	168	6	147	21	0
s386	7	8	746	727	19	529	115	83
mark1	5	17	684	625	59	503	86	36
beecount	4	5	292	274	18	253	17	4
pma	6	9	1236	1131	105	826	99	206
ex6	4	9	670	645	25	407	143	95

Table 5. Results of Experiment 4 – sequential circuits and M-out-of-N code

Circuit	Sate bits	Outputs	All faults	X	A	B	C	D
mc	4	7	322	293	29	272	21	0
s386	13	10	1018	988	30	943	45	0
mark1	15	18	678	649	29	616	33	0
beecount	7	4	306	296	10	275	17	4
pma	24	15	1214	1185	29	1146	17	22
ex6	8	14	790	760	30	725	24	11

Approximately 96% of all faults (X) cause a change at the outputs for at least one input vector. Approximately 80% of all faults (A+B) in the case of even parity and 95% of all faults in the case of M-out-of-N code satisfy the FS property. Approximately 82% of all faults in the case of even parity and 91% of all faults in the case of M-out-of-N code satisfy the ST property (B + D). We can say that the use of M-out-of-n code produces better results.

This work supports the design process of CED circuits implemented in FPGAs. We propose a new fault classification. We can summarize that our classification leads to a more accurate evaluation of the fault coverage, and we can determine whether the tested circuit satisfies the FS and ST properties. We can also evaluate how many of the considered faults violate the FS and ST property. The proposed fault classification was used in four experiments. The classification allows us to distinguish which ED code is suitable for the chosen synthesis method for the fault model that is used.

3.3. SW Simulator

If we want to compare different techniques for TSC circuit design, the distribution of the faults considered here into the above defined classes has to be obtained. A suitable fault simulator is needed. Most simulators (e.g. FSIM [12] or HOPE [50]) compare the

correct outputs with outputs when there is a fault. They cannot classify the faults as precisely as we need. These simulators are therefore not suitable. We used the simulator described in [31, A.13]. This simulator has the following features:

- The simulation is performed for circuits described by a netlist format (EDIF).
- Stuck-at-1 and stuck-at-0 faults on the inputs and outputs of components are considered.

- Combinational and sequential circuits are supported.

- This simulator supports circuits where inputs, outputs and internal states (in the case of a sequential circuit) are coded by even parity, multiple parity and 1 out of N code. Multiple code groups can be used to ensure TSC. The simulator also supports Hamming-like codes and M out of N code.

- Only Xilinx Virtex netlist format is supported.

4. Concurrent Error Detection

Systems realized by FPGAs have become more and more popular due to several properties and advantages:

- High flexibility in achieving multiple requirements such as cost, performance, turnaround time.
- Possible reconfiguration and later changes of the implemented circuit, e.g., only via radio net connections.
- Mission critical applications such as aviation, medicine, space missions, and also railway applications [24].

The design process for FPGAs differs from the design process for ASICs mainly in the "design time", i.e., in the time needed from the idea to its realization. Moreover, FPGAs enable different design properties, e.g., in-system reconfiguration to correct functional bugs or update the firmware to implement new standards. For this reason, and due to the growing complexity of FPGAs, these circuits can also be used in mission-critical applications such as aviation, medicine or space missions.

The process when high-energy particles impact sensitive parts is described as a Single Event Upsets (SEUs) [2]. SEUs can lead to bit-flips in SRAM. The FGPA configuration is stored in SRAM, and any changes to this memory may lead to a malfunction of the implemented circuit. Some results of SEU effects on FPGA configuration memory are described in [2, 3, 29, 57, 64].

CED techniques can enable faster detection of a soft error (an error which can be corrected by a reconfiguration process) caused by an SEU. SEUs can also change values in the embedded memory used in the design, and can cause data corruption. These changes are not detectable by off-line tests, only by some CED techniques. The FPGA fabrication process allows the use of sub-micron technology with smaller and smaller transistor size. Due to this fact the changes in FPGA memory contents, affected by SEUs, can therefore be observable even at sea level. This is another reason why CED techniques are important.

There are three basic terms in the field of CED:

- The **Fault Security property (FS)** means that for each modeled fault, the produced erroneous output vector does not belong to the proper output code word.

- The **Self-Testing property (ST)** means that for each modeled fault there is an input vector occurring during normal operation that produces an output vector which does not belong to the proper output code word.

- The **Totally Self-Checking property (TSC)** means that the circuit must satisfy FS and ST properties.

There have been many papers [5, 6, 9, 20, 27, 28, 32, 33, 56] on concurrent error detection (CED) techniques. CED techniques can be divided into three basic groups according to the type of redundancy. The first group focuses on area redundancy, the second group on time redundancy and the third group on information redundancy. When we speak about area redundancy, we assume duplication or triplication of the original circuit. Time redundancy is based on repetition of some computation. Information redundancy is based on error detecting (ED) codes, and leads either to area redundancy or to time redundancy. Next, we will assume the utilization of information redundancy (area redundancy) due to the use of ED codes.

Concurrent checking verifies circuits during their normal operation. Because the outputs delivered by a circuit during its operations as a part of a system are unpredictable, we need to introduce some invariant property in order to be able to check for this invariance. Self-checking (S-C) design is used to achieve concurrent error detection using means of information (hardware) redundancy. A complex circuit is partitioned into its element functional blocks and each of these blocks is implemented according to the structure of Figure 1.

The basic method for the proper choice of a CED model is described in [5]. Techniques using ED codes have also been studied by other research groups [32, 33].

4.1. Method Using the Old Fault Classification

4.1.1. Single Parity and Hamming-Like Codes

There are many ways to generate checking bits. A single even parity code is the simplest code that can be used to get a code word at the output of the combinational circuit. This parity generator performs XOR over all primary outputs, and the modification is processed on a circuit described by a multi-level network. However, the single even parity code is mostly not appropriate to ensure the TSC goal. Nevertheless, our results show that parity code is most suitable for fault tolerant design, due to its low area overhead.

Another error code is a Hamming-like code, which is in essence based on the single parity code (multi parity code). The Hamming code is defined by its generating matrix.

4.1.1.1. The Fault Model

All our experiments are based on FPGA circuits. The circuit implemented in an FPGA consists of individual memory elements (LUTs - look up tables). We can see 3 gates mapped into an LUT in Figure 3.

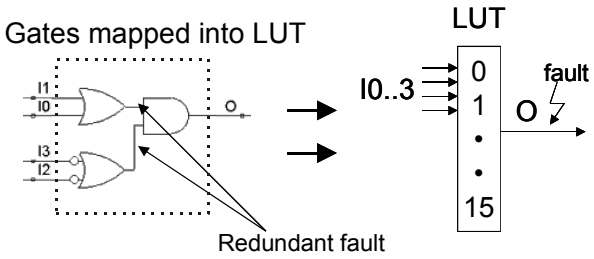


Figure 3. Fault model

The original circuit has two inner nets. The original set of test vectors covers all faults in these inner nets. These test vectors are redundant for an LUT. For circuits realized by LUTs a change (a defect) in the memory leads to a single event upset (SEU) at the primary output of the LUT. Therefore we can use the SEU fault model in our experiments to detect SEU – only some of the detected faults will be redundant.

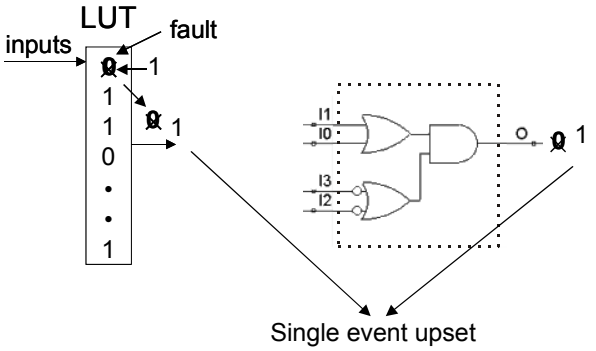


Figure 4. Fault Model – Example

Our SEU fault model is described by a simple example in Figure 4. For simplicity, only one LUT is used. This LUT implements a circuit containing 3 gates. The primary inputs from I0 to I1 are the same as the address inputs for LUT. When this address is selected its content is propagated to the output.

We assume the following situation: first the content of this LUT can be changed, e.g., due to electromagnetic interference, cross-talk or alpha particles. The appropriate memory cell is set to '1' and the wrong value is propagated to the output. This means that the realized function is changed and the output behaves as a single event upset. We can say that a change of any LUT cell leads to a stuck-at fault on the output, according to this example. This fault is observed only if the faulty cell is selected. This is the same situation as for circuits implemented by gates. Some faults can be masked and do not necessarily lead to an erroneous output.

Some faults are masked, and they may appear when previously unused logic is being used. E.g., One bit of an LUT is changed. If appropriate bit in the LUT is selected by the address decoder, the erroneous output will appear.

4.1.1.2. Parity Bits Predictor Using Hamming-Like Codes

We used a matrix containing the unity sub-matrix on the left side for simplicity. The generating matrix of the Hamming code (15, 11) is shown in Figure 5. The values a_{ij} have to be defined.

When a more complex Hamming code is used, more values have to be defined. The number of outputs o_i used for the checking bits determines the appropriate code. For example, the benchmark alu1 [35] having 8 outputs requires at least the Hamming code (15, 11). Therefore 8 data bits and 4 checking bits are used. The definition of values a_{ik} is also important.

Now we present a method for generating values a_{ik}. Let us mention the Hamming code (15, 11) with 4 checking bits. In our case (alu1) we have only 8 bits. Therefore the reduced Hamming matrix must be used.

$$G = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & 1 & \cdots & 0 & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & a_{11,1} & a_{11,2} & a_{11,3} & a_{11,4} \end{array}\right)$$

Figure 5. Generating matrix for Hamming code (15, 11)

The sub-matrix has only 8 rows and 4 columns after reduction. We can define eight 4-bit vectors or four 8-bit vectors. The second case will be used here. The search for erroneous output is a similar method to a binary search. The first vector is composed of log. 1s only. The last vector is composed of log. 1s in the odd places and log. 0s in the even places. Each vector except the first contains the same number of 1s and the same number of 0s. An example of the possible content of the right part of the sub-matrix is shown in Figure 6.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 6. Right part of the generating matrix

The number of vectors in the set is the same as the number of rows in the appropriate Hamming matrix. The way to generate parity output for checking bit x_k is described by equation 1:

$$x_k = a_{1k} o_1 \oplus a_{2k} o_2 \oplus \ldots \oplus a_{mk} o_m, \quad (1)$$

where $o_1 \ldots o_m$ are the primary outputs of the original circuit.

4.1.1.3. Experiments

The experiments in this section are based on a circuit modification described by the multi-level network. The parity bits are incorporated into the tested circuit as a tree composed of XOR gates. The maximum area of the parity generator can be calculated as the sum of the original circuit and the size of the XOR tree. These experiments are based on previous fault classification where only a minimal test set is used.

Atalanta software was used to generate the minimal test [12]. This tool allows process ISCAC benchmarks based on equations. A tool for circuit optimization described by tables cannot be used in this case because table creation is difficult due to the large number of inputs. Special software was written in C language for automatic modification of the original circuit and adding the parity bits generator. The ISCAS benchmarks are loaded into inner form by this software. The form is composed of list of nets and gates. Some functions are written to modify this inner form and can add new

gates and nets, or can change the names of all nets. Net renaming enables duplication of an original circuit. After the original circuit is loaded into memory all nets are renamed and the original circuit is loaded again. To add parity bits the original circuit is loaded and modified by the corresponding function. Then all nets are renamed and the original circuit is loaded again. By this procedure the original circuit with predicted parity bits has been obtained. For each tested checking code, new functions modifying a circuit have been written. The even parity, double parity, multiple parity generated by the Hamming code are used in the experiments described here. When the original circuit is modified, two methods of output form can be generated. The first form is the ISCAS benchmark used for simulation. The second form is the VHDL source code, which allows a modified circuit to be synthesized. Variety circuits can be created by this tool. Each part of the final circuit is created separately and combined together with the same tool. If the area occupied by the circuit in FPGA is to be computed, then the VHDL output is selected and a synthesize tool is used. Then the original circuit and the modified circuit can be compared. In our case the VHDL code is used to obtain the area overhead by each checking code. Two VHDL codes are generated: the first for the original circuit and the second for the circuit generating parity bits. For simulation, the design contains both the original circuit and the parity bits generator (Figure 7).

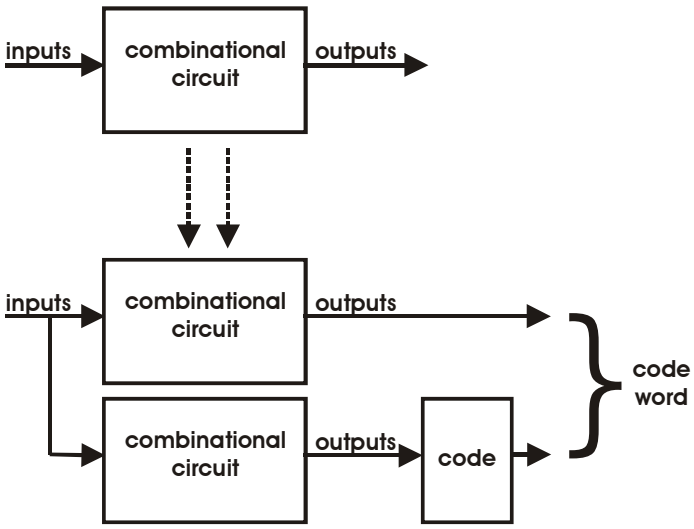


Figure 7. Circuit duplication and code word generation

Atalanta software processes the modified circuit and generates the minimal fault test. Both files, minimal test and modified circuit, are put into the simulator, which enables the efficiency of the checking code to be computed. The same tool that enables the original circuit to be modified is used for simulation. The same form of circuit stored in memory is used. The new simulation functions are added to the source code of our tool, see Figure 8. With this modified tool we can simulate faults. Firstly, our tool simulates stuck-at-zero faults by consecutive fault insertion in every net. When a fault is inserted the whole set of patterns from the minimal test is applied to every net. The same steps are used for simulating stuck-at-one. The simulator does not simulate faults on primary inputs and outputs. For each step of the test, the outputs of circuits with and without inserted faults are processed. In cases when the outputs are incorrect a check of the code word is performed.

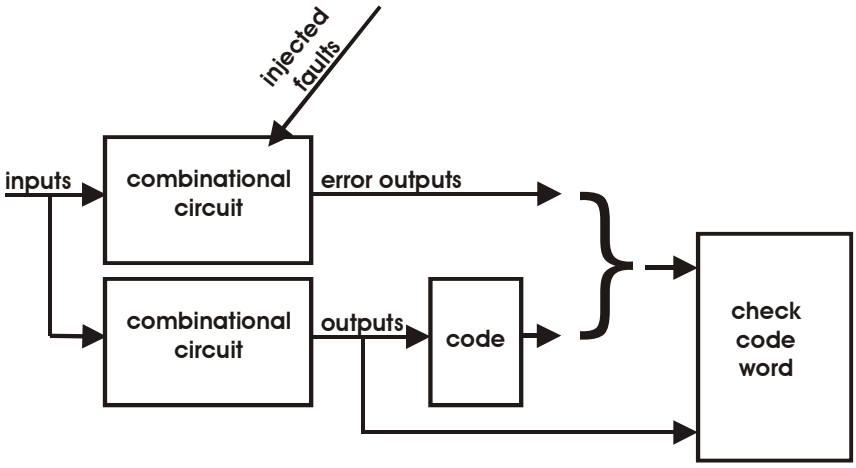


Figure 8. Automatic fault insertion and checking code word

We have chosen codes, e.g. Hamming codes, even parity and double parity, for which the checking combinational part can be generated easily. Berger code is not used because it is difficult to generate the checking combinational part.

Even parity, the simplest checking code, was used for the first experiment. The results, presented in Table 6, show that one parity bit cannot cover all faults inserted into the tested circuit.

Table 6. Application of even parity code

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	1	5536	4626	83,56	72	69
c499	41	32	1	15150	14628	96,55	87	88
c880	60	26	1	24567	23998	97,68	117	112
c1355	41	32	1	64165	62472	97,36	92	87
c1908	33	25	1	134012	119280	89,01	125	120
c2670	233	140	1	105532	84840	80,39	166	175

Circuit c17 is not used for our experiments because of its simplicity.

In the second experiment, double parity was used to generate checking bits. Even and odd bits of the outputs are coded separately by even parity. The results of this experiment are presented in Table 7.

Table 7. Application of double even parity code

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	2	5433	5012	92,25	73	69
c499	41	32	2	13984	13750	98,33	99	88
c880	60	26	2	27495	27206	98,95	120	112
c1355	41	32	2	62834	62012	98,69	90	87
c1908	33	25	2	130140	124958	96,02	124	120
c2670	233	140	2	116220	111270	95,74	219	175

Both of these experiments failed to achieve 100% fault coverage of the tested circuits.

The next way to generate checking bits is by using the Hamming code, which enables more checking bits to be added while retaining the quality of the Hamming code.

The Hamming code is defined by its generating matrix. For simplicity we use the matrix containing the unity submatrix on the left side. The generating matrix of Hamming code (15, 11) is shown in Figure 6. The values a_{ij} have to be defined.

When a more complex Hamming code is used, more values have to be defined. The number of outputs o_i used for the checking bits determines the appropriate code. For example, circuit c432, which has 7 outputs, requires at least Hamming code (15, 11). In this case we use 7 data bits and 4 checking bits. The definition of values a_{ik} is also important.

Now we present a method for generating values a_{ik}. Let us mention Hamming code (15, 11), which has 4 checking bits. We generate a set of all 4bit vectors. From all these vectors we remove vectors containing less than 2 binary '1'. The resulting subset is relatively regular - there are many zeros on the upper left side and many ones on the lower left side of the subset (see the left matrix in Figure 4). This regularity must be removed. If not, some parity bits will lose the capability to detect a fault. To eliminate this phenomenon, every even row from the beginning of the set is mutually exchanged with a corresponding even row from the end (see Figure 10).

$$\begin{pmatrix} 0&0&1&1\\ 0&1&0&1\\ 0&1&1&0\\ 0&1&1&1\\ 1&0&0&1\\ 1&0&1&0\\ 1&0&1&1\\ 1&1&0&0\\ 1&1&0&1\\ 1&1&1&0\\ 1&1&1&1 \end{pmatrix} \Rightarrow \begin{pmatrix} 0&0&1&1\\ 1&1&1&0\\ 0&1&1&0\\ 1&1&0&0\\ 1&0&0&1\\ 1&0&1&0\\ 1&0&1&1\\ 0&1&1&1\\ 1&1&0&1\\ 0&1&0&1\\ 1&1&1&1 \end{pmatrix}$$

Figure 9. Generating left side of matrix

The number of vectors in the set is the same as the number of rows in the appropriate Hamming matrix. Then we generate circuits for checking bits x_k (Equation 1).

The third experiment is based on Hamming code (63, 57), where the maximum number of data bits is 57 and the number of checking bits is 6. The experimental results are shown in Table 8.

Table 8. Application of Hamming code (63, 57)

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	6	5569	5544	99,55	77	69
c499	41	32	6	17791	17791	100,00	116	88
c880	60	26	6	27109	27106	99,99	140	112
c1355	41	32	6	68647	68647	100,00	117	87
c1908	33	25	6	123651	123376	99,78	145	120

The fault coverage for c499 and c1355 benchmarks is 100%. This means that Hamming code (63, 57) is appropriate. We should mention here that the fault coverage depends on the generated minimal test. If the minimal test created by Atalanta does not cover all faults, we cannot say that the simulated circuits are 100% fault covered. In other words,

some faults cannot be detected because the minimal test set does not cover all faults. This Hamming code cannot be used for benchmark c2670 because the number of its outputs is greater than the Hamming code can cover.

The fourth experiment is based on Hamming code (255, 247). The maximum number of data bits is 247 and the number of checking bits is 8. In our case only 7 outputs are used. The experimental results are shown in Table 9.

Table 9. Application of Hamming code (255,247)

Circuit	Inputs	Outputs	Redundancy parity bits	All tested faults	Detected faults	Detected faults[%]	Area occupation[LUT]	
							Redundancy part	Data part
c432	36	7	7	5694	5602	98,38	74	69
c499	41	32	7	18003	18003	100,00	111	88
c880	60	26	7	30277	30277	100,00	134	112
c1355	41	32	7	69634	69634	100,00	104	87
c1908	33	25	7	135402	134600	99,41	138	120
c2670	233	140	7	160092	160061	99,98	314	175

In summary, all of our experiments indicate that 100% fault coverage can be achieved using more redundancy outputs generated by special codes. The Hamming code can be used as a suitable code to generate parity bits. The type that is used depends on the number of outputs and on the complexity of the original circuit. More complex circuits need a greater number of parity outputs. Due to the fact that only a minimal test set was used, the final fault coverage reflects only the ST property. The exhaustive test set must be used to obtain the FS property. The new fault classification was used to better describe whether the circuit satisfies ST and FS properties.

4.2. Methods Using the New Fault Classification

To determine whether the circuit satisfies the TSC property, detectable faults belonging to one of four classes A, B, C and D [A.5] have to be calculated, see section 3.

This fault classification can be used to calculate how much the circuit satisfies the FS or ST properties and then to calculate the TSC property. This new approach to fault classification leads to creation of a new fault simulator.

In our design methodology we evaluate FS and ST properties. For ST properties a hidden fault is not assumed.

The evaluation of the FS property is independent of the set of allowed input words. If a fault does not manifest itself as an incorrect codeword for all possible input words, it cannot cause an undetectable error for any subset of input words. So we can use the exhaustive test set for combinational circuits.

The exhaustive test set is generated to evaluate the ST property for combinational circuits, where the set of input words is not defined. However, in a real situation some input words may not occur. This means that some faults can be undetectable. This can decrease the final fault coverage. Therefore, the number of faults that can be undetectable is higher.
The fault simulation process is performed for circuits described by a netlist (for example .edif).

4.2.1. Single Parity and Hamming-Like Codes

A parity predictor is used to generate the appropriate output code of the circuit in our research, see Figure 1. These techniques ensure a small area overhead and a higher SEU fault coverage, but the SEU fault coverage that is achieved is not 100% [27, 5, 28].

The circuit area overhead depends significantly on the parity codes that are used. If we use a strong error detecting code, e.g., Hamming code or Berger code, the FS parameter is almost 100% but the area overhead is high [A.3], [8].

The following FPGA structures are vulnerable to SEUs: mux select lines, programmable interconnect point states, buffer enables, LUT values, and control bit values.

Any changes to mux select lines, programmable interconnect point states or buffers lead to a significant circuit function change, but the function change is hardly detected for SEUs impacted in LUTs [29]. The probability of SEUs impacting routing resources (mux select lines, programmable interconnect point states and buffers) is about 78%, and only about 15-21% for LUTs. Thus, if there are many SEUs there will be a significant circuit function change. But any change in LUTs is hardly detected because of their small impact on the realized function. In some cases these faults may be undetected.

4.2.1.1. Area Overhead Minimization

The benchmarks used in this paper are described by a two-level network. The final area overhead depends on the minimization process. We used two different methods in our approach. Both these methods are based on simple duplication of the original circuit.

Our first method is based on a modification of the circuit described by a two-level network. The area of the parity bits predictor contributes significantly to the total area of the TSC circuit. As an example, we consider a circuit with 3 inputs (c, b and a) and 2 outputs (f and e). The parity bits predictor uses the odd parity code to generate the parity bits. In our example we have only one check bit x.

Our example is shown in Table 10. Output x was calculated from outputs e and f. We have to generate the minimal form of the equation at this time. We can achieve the minimal form using methods such as the Karnaugh map or Quine-McCluskey in our example. We use other minimization methods to minimize benchmarks or real circuits. After minimization we obtain three equations, one per output (f, e and x), where x means an odd parity of outputs f and e. If we want to know whether the odd parity covers all faults in our simple combinational circuit example, we have to generate the exhaustive test set and simulate all faults in each net in this circuit.

Table 10. Example of parity generator

c b a	f e	x
0 0 0	0 1	0
0 0 1	1 0	0
0 1 0	1 0	0
0 1 1	1 0	0
1 0 0	0 1	0
1 0 1	0 1	0
1 1 0	1 1	1
1 1 1	0 0	1

The final equations are:

$$e = bc + a(b + c) \quad (2)$$
$$f = ab + c(a + b) \quad (3)$$
$$x = bc \quad (4)$$

Our second method is based on a circuit modification, where the circuit is described by the multi-level network. The parity bits are incorporated into the tested circuit as a tree composed of XOR gates. The maximal area of the parity generator can be calculated as the sum of the original circuit and the size of the XOR tree.

4.2.1.2. Experimental Evaluation Software

Figure 11 describes how the test is performed for each detecting code, where the circuit modification is processed by the first method described above. The MCNC benchmarks [36] were used in our experiments. These benchmarks are described by a truth table. To generate the output parity bits, all the output values have to be defined for each particular input vector. Only a limited number of values are specified for each multi-dimensional input vector, and the rest are assigned as don't cares; they are left to be specified by another term. Thus, in order to be able to compute the parity bits, we have to split the intersecting terms, so that all the terms in the truth table are disjoint.

In the next step, the original primary outputs are replaced by parity bits. Two different error codes were used to calculate the output parity bits (single even parity code and Hamming code). Another tool was used in the case where the original circuit was modified in multilevel logic. This tool is described in [A.1]. Two circuits generated in the first step (the original circuit and the parity circuit) are processed separately to avoid sharing any part of the circuit. Each part is minimized by the Espresso tool [34]. The final area overhead depends on the software that was used in this step. Many tools were used to achieve a small area of the parity bits generator. Only Espresso was used to minimize the final area of the circuit described by the two level network. In this step the area overhead is known for implementation to ASIC. For FPGAs the area overhead is known after the synthesize process has been performed.

The "pla" format is converted into the "bench" format in the next step. The "bench" format was used because the tool which generates the exhaustive test set uses this format. An exhaustive test set has 2^n patterns, and we used it to evaluate the TSC goals.

Another conversion tool is used to generate two VHDL codes and the top level. The top level is used for incorporating the original and parity circuit generator. In the next step, the synthesis process is performed by Synplify [37]. The constraint properties set during the synthesis process express the area overhead and the fault coverage. If the maximum frequency is set too high, the synthesize process causes hidden faults to occur during the fault simulation. The hidden faults are caused by circuit duplication or by the constant distribution. The size of the area overhead is obtained from the synthesize process. The final netlist is generated by the Leonardo Spectrum [38] software. The fault coverage was obtained by simulation using our software.

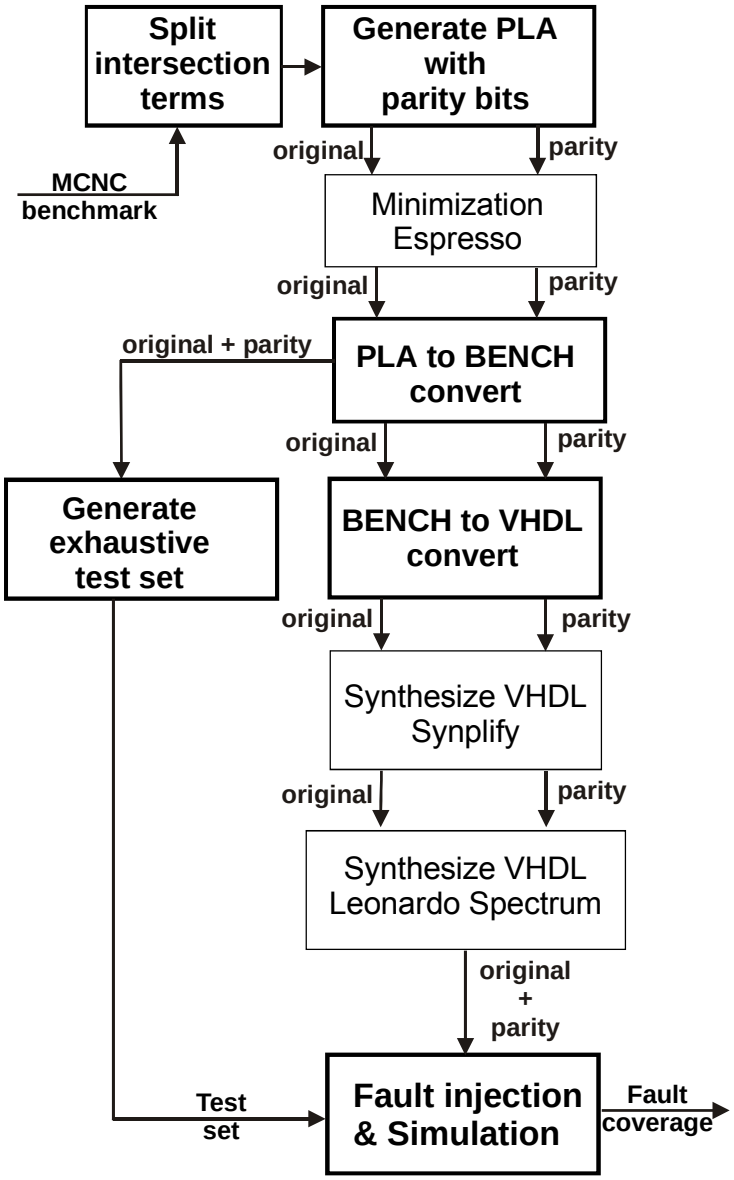


Figure 10. Design scheduling of a self-checking circuit

The format converting software and the parity generator software were written in Microsoft Visual C++.

4.2.1.3. Experiments and Results

The combinational MCNC benchmarks [36] were used for all the experiments. These benchmarks are based on real circuits used in large designs.

Since the whole circuit will be used for reconfiguration in FPGA, only small circuits were used. Real designs that have a large structure must be partitioned into several smaller parts. For large circuits, the process of area minimization and fault simulation takes a long time. This disadvantage prevents us examining more methods of designing the check bits generator.

The evaluated area, FS and ST properties depend on circuit properties such as the number of inputs and outputs, and the circuit complexity. The experimental results show that a more important property is the structure of the circuit. Two basic properties are described in Table 11.

Table 11. Description of the tested benchmarks

Circuit	Inputs	Outputs
alu1	12	8
apla	10	12
b11	8	31
br1	12	8
al2	16	47
alu2	10	8
alu3	10	8
c17	5	2

In the first set of experiments our goal was to obtain one hundred percent of the FS and ST property, while we measured the area overhead. In this case, the maximum of the parity bits was used.

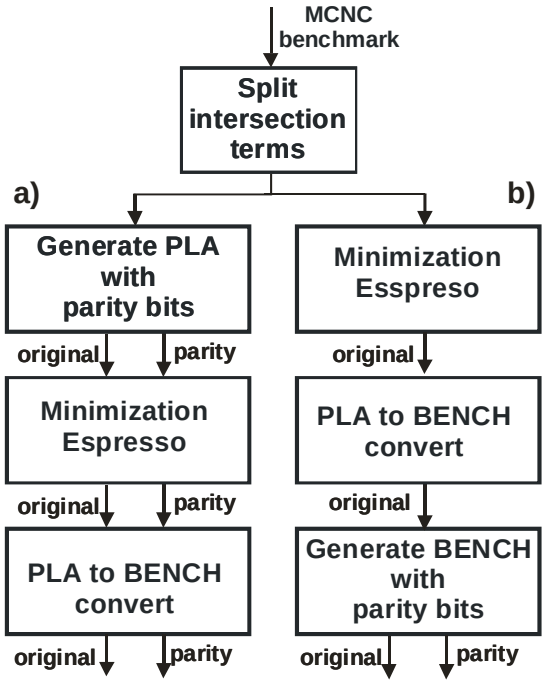


Figure 11. Two different flows for creating a parity generator

This task was divided into two experiments, see Figure 11. In the first experiment the two-level network was being modified, see Figure 11a. The results are shown in Table 12.

Table 12. Hamming code – PLA

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	ST	FS
alu1	4	8	84	1050	100	100
apla	5	45	105	233	100	98,3
b11	6	38	38	100	100	99,7
br1	4	50	59	118	100	95,9
al2	7	51	54	106	100	98,8
alu2	4	30	127	423	100	100
alu3	4	28	94	336	100	100
c17	2	2	3	150	100	100

The ST property was fulfilled in 7 cases and the FS property was fulfilled in 4 cases. The area overhead in many cases exceeds 100%. This means that the cost of one hundred percent fault coverage is too high. In these cases the TSC goal is satisfied for most tested benchmarks.

Table 13. Hamming code – XOR

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	ST	FS
alu1	4	8	13	163	100	100
apla	5	45	114	253	100	97,2
b11	6	38	73	192	100	99
br1	4	50	85	170	100	96,5
al2	7	52	109	210	100	99,1
alu2	4	30	52	173	100	100
alu3	4	28	44	157	100	100
c17	2	2	3	150	100	100

We then used an old method, where the original circuit described by a multi-level network is modified by additional XOR logic, see Figure 11b [A.1].

The results obtained from this experiment are shown in Table 13. The FS properties and the ST properties were fulfilled in the same cases as in the first experiment, but the overhead is in some cases smaller.

In the second set of experiments we tried to obtain a small area overhead, and the fault coverage was measured. In this case the minimum of parity bits is used (single even parity).The experiments are divided into two groups, a) and b), see Figure 11. The procedure is the same as described above.

Table 14. Single even parity – PLA

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	ST	FS
alu1	1	8	271	3388	100	98,9
apla	1	46	23	50	99,5	82,6
b11	1	37	3	8	89,9	77,3
br1	1	54	10	19	86,9	62,1
al2	1	52	4	8	97,3	91,7
alu2	1	29	47	162	100	91,2
alu3	1	26	32	123	100	92
c17	1	2	2	100	100	100

In the first experiment the two-level network of the original circuit was modified (Figure 11a). The results are shown in Table 14. The ST property is achieved in four cases, but the area overhead is smaller in five cases. The FS property is satisfied in one case.

In the last experiment, we modified the circuit described by a multilevel network (Figure 11b). The ST property was satisfied in four cases and the FS property in two cases. The area overhead is higher than 100% for most benchmarks, but the fault coverage did not increase, Table 15.

Table 15. Single even parity – XOR

Circuit	Parity nets	Original [LUT]	Parity [LUT]	Overhead [%]	ST	FS
alu1	1	8	10	125	100	100
apla	1	46	56	122	99,7	87,2
b11	1	37	36	97	93,9	81,4
br1	1	54	61	113	92,7	69
al2	1	52	23	44	97,9	93,2
alu2	1	29	44	152	100	91,1
alu3	1	26	39	150	100	91,6
c17	1	2	2	100	100	100

4.2.1.4. Summary

This chapter describes one part of the automatic design process methodology for a dynamic reconfiguration system. We designed concurrent error detection (CED) circuits based on FPGAs with a possible dynamic reconfiguration of the faulty part. The reliability characteristics can be increased by reconfiguration after error detection. The most important criterion is the speed of the fault detection and the safety of the whole circuit with respect to the surrounding environment.

In summary, FS and ST properties can be satisfied for the whole design, including the checking parts. This is achieved by using more redundancy outputs generated by the special codes.

All of our experiments apply combinational circuits only. Sequential circuits can be disjoint to the simple combinational parts separated by flip-flops. Therefore, the restriction to combinational circuits only does not reduce the quality of our methods and experimental results.

The FS property depends on class B. A low number of faults belonging to class B leads to a low FS property. The FS values for the MCNC [36] and ISCAS [23, 30] benchmarks used to validate our modified duplex system are shown in Table 16. Here "C" is benchmark circuit, "IN" is number of inputs, "OUT" is number of outputs, "AO" is area overhead, "FS" is the probability that a fault is detected by a code word, and "Ass" is the steady-state availability.

Table 16. Single even parity – PLA

C	IN	OUT	ORIG [LUT]	AO [%]	FS [%]
alu1	12	8	8	688	100
apla	10	12	45	53	83
b11	8	31	38	8	75
br1	12	8	50	20	63
al2	16	47	52	12	94
alu2	10	8	30	140	92
alu3	10	8	28	121	90
s1488	14	25	312	13	86
s1494	14	25	317	13	86
s2081	18	9	24	125	96
s27	7	4	4	75	72
s298	17	20	39	49	91
s386	13	13	51	39	71

The FS property expresses the probability that an existing fault is detected on a primary output of the circuit. If FS is fully satisfied (to 100%), a fault occurring in a circuit is always detected.

Special tools had to be developed to evaluate the area overhead and fault coverage. In addition to some commercial tools such as Leonardo Spectrum [38] and Synplify [37] we used format converting tools, parity circuit generator tools and simulation tools.

4.2.2. Single Parity and Parity Net Grouping

It is always necessary to perform some kind of decomposition when designing complex VLSI circuits, taking into account available components. Most methods proposed in the past start with a two-level Boolean network (sum-of-products) and try to decompose it into a multi-level network. The Boolean function is manipulated in order to extract subfunctions common to several of its parts. This is done either algebraically, by finding the function's common divisors (kernels) [39], by using computationally demanding Boolean methods [40, 41], or by functional decomposition [42, 43], recently based on BDDs [44, 45]. Nowadays, functional bi-decomposition plays a big role, and is generally usable for most applications [46, 47, 48].

Most of the methods mentioned above are primarily intended for single-output functions, even when they can be extended to multi-output functions. However, there is no method that strictly determines the relations between the outputs of the multi-output Boolean function. However, a method for selecting an appropriate code based on multiple net grouping described in [51] is slower than ours, and also the final modification is processed on a circuit described by multi-level logic. My partitioning method is based on grouping of *output* variables. There can be a relationship between several outputs of the function found. The proposed method is based on computing a measure of the "similarity" of the functions. When two Boolean functions are similar, there is a big probability they may efficiently share many logics. Thus, grouping these "similar" functions together could be advantageous, when output decomposition is needed. If appropriate output grouping is found, the resulting logic of the overall design is reduced to a minimum.

The method found its next application in on-line BIST (Built-in Self-Test) design [A.3, A.8]. Here the functions are grouped together to form the parity bits of the parity predictor. The parity groups are generated from the original circuit outputs, by successively XOR-ing them. The choice of outputs to be XORed plays an important role in the resulting area overhead.

4.2.2.1. Output Grouping

Let us assume there is a need to divide a circuit into several stand-alone blocks having a limited number of outputs (e.g., into PLAs, PALs, GALs). These blocks will have to be synthesized separately, since they cannot share internal signals. The blocks can share the input variables only. Such a case of decomposition will be denoted as *single-level partitioning*, since the number of levels of the circuit remains the same, see Figure 12.

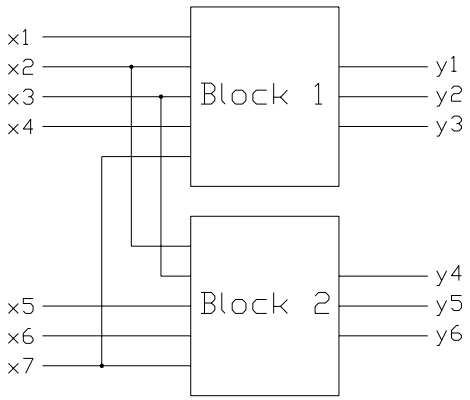


Figure 12. Single-level partitioning

Our task is to determine which outputs should be grouped together, i.e., to find an *output grouping*. If the outputs that are to be grouped together are properly determined, the complexity of the individual blocks is reduced.

Since the blocks cannot share the group terms (or subfunctions when they are designed as multi-level), the total complexity of the overall design must be increased, in comparison to the all-in-one design. When our technique is used, this cost of the decomposition is reduced to a minimum.

4.2.2.2. Similarity-Based Output Grouping

The method is based on the idea of joining functions (i.e., outputs of multi-output functions) that are somehow similar. Then there should be a big probability that the two functions will share a many logics. The task is to determine how to compute the measure of similarity of two Boolean functions.

The main idea is based on the following straightforward observations:

(I) Two equal functions are “very similar”

(II) Two inverse functions are also very similar, since they differ by one inverter only in the final design

These two criteria can be combined to form a new criterion:

(III) Two functions are similar, if a change in the value of one input variable induces a change in the values of *both* the functions, or the values of *both* functions do not change. This should be checked for all possible input variable changes.

To quantify the vague term “similarity” of two Boolean functions, a *scoring function* is introduced. To compute the value of the scoring function, all the minterms of the functions are processed. For each minterm each input variable value is switched and values of the outputs of the two functions are observed. If both values remain unchanged, the scoring function is increased by one, since these two functions have demonstrated the same behavior. If both values change, regardless the logic values, the scoring function is increased by one as well. In other cases, when one output value changes and the other does not, the score remains unchanged.

The complexity of the described algorithm is $O(n.2^n)$, where n is the number of input variables. For all the 2^n minterms n variable swaps are explored. The actual complexity can be reduced to $\frac{1}{2}\, n.2^n$. Only 0->1 swaps can be considered, since all the reverse swaps will yield the same result, thus doubling the score.

Two equal functions will obtain the highest score by this algorithm. Two inverse functions will also obtain the highest score.

This approach is straightforward, but it is inefficient, due to its prohibitively high complexity. However, we used it in our measurements, since functions described by minterms were needed for on-line BIST design [A.8].

Nevertheless, there is a much more efficient way to compute the scoring function: by using a Boolean difference. The Boolean difference of a function $f(x_0, \ldots x_n)$, with respect to an input variable x_i can be computed as:

$$\frac{\partial f(x_0,...,x_n)}{\partial x_i} = f(x_0,...,x_i = 1,...,x_n) \oplus f(x_0,...,x_i = 0,...,x_n) \tag{5}$$

As a result we obtain a Boolean function which is equal to 1, if a change of x_i induces a change of $f(x_0, \ldots x_n)$. Thus, the size of the Boolean difference function (i.e., number of its 1-minterms) describes the number of 1-minterms of $f(x_0, \ldots x_n)$ for which a change of x_i induces a change of $f(x_0, \ldots x_n)$. To derive the cubes for which two functions simultaneously change their value by changing the value of x_i, we compute the Boolean differences of these two functions with respect to x_i and compute their intersection, i.e., a Boolean product. The size of this product will correspond to the number of minterms for which both functions will change a value if x_i changes.

Similarly, the number of minterms for which both functions will not change a value if x_i changes can be computed using a Boolean indifference, i.e., a negation of a Boolean difference:

$$\frac{\bar{\partial} f(x_0,...,x_n)}{\bar{\partial} x_i} = f(x_0,...,x_i = 1,...,x_n) \Leftrightarrow f(x_0,...,x_i = 0,...,x_n) \tag{6}$$

As a result, the scoring function for functions $f(x_0, \ldots x_n)$ and $g(x_0, \ldots x_n)$ is computed as:

$$s = \left| \frac{\partial f(x_0,...,x_n)}{\partial x_i} \cdot \frac{\partial g(x_0,...,x_n)}{\partial x_i} \right| + \left| \frac{\bar{\partial} f(x_0,...,x_n)}{\bar{\partial} x_i} \cdot \frac{\bar{\partial} g(x_0,...,x_n)}{\bar{\partial} x_i} \right| \tag{7}$$

Note that the complexity of the computation of the score is a polynomial with n, so it can be used for any problem sizes.

4.2.2.3. Scoring Matrix

By computing a scoring function for each pair of output variables we obtain a *scoring matrix*. This is a symmetric matrix of dimensions (m, m), where m is the number of

output variables. The value in a cell $[i, j]$ represents the scoring function value of variables i and j. The outputs of the multi-output function are grouped together according the scoring matrix values. The output-grouping algorithm proceeds as follows:

1. Assign the first output variable to the first block.

2. Find the maximum scoring matrix value corresponding to outputs i and j. These outputs should be grouped together, since they have the highest "similarity value".

3. If one of these outputs is already assigned to a block, append the second output, if possible (the maximum number of outputs of the block is not exceeded).

4. If none is assigned, try to find an empty block and assign both outputs to this block.

5. If no free block is available, try to put them both into some block.

6. If there is not enough place to put both the outputs into one block, assign them randomly.

This simple algorithm yields an assignment of all output variables to the blocks, while the function's similarity is maximally exploited.

4.2.2.4. Experimental Results

We evaluated the efficiency of the algorithm on the MCNC benchmarks [36]. For each of the benchmark circuits we performed three experiments:

- First, the respective benchmark circuit was minimized by BOOM [35]. This experiment was performed to estimate the circuit size when partitioning is used.

- In the second group of experiments we divided the circuit into several blocks (b), and all the output variables were assigned to the individual blocks *purely at random*. Then the circuit was minimized by BOOM. One hundred experiments were performed and an average value was taken, to ensure sufficient statistical values.

- Finally the similarity-based output grouping method was used. We performed a similar experiment to the previously described one, but output variables were assigned to the blocks using the proposed method.

These three experiments show the differences between the all-in-one implementation of the benchmark circuit, the circuit divided into several blocks with randomly assigned outputs and our method. The number of blocks was selected according to the number of outputs of the circuit, to be somehow balanced with the number of outputs. However, any circuit may be divided into an arbitrary number of blocks without losing the efficiency of the algorithm.

The benchmark results are shown in Table 17. After the benchmark name the numbers of the primary inputs (*i*) and outputs (*o*) of the circuit are presented. The next column gives the number of gate equivalents [49] of the minimized circuit. Next, there is the number of blocks into which the circuit is being decomposed. The numbers of outputs of all the blocks are equal. The "Random output grouping" column shows the minimization results, for the experiment where the outputs are assigned to the blocks randomly. The "Similarity-based output grouping" labeled columns describe the results obtained by our newly proposed method. The last column, "*impr.*" shows the improvement against the previous (random) method.

Table 17. Output grouping results

			No decomp.		Random output grouping	Similarity-based output grouping	
bench	***i***	***o***	***GEs***	***blocks***	***GEs***	***GEs***	***impr.***
al2	16	47	206.5	5	244.0	218.0	10.7%
amd	14	24	334.5	3	460.0	429.0	6.7%
b2	16	17	989.5	4	2018.5	1807.5	10.5%
b7	8	31	81.0	4	105.0	88.5	15.7%
b11	8	31	81.0	4	105.5	87.5	17%
br1	12	8	130.0	3	215.0	186.5	13%
dk17	10	11	70.5	3	84.0	72.5	13.7%
exps	8	38	910.0	4	1473.0	1256.0	14.7%
luc	8	27	162.5	3	244.0	228.0	6.6.%

4.2.3. Parity Net Grouping and the Self-Checking Circuit

The above-described algorithm can be very efficiently applied to on-line BIST (Built-in Self-Test). The parity predictor is used to generate a proper output parity, see Figure 1. The parity predictor is designed by duplicating an original combinational circuit. The output nets of the duplicate circuit are XORed together to obtain output check bits. The predictor outputs are sequentially XORed, until one parity output is obtained (see Figure 13). Only two nets are XORed, together in each step, according to the scoring function that is computed.

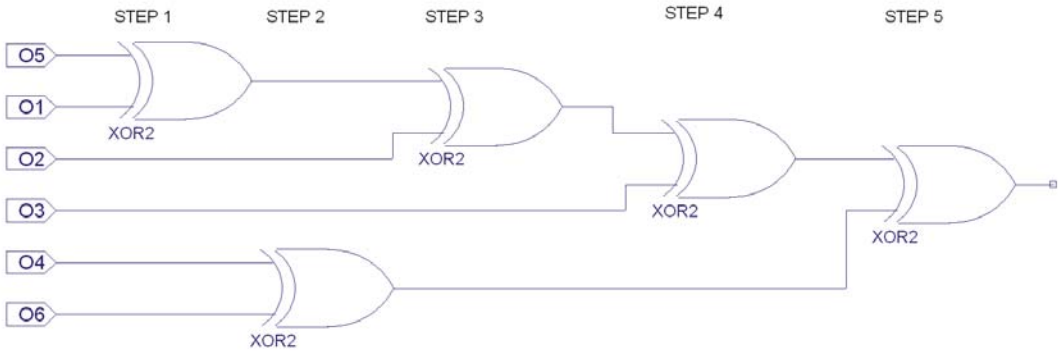


Figure 13. The parity prediction

4.2.3.1. Parity Bits Grouping Algorithm

An algorithm used for grouping the circuit outputs to form parity bits is described here. The selection of outputs to be joined by an XOR gate is of key importance for the final design area overhead.

Since the parity predictor is constructed by gradually joining the original circuit outputs by 2-input XOR gates, our primary task is to properly choose the two outputs to be joined in each step. The function similarity-based approach can be exploited very well. The basic idea of the algorithm depends on the following facts and assumptions:

(1) When two equal functions are joined by an XOR gate, the resulting value will be '0' for all minterms. If the values of two functions differ in a couple of minterms only, there will be only a small number of '1' values in the resulting XORed function. Experiments show that a low number of '1's at the output is very advantageous for the subsequent minimization process (Figure 15).

(2) Two inverse functions, when XORed, yield a '1' value for each minterm. If the output values of two functions are inverse but few minterms, there will be only a few '0' values in the result. This is also advantageous for minimization (Figure 15).

(3) And, consequently, if two functions are "similar", there is a big probability that they will share a lot of logic in the implemented design. If these functions are joined together, an overall area reduction is very probable.

The first two statements were based on the assumption that it is advantageous for minimization when the function values are either '0's or '1's for most of the minterms. This is documented in Figure 14. The typical dependency of an area overhead on the number of '1' values in the output is shown. 100-input and 20-output functions with 100 terms defined were minimized in this experiment. The number of '1's in the output varied between 10% and 90% while measuring the number of gate equivalents [49] of the circuit obtained after a minimization using BOOM [35]. We can see that low or high values of the ratio of '1's to '0's produce the best solutions.

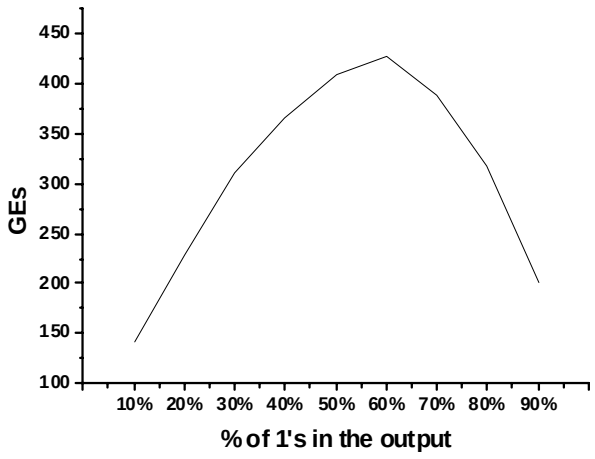


Figure 14. Dependency of the area overhead on the ratio of output '1's

The functions are described by the values of all minterms, i.e., functionally, not by a netlist. Thus, the final checker design has to be synthesized "from scratch". This gives us an advantage, since the synthesis process is able to recognize the similarity of the functions and to design the decoder efficiently.

4.2.3.2. Experimental Results

As in the previous set of experiments, our method is compared with a purely randomized method. All the values are measured as gate equivalents [49], obtained after synthesis. The area reduction obtained by the proposed method, in comparison with the random method, is shown in the "Red." column. The random assignments were run 500 times, and the values were averaged.

In some cases there is a very significant improvement in comparison with the random method, see Table 18.

Table 18. Comparison results

Circuit	Random [GEs]	Similarity [GEs]	Red.
alu1	967	156	83.9 %
apla	128	76	40.6 %
b11	36	21	41.7 %
br1	80	68	15 %
alu2	418	40	90.4 %
alu3	433	320	26.1 %
s1488	364	241	33.8 %
s386	87	73	16.1 %

The results obtained by our design methodology for a highly reliable system was validated on the MCNC [36] and ISCAS [30] benchmarks. The results are shown in Table 19. Availability calculation methodology is described in Section 7.

Table 19. Availability parameters

CIRCUIT	AO [LUT]	PARITY NETS	AO [%]	C	D	Ass [%]	DIFFERENCE FROM DUPLEX
alu1	55	1	687	0	0	1	0,000022
alu1	16	2	200	0	0	1	0,000022
apla	24	1	53	1	109	0,999991	0,000013
apla	22	2	49	1	92	0,999993	0,000015
b11	3	1	8	42	59	0,999994	0,000016
b11	7	2	18	38	52	0,999994	0,000016
br1	10	1	20	47	154	0,999988	0,000010
br1	23	2	46	41	142	0,999987	0,000009
alu2	42	1	140	0	58	0,999991	0,000013
alu2	40	2	133	0	52	0,999991	0,000013
alu3	34	1	121	0	63	0,99999	0,000012
alu3	33	2	118	0	63	0,999989	0,000011
s1488	41	1	13	94	321	0,999996	0,000018
s1488	94	2	30	80	267	0,999996	0,000018
s386	20	1	39	15	176	0,999988	0,000010
s386	25	2	39	8	149	0,999989	0,000011

Here "Circuit" is benchmark circuit, "AO" is area overhead, "PN" is number of parity nets, "C" and "D" is number of undetected faults that are not detected by the code word, and "Ass" is steady-state availability

4.2.3.3. Summary

A novel circuit decomposition and output grouping method has been presented here. It is based on an evaluation of the "similarity" of Boolean functions. Functions that are found to be "similar" share a lot of logic. Therefore, when they are grouped together, many resources are saved. The output grouping retains the two-level nature of the circuit; hence we call it single-level partitioning.

A very efficient application of the method to on-line BIST design is proposed. Here the circuit outputs are joined together by XOR gates to form a parity predictor. The parity predictor outputs are compared with the outputs of the original circuit, and thus the proper circuit function is checked. The proposed method helps to minimize the parity predictor logic overhead. The area overhead is sometimes reduced by more than 90% in comparison with a random method.

The results obtained using our method are presented and compared with a random-based approach. Standard MCNC [36] and ISCAS [30] benchmarks were used.

5. Architecture of a Modified Duplex System (MDS)

The TMR structure is unsuitable when a high area overhead is unacceptable. Some hybrid architecture must be used. TMR architecture and a hybrid system, e.g., the modified duplex system with a comparator and some CED technique are compared in [59, 61]. A technique based on Duplication With Comparison (DWC) and the Concurrent Error Detection (CED) technique are described in [60, 62].

The fault tolerant system proposed in this chapter is based on DWC-CED with reconfiguration. This thesis is devoted to methods for maximally increasing the dependability parameters with maintaining the minimal area overhead. The complex structure implemented in each FPGA is divided into small blocks, where every block satisfies TSC properties. This approach can detect a fault before the fault is detected by the output comparator.

Our previous results show that it is difficult to satisfy the TSC property on 100%, so we have proposed a new structure (MDS) based on two FPGAs , see Figure 15.

Each FPGA has one primary input, one primary output and two pairs of checking signals OK/FAIL. The probability of the information correctness depends on the FS property. When the FS property is satisfied only to 75%, the correctness of the checking information is also 75%. This means that the "OK" signal gives correct information for 75% of the errors that have occurred errors (the same probabilities for both "OK" and "FAIL" signals).

To increase the dependability parameters we must add two comparators, one for each FPGA. The comparator compares the outputs of the two FPGAs. The fail signal is generated when the output values are different. This information is not sufficient to determine which TSC circuit is wrong. Additional information to mark out the wrong circuit is generated by the original TSC circuit. The probability of information correctness depends on the FS property, and in many cases it is higher than 75%. When the outputs are different and one of the TSC circuits signals a fail function, the wrong FPGA is correctly recognized. Correct outputs are processed by the next circuit.

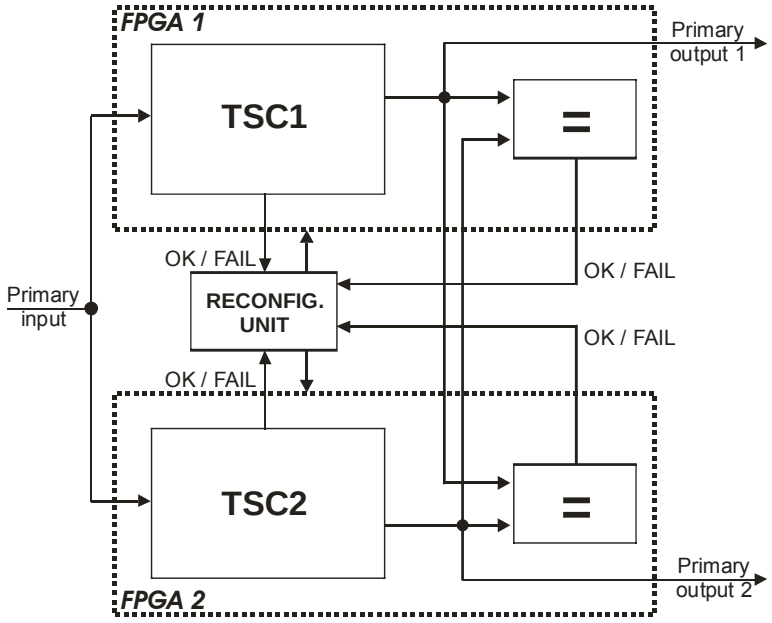


Figure 15. MDS architecture

The reconfiguration process is initiated after a fault is detected. The reconfiguration solves two problems: localization and correction of the faulty part. The time needed to localize the faulty part is not negligible and must be included in the calculation of the dependability parameters. We select only the faulty FPGA and we reconfigure it in our solution. This means that we do not localize the faulty block inside the compound design. The time taken to localize a fault and to reconfigure the faulty part can be similar to the time taken to reconfigure the whole FPGA. The whole FPGA reconfiguration also repairs the faults which occurred in an unused logic. The reconfiguration process can also be initiated when one of the two FPGAs produces the "FAIL" signal. This situation occurs when a fault is detected by one of the small TSC blocks inside the compound design. Fault propagation to the primary outputs may take a long time.

When the outputs are different, and both circuits signalize a correct function, we must stop the circuit function and the reconfiguration process is initiated for both FPGA circuits. After the reconfiguration process is performed, the states of the two FPGAs are synchronized. This means that our modified duplex system can be used in an application where system reset synchronization is possible.

5.1. Implementation of TSC Based on MDS

Each FPGA contains a TSC circuit and a comparator. The TSC circuit is composed of small blocks, where each block satisfies the TSC property. The structure of the compound design satisfying the TSC property is shown in Figure 16.

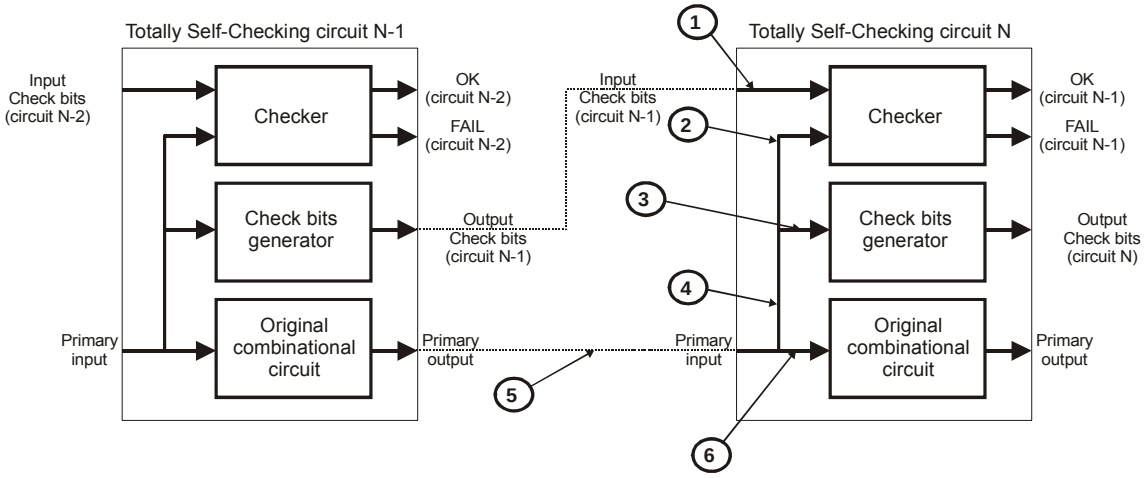


Figure 16. Proposed structure of TSC circuits implemented in FPGA

We can assume six places where an error is observable for this compound design. We assume, for simplicity, that an error occurring in the check bit generator will be observable at the parity nets (number 1), and an error occurring in the original circuit will be observable at the primary outputs (number 5).

The checker in block N will detect the error if it occurs in net number 1, 2, 4 or 5. If an error occurs in net number 3 or 6, it will be detected in the next checker (N+1). The method used to satisfy the TSC property for the compound design is described in greater detail in [A.16].

Not every small block (in the compound design) satisfies the TSC property to 100%. The TSC property depends on the FS and ST properties, which are also not satisfied to 100%. For availability computations, we find the block with the lowest FS property value in the compound design.

5.2. HW Emulation of MDS in FPGA

Every reaction to an input vector change must by calculated in SW simulation. Each simulation step takes many processor cycles especially for circuits with many gates. On the other hand if we process one simulation step, the time needed for calculation is equal to one system cycle. However, the results should be compared and evaluated concurrently. This leads us to utilize the HW emulator.

A fault injection into the implemented circuit allows us to calculate the dependability parameters more precisely. Moreover, some faults such as interconnection faults can be tested only for a really implemented circuit. The SW simulator is not able to test faults of this type, because the final circuit structure is not known. HW emulation enables more types of faults to be tested.

The FPGA circuit selection process depends on the demands on the tested structure and on the granularity of the reconfiguration [10, 13, 14, 15]. Atmel AT40K satisfies all these requirements. It allows fine grain dynamical reconfiguration of the implemented circuit. The reconfiguration process is controlled and realized by an on-chip AVR microcontroller. This microcontroller also stores the tested vectors and simulation results.

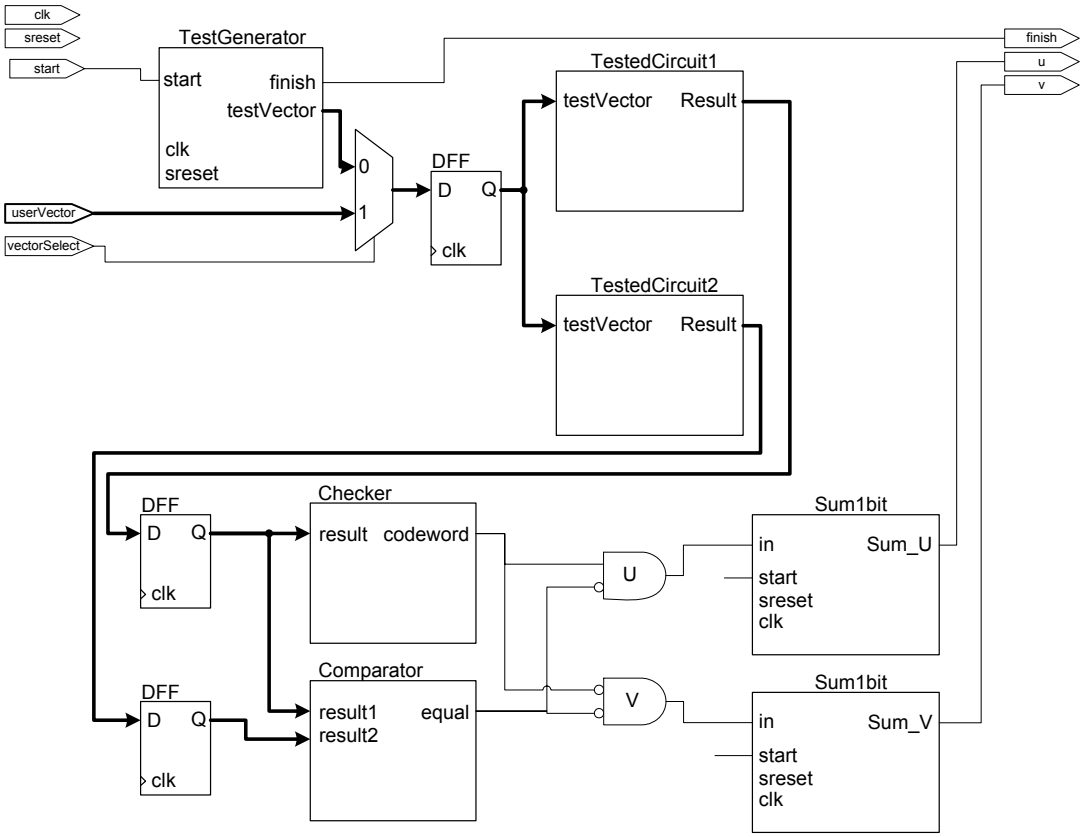


Figure 17. Final structure for the benchmark test

The HW emulator consists of a collection of SW tools and the Atmel AT40K testing board. The SW tools are used to convert benchmarks from ".pla" format to ".vhdl" format. These SW tools also enable the self-checking modification of the original circuit. Synthesis and the mapping process are fulfilled manually by the Atmel FPSLIC design tool. The final bitstream is put into Atmel FPGA. Finally the tested vectors are loaded into the AVR microcontroller. The exhaustive test must be processed for the on-line test. For this case the testing vectors are generated automatically and only the set of tested faults is loaded.

The test is divided into two parts. These parts are composed of a safe test set and a risk test set. The risk test set is composed of interconnection tests and can cause shorts. Our HW emulator can test only a part of the safe test set composed mainly of look-up-table tests. This test set covers only 11% of the AT40K bitstream size. The other parts belong to the interconnection between the cells (23%), the interconnection inside the cells (36%) and other configuration bits used, e.g., clock distribution, the input/output cell and the RAMs (30%).

The test structure implemented in FPGA can be divided into four parts: circuit under test, test vectors generator, check block and results processing block (Figure 17). The self-checking testing part is shown in greater detail in Figure 18. The two most important blocks are highlighted in light gray.

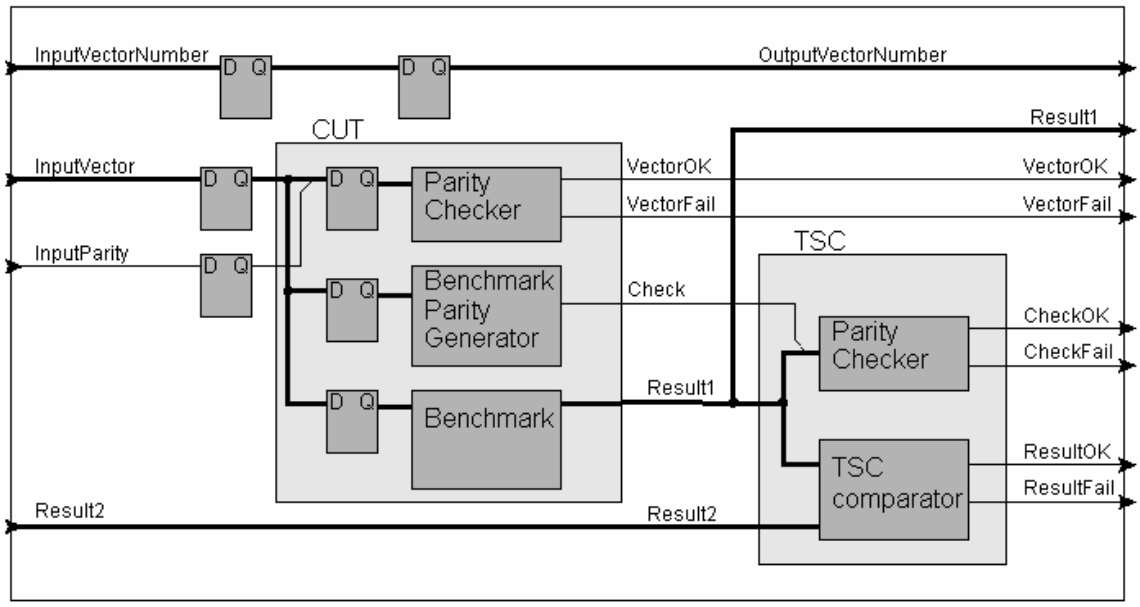


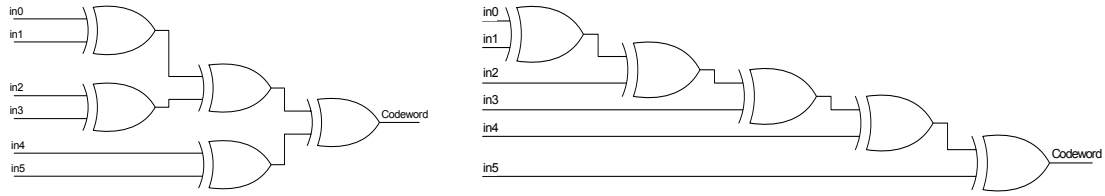
Figure 18. Design scheduling of the self-checking circuit

The final testing area of the implemented design is calculated individually for each important block. These blocks are the parity checker, the original benchmark, the parity generator and the comparator. The area of the parity checker and the comparator depends on the number of checked or compared nets, and can be calculated before final implementation.

5.2.1. Checker

The final area of the even parity checker is shown in Figure 19. The final area does not depend on the realized structure, and is equal to both the optimal and the unbalanced

variant. Only the final delay depends on the realized structure. There is a smaller delay for the optimal tree structure than for the unbalanced tree structure, see Table 20.



Optimal tree Unbalanced tree

Figure 19. Even parity checker and length of tree

The solution presented above is for the even parity checker realizing only the "CheckOK" signal. The odd parity checker must be used to generate the "CheckFAIL" signal. The final area of the checker is the sum of the even and odd parity checkers. The area of even and odd parity is equivalent.

Table 20. Even parity checker and length of tree

Tree	Net length
Optimal tree	$L = \lceil \log_2 n \rceil$
Unbalanced tree	$L = n - 1$

The number of LUTs *M* used for the even parity checker is:

$$M = \left\lceil \frac{n-1}{3} \right\rceil \tag{8}$$

where M is the number of LUTs and n is the number of inputs.
Even and odd parity can by calculated by the follow equations:

a) Even parity $codeword = \overline{in_0 \oplus in_1 \oplus in_2 \oplus \ldots \oplus in_{n-1}}$ (9)

b) Odd parity $codeword = n_0 \oplus in_1 \oplus in_2 \oplus \ldots \oplus in_{n-1}$ (10)

5.2.2. Comparator

The structure of the comparator is shown in Figure 20. The comparator in this simulator is used to analyze whether the primary output is correct. In the final solution, the comparator is used only for outputs leaving an FPGA. Only checkers are used inside the design.

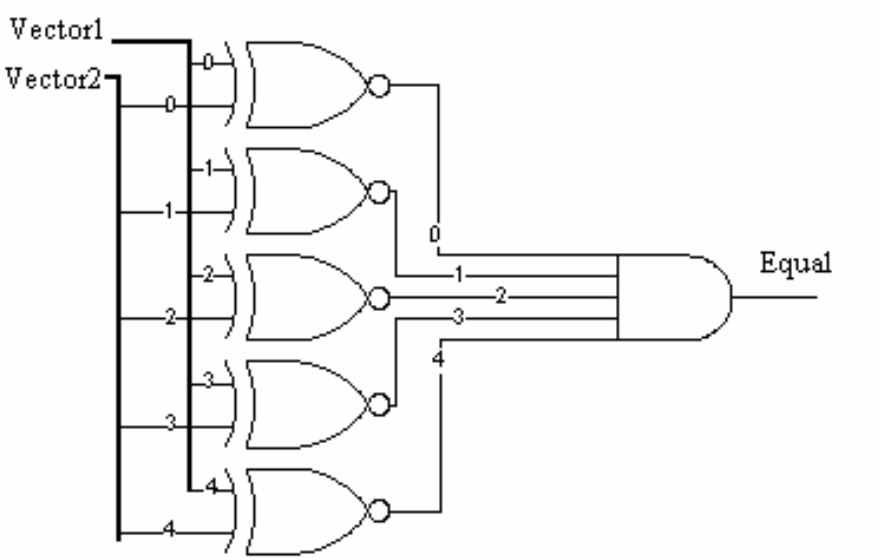


Figure 20. Comparator

The output of the comparator can be calculated by the following equation:

$$equal = \overline{r_0 \oplus s_0} \cdot \overline{r_1 \oplus s_1} \cdot \overline{r_2 \oplus s_2} \cdot \ldots \cdot \overline{r_{n-1} \oplus s_{n-1}} \,. \tag{11}$$

The final area does not depend on the realized structure, and it is equal for both the optimal and the unbalanced variant. Only the final delay depends on the realized structure. There is a smaller delay for the optimal tree structure than for the unbalanced tree structure, see Table 21.

Table 21. Length of tree

Tree	Net length
Optimal tree	$L = 1 + \left\lceil \log_4 \left\lceil \frac{n}{2} \right\rceil \right\rceil$
Unbalanced tree	$L = 1 + \left\lceil \frac{\left\lceil \frac{n}{2} \right\rceil - 1}{3} \right\rceil$

The solution described above is only for the "CheckOK" signal. For the "CheckFAIL" signal, the solution must be doubled and one inverter must be added. The number of LUTs *M* used for the comparator is:

$$M = \left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{\left\lceil \frac{n}{2} \right\rceil - 1}{3} \right\rceil \tag{12}$$

where M is the number of LUTs and n is the number of inputs.

5.3. Emulation Process

The MCNC benchmarks [36] are converted, modified and tested by the simulation design flow shown in Figure 21.

The white blocks in the design flow are used for both SW simulator and HW emulator. The gray blocks are used only for HW emulation. The VHDL code in the last white block is synthesized for XILINX Virtex FPGA in cases when an SW simulator is used. The Atmel FPSLIC FPGA is used as the HW emulator.

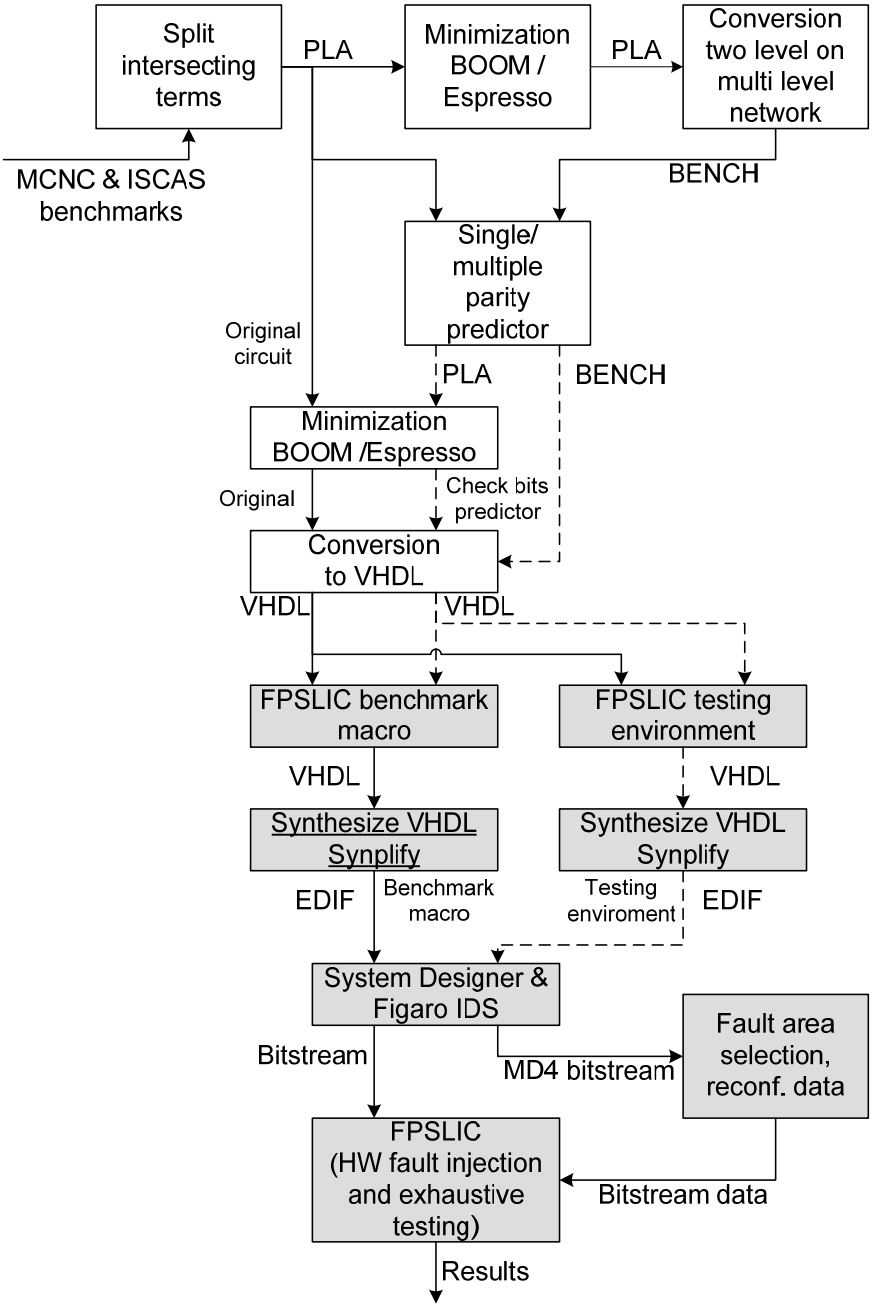


Figure 21. Emulation Process

5.4. Emulation Results

The simulation design methodology shown above was used to generate the bitstream for many MCNC benchmarks [36]. The benchmark's bitstream was loaded into Atmel FPGA. The faults were injected only into the used parts of LUTs. Fault injection into unused logic would only increase the number of undetected faults. The unused logic is generated in cases when fever than 4-inputs LUTs are used. It is obvious that the faults in unused logic are hidden, because there is no way of detecting them.

Table 22. Results obtained from our HW emulator without routing cells

Circuit	Input	Output	Original circuit [LUTs]	Parity generator [LUTs]	Area overhead [%]	Number of all faults	A (hidden faults)	B (detected faults)	C (undetected faults)	D (temporary detected)	ST coverage[%]	FS coverage [%]	Test time [s]
alu1	12	8	8	47	587.5	656	0	656	0	0	100.00	100.00	0.68
alu2	10	8	44	47	106.8	1072	109	935	0	28	89.83	97.39	0.29
alu3	10	8	45	45	100.0	1044	130	877	8	29	86.78	96.46	0.29
apla	10	12	48	25	52.1	900	141	625	5	129	83.78	85.11	0.25
br1	12	8	50	15	30.0	810	141	456	69	144	74.07	73.70	0.84
s1488	14	25	310	50	16.1	4286	638	3060	85	503	83.13	86.28	17.64
s1494	14	25	276	53	19.2	3938	645	2785	67	441	81.92	87.10	16.21
s2081	18	9	22	25	113.6	536	22	494	0	20	95.90	96.27	35.14
s386	13	13	57	18	31.6	976	170	646	25	135	80.02	83.61	2.02

The HW emulation results are shown in Table 22. Here "Circuit" refers to the benchmark circuit, "Input and Output" are the numbers of primary inputs and primary outputs, "Original circuit" is the number of LUTs used for the original circuit, "Parity generator" is the number of LUTs used for the parity generator, "Area overhead" is the area needed for the parity generator as a percentage, "Number of all faults" are all tested faults, "A, B, C, D" are the classes derived by our fault classification, "ST, FS" are self-testing and fault secure properties, and "Test time" is the time needed for full test execution.

Table 23. SW simulation/HW emulation time

	Inputs	SW simulation [s]	HW estimation [s]	HW emulation [s]	SW/HW time rate
alu1	12	34.0	0.92	0.92	37.0
alu2	10	9.0	0.40	0.41	22.0
alu3	10	6.9	0.39	0.41	16.8
apla	10	6.0	0.32	0.34	17.6
b11	8	0.8	0.05	0.06	13.3
br1	12	18.0	1.08	1.10	16.4
s1488	14	2406.3	23.76	23.82	101.0
s1494	14	2518.9	21.79	21.84	115.3
s2081	18	1217.9	49.30	49.30	24.7
S27	7	0.1	0.01	0.03	3.3
s386	13	677.7	2.48	2.49	272.2

The results show that for all benchmarks the FS property is higher than 70% of fault coverage. The area overhead depends on the benchmarks that are tested. For 50% of the benchmarks the area overhead is less than 50%. The “alu” benchmarks are a typical problem for a single parity generator. The area overhead decreases rapidly when the two parity groups are used for “alu” benchmarks.

The SW simulator is slower than the HW emulator. The simulation time results are shown in Table 23.

6. Proof of MDS Optimality

To evaluate the influence of a sequence of SEU faults, a more precise definition of a "single fault" is needed. We use availability computation for dependability analysis. In the following text we will assume that "damage to a single data item" is defined as follows:

- It will occur at a single time that is arbitrarily located on the time axis.
- The fault can change a data item located within the FPGA configuration memory. Both FPGAs can be affected with the same probability. We assume the single fault changes only one bit of the FPGA configuration memory. Each bit in the FPGA configuration memory can be attacked with the same probability.
- The time between any two single faults is long enough for a single fault to be successfully detected and corrected. Otherwise, it is a multiple fault.

Some basic rules are defined to calculate the availability parameters. We assume that:

- There is at least one input vector coming between two SEUs that make an output differ from the normal operation.
- SEUs impacting an unused logic do not change the function of the part that is used. These faults are hidden faults.
- The comparator and the checker are fully TSC.
- The area overhead of the comparator and the checker is negligible.
- The reconfiguration unit loads correct configuration data after a fault is detected.
- The time needed to reconfigure the faulty part depends on the configuration data size.
- A fault that occurred in unused logic does not damage the whole FPGA.

6.1. Reliability Model

The Markov model shown in Figure 22 describes our architecture.

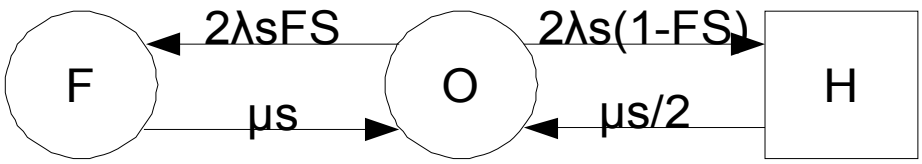


Figure 22. Model of our modified duplex system

There are three states (**O**, **F**, **H**).

The **O** state (operational) represents the regular fault-free state of the system, where both FPGAs are operating correctly. This means that the fail function is signalized neither by the TSC circuit nor by the comparator.

There is a transition from the **O** state to the **F** state (one FPGA is faulty) corresponding to the situation when a fault occurs in one FPGA and this fault is detected by one of the TSC circuits. The system enters this state with a probability FS. λ is the failure rate for one bit of a configuration memory and s is the size of a configuration memory. Number 2 (in the 2λsFS expression) means that one of two FPGAs can be affected by SEUs. The reconfiguration process is initiated only for the faulty FPGA. The repair rate is represented by μ. The second FPGA is running correctly and performs the function of the system.

Some faults are not detected when the output vector is an incorrect codeword. The probability that an occurred fault causes an incorrect codeword is equal to **1-*FS***. In this case the system comes to the state **H**.

The **H** (hazard) state means that the system is in the hazard state. The hazard state is detected (e.g., by comparators), because the output vectors are not identical. Both FPGAs have to be reconfigured in this case. The repair rate is equal to $\mu/2$, because we reconfigure each FPGA separately. If we are able to reconfigure both FPGAs at the same time, the availability parameters will increase.

$$\begin{aligned} & 2s\lambda\, p_O - \mu s\, p_F - \frac{\mu s\, p_H}{2} = 0 \\ & \mu s\, p_F - 2s\lambda FS\, p_O = 0 \\ & \frac{\mu s\, p_H}{2} - 2s\lambda\left(1 - FS\right) p_O = 0 \\ & p_O + p_F + p_H = 1 \end{aligned} \tag{13}$$

The model described here introduces four parameters: failure rate (λ), repair rate (μ), fault security (***FS***) and the configuration memory size (s). These parameters are discussed in the next section. Now let us transform the Markov model into a system of equations describing the steady state probabilities of each of the states (Equations 13). The system of equations is completed by a normalisation condition.

$$A_{SS} = p_O + p_F \tag{14}$$

The value of the steady-state availability A_{SS} is the sum of probabilities for all working states (Equation 14).

6.2. Evaluation of the Reliability Model

First, we discuss the model parameters. The failure rate (λ) depends on the probability that the impacting SEUs will change a bit in the FPGA configuration memory. Due to this fact we took into account the result presented in [4] and set the failure rate to:

$$\lambda = 1.8\,e^{-5}\ [h^{-1}] \tag{15}$$

The repair rate (μ) depends on the time needed for reconfiguring an FPGA. The clock frequency was set to 25 MHz. The configuration memory size ***s*** (needed for each benchmark) was calculated as the product of the configuration memory size for AT94K40 ATMEL FPSLIC and the circuit area overhead (AO[%]).

$$s = 233k \cdot AO\,[bits] \tag{16}$$

The graphs in Figures 23, 24, 25, 26 were constructed by solving equations (13) and (14). We used equations (13) and (14) for the following calculations. Firstly the circuit area overhead was fixed to 50 percent. The FS parameter varies from 0 to 100% FS. The availability parameter increases with higher FS, see Figure 23.

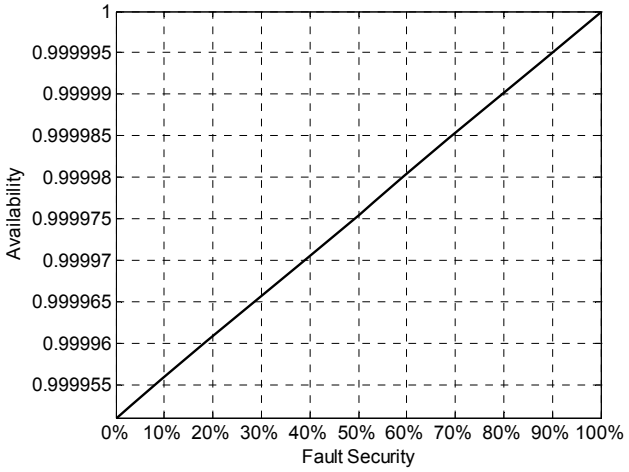


Figure 23. Availability for 50% overhead

The curve in Figure 23 is generally described by equation (17).

$$A_{SS} = \frac{2\,FS\,\lambda + \mu}{4\,\lambda - 2\,FS\,\lambda + \mu} \tag{17}$$

In the second case, FS is 80% and the area overhead varies from 0 to 100%.

Figure 24 shows that a higher area overhead means a low availability parameter, but the availability parameter decreases more slowly than in our first case, when the value of the FS parameter is changing.

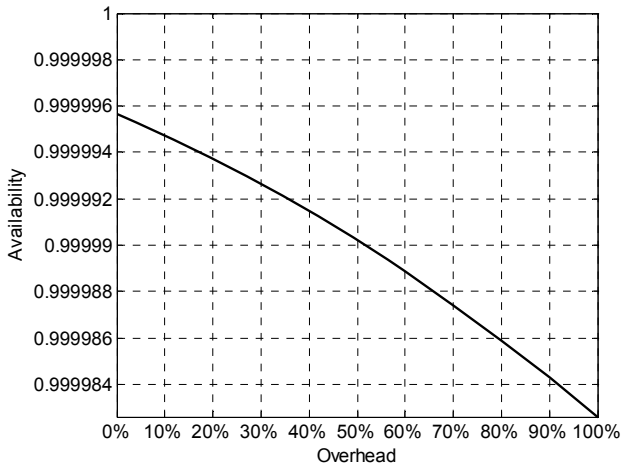


Figure 24. Availability for 80% FS

In the third case, we show the relation between the area overhead, the FS property and the availability. The results are shown in Figure 25. One point (number 1 in Figure 25) corresponds to the standard duplex system. The availability of the standard duplex system is 0,999978248.

When both the area overhead and FS are 0% (the front corner in Figure 25), the availability of our system would be the same as for the standard duplex system without any detection of a faulty FPGA.

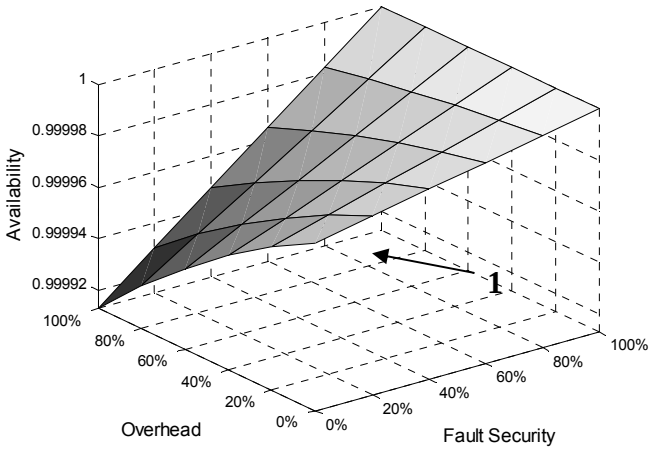


Figure 25. Availability 3D graph

The graph in Figure 25 describes the dependency of AO on FS parameterized by the availability. One curve (number 1 in Figure 25, Figure 26) corresponds to the standard duplex system. Due to this, when FS is 50%, the area overhead must be less than 40%. In other cases the system is worse than the standard duplex system with respect to availability.

Arrow 2 in Figure 26 shows the area where the system is worse than a standard duplex system with respect to availability. Arrow 3 shows where the system is better than a standard duplex system with respect to availability. Each curve in Figure 26 represents one value of the availability parameter.

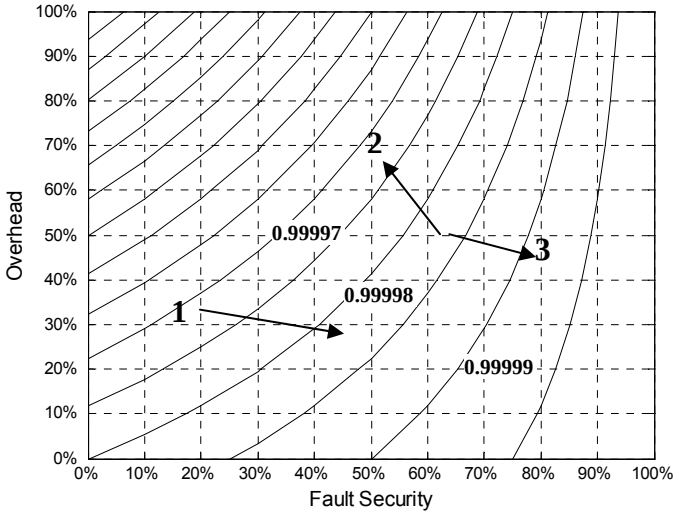


Figure 26. Curves of availability values

6.3. Main results

The results obtained by our case study were validated on the MCNC [36] and ISCAS [30] benchmarks. Our results are shown in Table 23. The fault security (***FS***) and the area overhead (***AO***) are summarized in Table 24, where the results obtained by the computation of the models are also included.

Here "CIRCUIT" is benchmark circuit, "AO" is area overhead, "FS" is the probability that a fault is detected by code word, and "Ass" is steady-state availability.

Table 24. Availability parameters

CIRCUIT	AO [%]	FS [%]	ASS [%]
alu11	687,5	100	1
apla	53,3	82,8	0.9999912
b11	7,9	75,5	0.9999938
br1	20,0	62,9	0.9999847
al2	11,5	94,3	0.9999985
alu2	140,0	92,5	0.9999906
alu3	121,4	90,3	0.9999897
s1488	13,1	86,3	0.9999962
s1494	12,9	86,3	0.9999962
s2081	125,0	96,2	0.9999958
s27	75,0	72,2	0.9999815
s298	48,7	91,0	0.9999957
s386	39,2	71,1	0.9999878

6.4. Generalization

The availability of the original duplex system is 0,999978248, and the availability of TMR architecture is 1. If we compare the original duplex system with our modified duplex system we increase the availability parameter for all tested benchmarks. The availability parameter is the same as for a triplex system in case when the FS property is 100%. We found that availability depends more on the FS property than on the area overhead. When the FS is not 100% achieved, the area overhead is strictly limited by the availability value of the standard duplex system. When this value is surpassed, the availability is inferior to the standard duplex system. We can summarize that for the benchmarks tested, the availability parameters have increased. For example, when "apla" has 82.8 % of FS and 53 % of the area overhead, the time when the system is unavailable is about 2.5 times shorter than for the standard duplex system. The dependability parameters of our modified duplex system are better than the standard duplex system and a little worse than or equal to the TMR system (which has a greater area overhead than our reconfigurable and duplex system).

7. Design Methodology

The design methodology for creating a TSC circuit is shown in Figure 27. To generate the output parity bits, all the output values have to be defined for each particular input vector. Unfortunately, the benchmark definition files do not do this. Only some output values are specified for each multi-dimensional input vector, while the rest are assigned as don't cares; they are left to be specified by another term. Thus, to be able to compute the parity bits, we have to split the intersecting terms, so that all the terms in the truth table are disjoint.

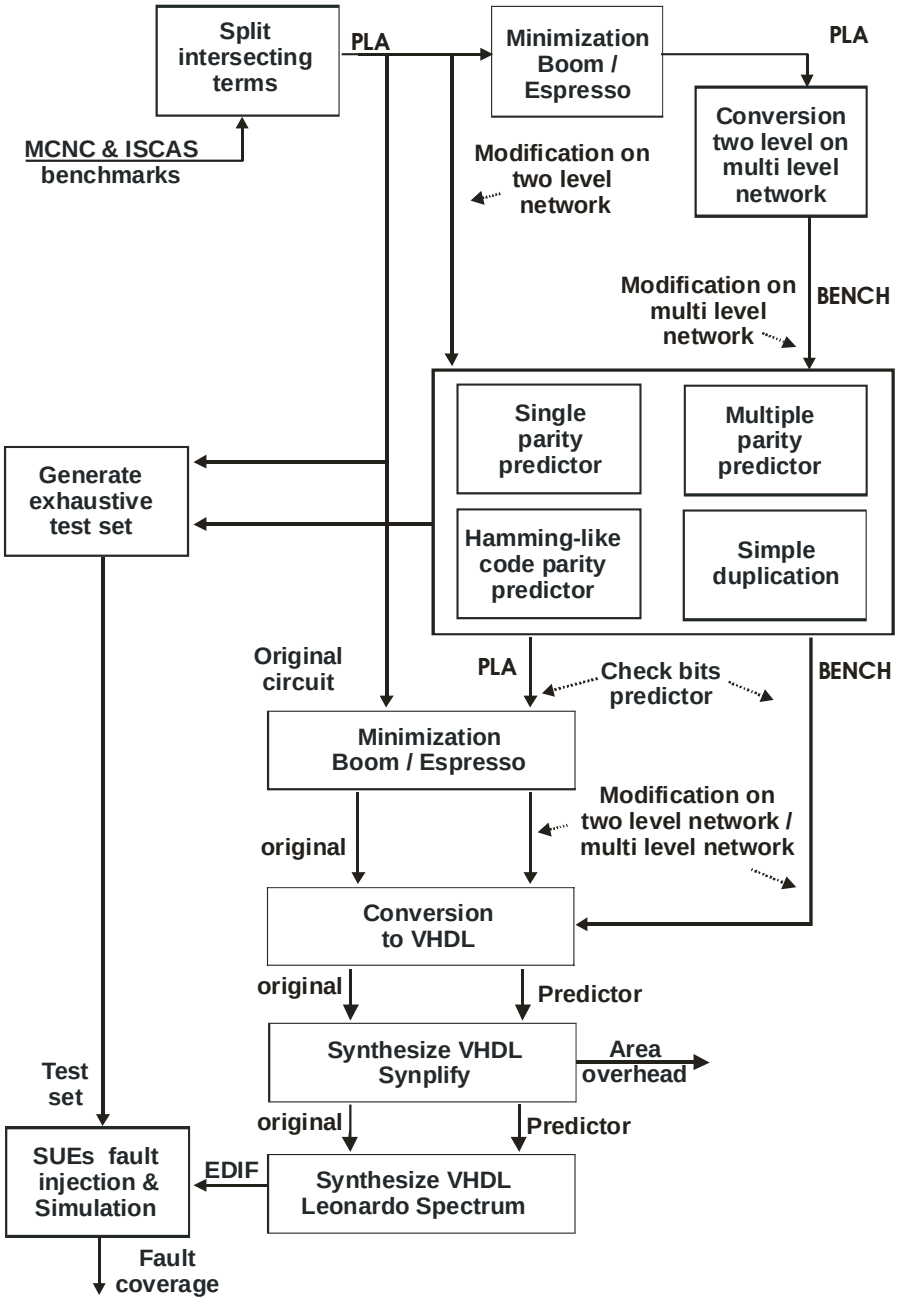


Figure 27. Design methodology flow

In the next step the original primary outputs are replaced by parity bits. Two different error codes were used to calculate the output parity bits (single even parity code and

multiple parity code), but our design methodology also enables the use of Hamming-like code or standard duplication. Another tool was used in the case where the original circuit was modified in the multilevel logic. This tool is described in [A.1]. The two circuits generated in the first step (original circuit and parity circuit) are processed separately to avoid sharing any part of the circuit. Each part can be minimized by the BOOM [35] or Espresso tool [34]. The final area overhead depends on the software that was used in this step. Many tools were used to reduce the area of the parity bits generator. BOOM was used to minimize the final area. In this step the area overhead is known, but we can decide whether the fault coverage is sufficient.

In the next step, "pla" format is converted into "bench" format. "Bench" format was used because the tool that generates the exhaustive test set uses this format. An exhaustive test set has 2n patterns and we used it to evaluate the TSC goals.

Another conversion tool is used to generate two VHDL codes and the top level. The top level is used for incorporating the original and parity circuit generator. In the next step the synthesis process is performed by the Synplicity Synplify Pro tool. The constraint properties set during the synthesis process express the area overhead and the SEU fault coverage. If the maximal frequency is too high, hidden faults occur during fault simulation. The hidden faults are caused by circuit duplication. The size of the area overhead is obtained from the synthesis process. The final netlist is generated by the Leonardo Spectrum software. The fault coverage was obtained by simulation using our software [31].

To evaluate the area overhead and the fault coverage special tools had to be developed. In addition to some commercial tools such as Leonardo Spectrum and Synplify, we used format converting tools, parity circuit generator tools and simulation tools.

At first, the area minimization and term splitting is performed for the original circuit by BOOM [35]. The Hamming code generator (or single parity generator) is generated by the second software. These two circuits are minimized again with BOOM. The next two tools convert the two-level format into a multi-level format. The first of them converts a "pla" file to "bench", and the second converts "bench" to VHDL. The second software is used for generating the final circuit in "bench" format due to its further usage in the exhaustive test set generator. The format converting software and parity generator software were written in Microsoft Visual C++. The netlist fault simulator was written in Java. The parser source code was used to parse the netlist generated by the two commercial tools mentioned above.

Our modified duplex system based on two FPGAs with a highly reliable system design methodology has been presented here. The design methodology enables an appropriate code to be selected, taking into account the system requirements.

Our methods we can use for designing a totally self checking circuit. The selected method depends on the final area overhead and the SEU fault coverage. When a highly reliable system is required and the area overhead can be high, it is better to use duplication or a Hamming-like code. These two methods ensure that fault security is fulfilled one hundred percent, and the Ass parameter is also equal to hundred percent.

When a low area overhead and a highly reliable system are required, it is better to use the simple or multiple parity predictor.

Our highly reliable structure ensures that the final system is better than a standard duplex system with 0,999978248 of Ass [A.6].

8. Conclusions and Future Work

A modified duplex system based on two reconfigurable FPGAs has been presented. Our MDS architecture increases the dependability parameters in comparison with the standard duplex system. The dependability parameters have been increased due to the reconfiguration process and the two methods of SEU detection that are used. The first method compares the primary outputs of each FPGA, and the second signalizes the faulty FPGA.

The whole proposed system has been described by a dependability Markov model. This model was used for computing the reliability and availability parameters for SEU fault models. The results for the MCNC [36] and ISCAS [30] benchmarks have been compared with those of the standard duplex system. We found that availability depends more on the FS property than on the area overhead. When the FS is not 100% achieved, the area overhead is strictly limited by the availability value of the standard duplex system. When this value is surpassed, the availability is inferior to the standard duplex system. We can summarize that for the benchmarks tested, the availability parameters have increased. For example, when "apla" has 82.8 % of FS and 53 % of the area overhead, the time when the system is unavailable is about 2.5 times shorter than for the standard duplex system. The dependability parameters of our modified duplex system are better than the standard duplex system and a little worse than or equal to the TMR system (which has a greater area overhead than our reconfigurable and duplex system).

Single parity is most suitable code for the self-checking circuit, due to its low area overhead and high fault coverage. Other error detection codes lead to higher fault coverage, but the area overhead and the number of possible faults are also higher. Single parity is good trade-off between the area overhead and fault coverage.

Faulty FPGA can be reconfigured as a whole rather than partially. Partial reconfiguration involves initiating the localization process. The partial reconfiguration process with fault localization takes a longer time than the whole FPGA reconfiguration process without fault localization. Moreover, hidden faults that have occurred in unused logic are fixed because the FPGA reconfiguration (reprogramming) includes unused logic. The circuit design implemented in both FPGA must enable the self synchronization process which synchronizes both FPGAs after one of them is reconfigured. In many applications, it is sufficient to reset both FPGAs.

A new fault classification has been proposed. The four classes were proposed to describe possibility of all situations of occurred faults.

8.1. Contribution of the dissertation:

The main result of the dissertation thesis is a fault tolerant design methodology based on self-checking circuits implemented with using FPGAs. The methodology describes the design steps of the fault tolerant system realization. The main contribution of this work can be divided into two groups: primary results and secondary results.

The primary results are: design methodology steps, fault classification, self-checking and fault tolerant structures and etc.

Fault classification: This work supports the design process of CED circuits implemented in FPGAs. A new fault classification was proposed. Briefly, our classification leads to more accurate evaluation of the fault coverage, and it is possible to determine whether the tested circuit satisfies the FS and ST properties. We can also evaluate how many of the faults violate the FS and ST property. The proposed fault classification is used in our experiments. The classification enables us to distinguish which ED code is suitable for the chosen synthesis method with respect to the used fault model.

Self-checking circuit suitable for a fault tolerant system: Previous works has shown that the area overhead depends on the used ED codes. To obtain the minimal area overhead and 100 percent of fault coverage, the appropriate ED code has to be chosen (selected). It may increase the dependability parameters. Single parity is most suitable code for the self-checking circuit, due to its low area overhead and high fault coverage. Other error detection codes lead to higher fault coverage, but the area overhead and the number of possible faults are also higher. A single parity is good trade-off between the area overhead and a fault coverage.

Single parity and parity net grouping: A very efficient application of the on-line BIST design was proposed. Here the circuit outputs are joined together by XOR gates, to form a parity predictor. The parity predictor outputs are compared with the outputs of the original circuit, and thus the appropriate circuit function is checked. The proposed method helps to minimize the parity predictor logic overhead.

Modified duplex system (MDS): CED techniques are not able to increase dependability parameters sufficiently. A new structure based on the DWC-CED technique has been developed. An appropriate ED code was selected to ensure a trade-off between area overhead and fault coverage. The dependability parameters depend on these two criteria.

MDS Implementation with TSC: Our methodology for fault tolerant design is based on SC circuits. It assumes a combinational circuit with up to 16 primary inputs, because simulation time grows by the square of the number of inputs. Therefore there is a need of the compound design architecture. The proposed architecture enables combinational circuits and sequential circuits to be combined in this compound design.

HW emulation of MDS: Each reaction to an input vector change must be calculated in the SW simulation. Each simulation step takes many processor cycles, especially for circuits with many gates. One simulation step is processed, and the time needed for calculation is equal to one system cycle. However the results need to be compared and evaluated concurrently. We therefore decided to use an HW emulator. The HW emulation allow to us calculate the final area of individual parts of MDS structure e.g., parity predictor, checker and area overhead. The HW emulator was programmed with respect to the Atmel FPSLIC FPGA design process.

Proof of MDS optimality: The system was described by a Markov dependability model. This model was used for computing of availability parameters for the SEU fault

model. The results of MCNC [36] and ISCAS [30] benchmarks used in our modified duplex, reconfigurable and on-line testing design method were compared with the result for the standard duplex and TMR systems. It was found that for availability the FS property is more important than the area overhead. When the FS is not 100% achieved, the area overhead is strictly limited by the availability value of the standard duplex system. When this value is surpassed, the availability is inferior to the standard duplex system.

Design methodology: A fault tolerant system design methodology is presented with the aim to obtain results from individual parts of this study. The design methodology enables to use the system in mission-critical applications, where the dependability parameter requirements are very high.

The secondary results are: implementation, modification and other tools.

SW simulator: Because a new fault classification is being presented here, a new fault simulator is needed. This SW simulator has been written in Java programming language.

HW emulator: The new HW emulator was designed to evaluate faults more precisely. The HW emulator was programmed with respect to the Atmel FPSLIC FPGA design process.

Tools that add single parity nets: Some special tools for modifying benchmark circuits had to be used in this work. Utilities allowing circuit modification were programmed. The circuit was described by two-level networks and also by multilevel networks. Utilities enabling us to simulate and calculate fault coverage were implemented.

Tools that add multiple parity nets: A tool enabling modifications of the combinational circuit and selection of the appropriate ED code was programmed. This tool can generate a single event parity predictor, a multiple parity predictor and a Hamming-like code predictor. The BOOM [35] and Espresso [34] minimization tools were used to evaluate the area overhead and thus to select the appropriate ED code.

8.2. Future work

Our future work will deal with several practical case studies (e.g., railway applications). The dependability parameters will be calculated more precisely using assumptions about routing resources impacted by SEUs. We will use a hardware fault emulator based on the ATMEL FPSLIC circuit.

9. References

[1] Nicolaidis, M., Zorian, Y.: ”On-Line Testing for VLSI - A Compendium of Approaches to On-Line Testing for VLSI”, Kluwer Academic Publisher, London 1998, ISBN 0-7923-8132-7.

[2] QuickLogic Corporation.: “Single Event Upsets in FPGAs”, www.quicklogic.com, 2003.

[3] Bellato, M., Bernardi, P., Bortalato, D., Candelaro, A., Ceschia, M., Paccagnella, A., Rebaudego, M., Sonza Reorda, M., Violante, M., Zambolin, P.: “Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA.” Design Automation Event for Electronic System in Europe 2004, pp. 584-589.

[4] Normand, E.: “Single Event Upset at Ground Level,” IEEE Transactions on Nuclear Science, vol. 43, 1996, pp. 2742-2750.

[5] Mitra, S., McCluskey E. J.: "Which Concurrent Error Detection Scheme To Choose?" Proc. International Test Conf. 2000, pp. 985-994.

[6] Mitra, S., McCluskey, E., J.: “Diversity Techniques for Concurrent Error Detection Center for Reliable Computing”, Dept. of Electrical Engineering and Computer Science, Stanford University.

[7] Mitra, S., Saxena, N., R., McCluskey, E., J.: “Common-Mode Failures in Redundant VLSI Systems”, A Survey IEEE Trans. Reliability, 2000.

[8] Bolchini, C., Salice, F., and Sciuto, D.:”Designing Self-Checking FPGAs through Error Detection Codes”, 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT’02), pp. 60, Canada.

[9] Elshafey, K., Hlavicka, J.: “On-Line Detection and Location of Faulty CLBs in FPGA-Based Systems”, IEEE DDECSWorkshop, Brno, Czech Republic, April 17-19, 2002, pp. 183-190.

[10] Xilinx Corp.: “Virtex Series Configuration Architecture User Guide”, XAPP 151 (v1.5), 2000.

[11] Abramovici, M., Stroud, C., Hamiliton, C., Wijesuriya, S., Verma, V.: “Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications”, Proceeding of IEEE International Test Conference, pp. 973-982, 1999.

[12] Lee, H., K., Ha, D. S.: “Atalanta: an Efficient ATPG for Combinational Circuits”, Technical Report, 93-12, Department of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993.

[13] Atmel Corp.: “AT40K Series Configuration”, 2002.

[14] Xilinx Corp.: “Virtex FPGA Series Configuration and Readback”, XAPP 138 (v2.5), Xilinx Corp., 2001.

[15] Xilinx Corp.: “Status and Control Semaphore Registers Using Partial Reconfiguration”, XAPP 153 (v1.0), 1999.

[16] Adamek, J.: “Foundations of coding”, A Wiley-Interscience Publication, John Wiley & Sons, Inc, United States of America 1991, ISBN 0-471-62187-0.

[17] Stroud, Ch., E.: “A Designer’s Guide to Built-In Self-Test”, Kluwer Academic Publisher, London 2002, ISBN 1-4020-7050-0.

[18] Bushnell, M., L., Agrawal, V., D.: “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits”, Kluwer Academic Publisher, London 2000, ISBN 0-7923-7991-8.

[19] Pradhan, D., K.: “Fault-Tolerant Computer System Design”, Prentice Hall PTR, Upper Saddle River, New Jersey 1996, ISBN 0-7923-7991-8.

[20] Paschalis, A., Gizopoulos, D., Gaitanis, N.: “Concurrent Delay Testing in Totally Self-Checking System”, On-Line Testing for VLSI, Kluwer Academic Publisher, London, 1998.

[21] Piestrak, S., J.: “Design of Self-Testing Checkers for m-out-of-n Codes Using Parallel Counters”, On-Line Testing for VLSI, Kluwer Academic Publisher, London, 1998.

[22] Nikolos, D.: “Self-Testing Embedded Two-Rail Checkers”, On-Line Testing for VLSI, Kluwer Academic Publisher, London, 1998.

[23] Brglez, F., Fujiwara, H.: “A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan”, Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985.

[24] Dobias, R., Kubatova, H.:“FPGA Based Design of Raiway's Interlocking Equipment”, In Proceedings of EUROMICRO Symposium on Digital System Design. Piscataway: IEEE, 2004, pp 467-473.

[25] Sterpone, L., Violante, M.: “A design flow for protecting FPGA-based systems against single event upsets “, DFT2005, 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 436 – 444, 2005.

[26] Pradhan, D. K., Fault-Tolerant Computer System Design, Prentice-Hall, Inc., New Jersey, 1996, ISBN 0-13-057887-8.

[27] Drineas, P., Makris, Y.: "Concurrent Fault Detection in Random Combinational Logic”, Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED), 2003, pp. 425-430.

[28] Mohanram, K., Sogomonyan, E. S., Gössel, M., Touba, N. A.: "Synthesis of Low-Cost Parity-Based Partially Self-Checking Circuits", Proceedings of the 9th IEEE International On-Line Testing Symposium 2003, pp. 35.

[29] Graham, P., Caffrey, M., Zimmerman, J., Sundararajan, P., Johnson, E., Patterson, C.:"Consequences and Categories of SRAM FPGA Configuration SEUs", Military and Aerospace Programmable Logic Devices International Conference, Washington DC, MAPLD 2003 Paper C6.

[30] Brglez, F., Bryan, D., Kozminski, D.: “Combinational Profiles of Sequential Benchmark Circuits”, Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989.

[31] Kafka, L.: Design of TSC circuits implemented in FPGA, CTU FEE, 2004.

[32] Bolchini, C., Salice, F., Sciuto, D.: “Fault Analysis for Networks with Concurrent Error Detection”, IEEE Design & Test 15, 4 (Oct. 1998), pp. 66-74, 1998.

[33] Bolchini, C., Salice, F., Sciuto, D.,Zavaglia R.: An Integrated Design Approach for Self-Checking FPGAs, 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03), 2003, pp. 443.

[34] Brayton, R. K. et al. (1984). Logic minimization algorithms for VLSI synthesis, Boston, MA, Kluwer Academic Publishers, 192 pp.

[35] Hlavicka, J. and Fiser, P.: BOOM - a Heuristic Boolean Minimizer. Proc. International Conference on Computer-Aided Design ICCAD 2001, San Jose, California (USA), pp. 439-442.

[36] Yang, S.: “Logic synthesis and optimization benchmarks user guide”, Technical Report 3, Microelectronics Center of North Carolina, 1991.

[37] http://www.synplicity.com

[38] http://www.mentor.com

[39] Brayton, R., K., McMullen, C.T.: “The Decomposition and Factorization of Boolean Expressions”, In Proc. of the IEEE International Symposium on Circuits and Systems, pp. 49-54, 1982.

[40] Muroga, S., Kambayashi, Y., Lai, J., C., Culliney, J., N.: “The Transduction Method – Design of Logic Networks Based on Permissible Functions”, IEEE Trans. on Computers, C-38(10), pp. 1404-1424, 1989.

[41] Stanion, T., Sechen, C.: “Boolean Division and Factorization using Binary Decision Diagrams”, IEEE Trans on CAD, CAD-13(9), pp. 1179-1184, 1994.

[42] Ashenhurst, R., L.: “The Decomposition of Switching Functions”, In Proc. of International Symposium on the Theory of Switching, pp. 74-116, 1957.

[43] Roth, J., P., Karp, R., M.: “Minimization over Boolean Graphs”, IBM Journal of Research and Development, Vol. 6, No. 2, pp. 227-238, 1962.

[44] Bryant, R., E.: “Graph-Based Algorithms for Boolean Function Manipulation”, IEEE Trans. on Computers, C-35(8), pp. 677-691, 1986.

[45] Lai, Y., T., Pedram, M., Vrudhula, S.: “BDD Based Decomposition of Logic for Functions with Applications to FPGA Synthesis”, In Proc. Design Automation Conference, pp. 642-647, 1993.

[46] Sasao, T., Butler, J., T.: “On Bi-Decompositions of Logic Functions”, ACM/IEEE International Workshop on Logic Synthesis, Tahoe City, California, 1997.

[47] Mischenko, A., Steinbach, B., Perkowski, M.: “An Algorithm for Bi-decomposition of Logic Functions”, In Proc. of Design Automation Conference, pp. 103-108, 2001.

[48] Jozwiak, L., Bieganski, S.: “Information Trans-coders in Information-driven Circuit Synthesis”, Proc. 30th Euromicro Symposium on Digital Systems Design (DSD'04), Rennes (FR), 31.8. - 3.9.04, pp. 288-297.

[49] De Micheli, G.: “Synthesis and Optimization of Digital Circuits“. McGraw-Hill, 1994.

[50] Lee, H., K., Ha, D. S.: “HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 9, pp. 1048- 1058, September 1996.

[51] Touba, N. A., McCluskey, E. J.:”Logic Synthesis Techniques for Reduced Area Implementation of Multilevel Circuits with Concurrent Error Detection”, Proc. of ACM/IEEE International Conference on Computer-Aided Design (ICCAD), 1994, pp. 651-654.

[52] Novak, O., Gramatova, E., Ubar, R.: “Handbook of Electronic Testing”, ČVUT, 2005, ISBN 80-01-03318-X.

[53] Leveugle, R. and Cercueil, R.: “High Level Modifications of VHDL Descriptions for On-Line Test or Fault Tolerance”, In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (Dft'01), (October 24 - 26, 2001), IEEE Computer Society, Washington, DC, pp. 84, 2001.

[54] Entrena, L., Lopez, C., Olias, E.: “Automatic insertion of fault-tolerant structures at the RT level”, In Proceedings of Seventh International On-Line Testing Workshop, 9-11 July, pp. 48–50, 2001.

[55] Abramovici, M., Stroud, C., Hamiliton, C., Wijesuriya, S., Verma, V.: “On-Line Testing and Diagnosis of FPGAs with Roving STARs”, In Proceedings of IEEE International On-Line Testing Workshop, Rhodes, Greece, July 5-7, 1999.

[56] Piestrak, J., S.: "Self-Checking Design in Eastern Europe," IEEE Design and Test of Computers, vol. 13, no. 1, pp. 16-25, 1996.

[57] Bernardi, P., Reorda, M. S., Sterpone, L., Violante, M.,: "On the evaluation of SEU sensitiveness in SRAM-based FPGAs," IOLTS2004: IEEE International On-Line Testing Symposium, pp. 115-120, 2004.

[58] Nakahara, K., Kouyama, S., Izumi, T., Ochi, H., Nakamura, Y.: “Autonomous-repair cell for fault tolerant dynamic-reconfigurable devices”, In Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, Monterey, California, USA, February 22 - 24, FPGA '06, pp. 224-224, 2006.

[59] Kastensmidt, de L., G., F., Neuberger, G., Hentschke, F., R., Carro, L., Reis, R.: "Designing Fault-Tolerant Techniques for SRAM-Based FPGAs", IEEE Design and Test of Computers ,vol. 21, no. 6, pp. 552-562, November/December, 2004.

[60] Lima, F., Carro, L., Reis, R.: “Designing Fault Tolerant Systems into SRAM-based FPGAs” In Proceedings of the 40th Design Automation Conference, DAC'03, pp. 650, June 2003.

[61] Yu, S.-Y., McCluskey, E., J.: "Permanent Fault Repair for FPGAs with Limited Redundant Area", In Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, DFT01, pp. 125, 2001.

[62] Mitra, S., Huang, W.-J., Saxena, R., N., Yu, S.-Y., McCluskey, J., E.: “Reconfigurable Architecture for Autonomous Self-Repair”, IEEE Design and Test of Computers, pp. 228-240, May 2004.

[63] Berg, M.: “Fault Tolerance Implementation within SRAM Based FPGA Design Based upon the Increased Level of Single Event Upset Susceptibility”, In

Proceedings of the 12th IEEE International On-Line Testing Symposium, IOLTS'06, pp. 89-91, July 2006.

[64] Wirthlin, M., Johnson, E., Rollins, N., Caffrey, M., Graham, P.: “The Reliability of FPGA Circuit Designs in the Presence of Radiation Induced Configuration Upsets”, In Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM, pp. 133- 142, April 2003.

[65] Touba, N., A., McCluskey E., J.: "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection", IEEE Transactions on Computer-Aided Design, Vol. 16, No. 7, pp. 783-789, Jul. 1997.

10. Publications of the author

10.1. Refereed publications relevant for the thesis

[A.1] Kubalik, P., Kubatova, H.:“Design of Self Checking Circuits Based on FPGA“, In Proceedings of 15th International Conf. on Microelectronics, Cairo, Cairo University, 2003, pp. 378-381. Work portion 50%.

[A.2] Kubalik, P., Kubatova, H.: “On-line Testing for FPGA”, In Proceedings of the Sixth International Scientific Conference Electronic Computers and Informatics, ECI 2004, Technical University Kosice, 2004, pp. 194-199. Work portion 50%.

[A.3] Kubalik, P., Fiser, P., Kubatova, H.: “Minimization of the Hamming Code Generator in Self Checking Circuits”, In Proceedings of the International Workshop on Discrete-Event System Design DESDes'04, Zielona Gora: University of Zielona Gora, 2004, pp. 161-166. Work portion 33%.

[A.4] Kubalik, P., Kubatova, H.: “Parity Codes Used for On-line Testing in FPGA”, In Acta Polytechnica, 2005, vol. 45, no. 6, pp. 53-59. Work portion 50%.

[A.5] Kafka L., Kubalik P., Kubatova H., Novak O.: “Fault Classification for Self-checking Circuits Implemented in FPGA”, In Proceedings of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop DDECS2005, Sopron University of Western Hungary, 2005, pp. 228-231. Work portion 25%.

[A.6] Dobias, R., Kubalik, P., Kubatova, H.: “Dependability Computations for Fault-Tolerant System Based on FPGA”, In Proceedings of the 12th International Conference on Electronics, Circuits and Systems ICECS2005,, IEEE Circuits and Systems Society, 2005, vol. 1, pp. 377-380. Work portion 33%.

[A.7] Kubalik, P., Dobias, R., Kubatova, H.: “Dependability Computation for Fault Tolerant Reconfigurable Duplex System”, In Proceedings of the 2006 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems DDECS2006, CTU Prague 2006, vol. 1, pp. 100-102. Work portion 33%.

[A.8] Kubalik, P., Fiser, P., Kubatova, H.: “Fault Tolerant System Design Method Based on Self-Checking Circuits”, In Proceedings of the 12th IEEE International On-Line Testing Symposium IOLTS 2006, Los Alamitos: IEEE Computer Society, 2006, pp. 185-186. Work portion 33%.

[A.9] Kubalik, P., Dobias, R., Kubatova, H.: “Dependable Design for FPGA based on Duplex System and Reconfiguration”, In Proceedings of 9th Euromicro Conference on Digital System Design, Los Alamitos: IEEE Computer Society, 2006, pp. 139-145. Work portion 33%.

[A.10] Fiser, P., Kubalik, P., Kubatova, H.: ”Output Grouping Method Based on a Similarity of Boolean Functions”, In Proceedings of the 7th International Workshop on Boolean Problems, Technische Universität Bergakademie, Freiberg, 2006, pp. 107-113. Work portion 33%.

[A.11] Kubalik, P., Kubatova, H.: ”Design Methodology for a Highly Reliable System”, In Proceedings of the Seventh International Scientific Conference on Electronic Computers and Informatics ECI 2006, Technical University, Košice, 2006, pp. 274-279. Work portion 50%.

10.2. Unrefereed publications

[A.12] Kubalik, P., Bucek, J.: “FPGA Implementation of USB 1.1 Device Core”, In Proceedings of Workshop 2003, CTU Prague, 2003, pp. 304-305. Work portion 50%.

[A.13] Kubalik, P., Bucek, J.: “FPGA Implementation of USB 1.1 Device Core”, Poster 2003, CTU Prague 2003, pp. IC22. Work portion 50%.

[A.14] Kubalik, P., Kubatova , H.: “Design of Self Checking Circuits Implemented in FPGA”, Postgraduate Study Report 2004, CTU FEE Prague 2004. Work portion 50%.

[A.15] Kubalik, P.: “Fault Tolerant Design Methodology”, POSTER2004, CVUT FEL Praha, 2004, CDROM. Work portion 100%.

[A.16] Kubalik, P., Kubatova, H.: “High Reliable FPGA Based System Design Methodology”, In Proceedings of the Work in Progress Session, DSD 2004, Johannes Kepler University, Linz, 2004, pp. 30-31. Work portion 50%.

[A.17] Kubalik, P., Kubatova, H.: “Design of Self-Testing Circuits Using Parity Codes”, In Proceedings of Workshop 2005, CTU Prague 2005, pp. 214-215. Work portion 50%.

[A.18] Kubalik, P., Kubatova, H.: “Highly Reliable Design Based on TSC Circuits”, In Proceedings of Pocitacove architektury & diagnostika, CTU FEE Prague 2005, pp. 101-106. Work portion 50%.

[A.19] Kubalik, P., Kubatova, H.: “Reconfigurable Duplex System Increasing Fault Tolerance for Circuits Based on FPGAs”, In Proceedings of the Work in Progress Session, DSD2005, Johannes Kepler University, Linz, 2005, pp. 13-14. Work portion 50%.

10.3. Citations

No citations or responses are known to the author.