ON-LINE TESTING AND RECOVERY OF FPGA-BASED SYSTEMS

Uroš Legat

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia, May, 2012

Evaluation Board:

Prof. dr. Gorazd Kandus, Chairman, Institut "Jožef Stefan", Jamova 39, 1000 Ljubljana

Prof. dr. Andrej Žemva, Member, Fakulteta za elektrotehniko, Tržaška 25, 1000 Ljubljana

Doc. dr. Gregor Papa, Member, Institut "Jožef Stefan", Jamova 39, 1000 Ljubljana

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Uroš Legat

ON-LINE TESTING AND RECOVERY OF FPGA-BASED SYSTEMS

Doctoral Dissertation

SPROTNO TESTIRANJE IN POPRAVLJANJE SISTEMOV OSNOVANIH NA VEZJIH FPGA

Doktorska disertacija

Supervisor: Prof. dr. Franc Novak

Ljubljana, Slovenia, May, 2012

Index

1 Introduction ... 1

2 Aims and Hypothesis ... 5

3 Preliminaries ... 7

3.1 Field-Programmable Gate Array ... 7

3.1.1 Basic Architecture of a SRAM-based FPGA ... 8

3.1.1.1 Configurable logic block (CLB) ... 9

3.1.1.2 Input output block (IOB) ... 10

3.1.1.3 Internal memory (block RAM) ... 10

3.1.1.4 Routing Resources ... 11

3.1.2 Organization of FPGA configuration memory ... 11

3.1.3 Available configuration interfaces ... 12

3.1.4 Partial runtime reconfiguration of FPGA ... 14

3.1.4.1 Types of partial reconfiguration ... 14

3.1.4.2 Implementation flows for a partial FPGA reconfiguration ... 14

3.2 Radiation effects in integrated circuits ... 15

3.2.1 Soft errors in integrated circuits ... 17

3.2.2 Soft errors in SRAM-based FPGAS ... 18

3.2.3 The reliability of Xilinx FPGA devices ... 20

4 On-line testing and fault-tolerance techniques ... 23

4.1 On-line testing techniques ... 23

4.2 Error-mitigation techniques ... 26

4.3 Error-recovery techniques in SRAM-based FPGAs ... 30

5 Verification of on-line testing and recovery solutions ... 33

5.1 Physical fault injection ... 33

5.2 Fault injection with HDL simulation ... 33

5.3 Fault injection with emulation on FPGA ... 34

5.4 Our fault-emulation approach ... 36

6 On-line testing techniques ... 39

6.1 Advanced Encryption Standard ... 39

6.2 State of the art ... 40

6.3 32-bit AES architecture ... 42

6.4 The AES algorithm with on-line error detection ... 44

6.4.1 Fault masking in the CT-box ... 47

6.4.2 Hardware implementation ... 48

6.5 BIST method for the AES algorithm with error detection ... 49

6.5.1 Hardware implementation results of the BIST ... 52

6.6 Our fault-emulation tool 52
6.6.1 Determining the fault sources inside the DUT 53
6.6.2 Fault-injection flow using the HWICAP core on the Virtex 4 FPGA 54
6.6.2.1 Configurable logic block 54
6.6.2.2 Block RAM 56
6.7 Fault-emulation results 56
6.7.1 BRAM faults 56
6.7.2 LUT faults 57

7 Error-recovery techniques 59
7.1 Self-recovery of embedded multi-processor systems on the FPGA 59
7.1.1 Required hardware platform 59
7.1.2 Error-recovery technique 60
7.1.2.1 Test scheduling 60
7.1.2.2 Configuration check and recovery 61
7.1.2.3 Error-recovery algorithm 61
7.1.3 Practical application of the error-recovery method 62
7.1.3.1 SEU fault-emulation experiment 63
7.1.3.2 Fault-emulation results 64
7.1.3.3 Reliability estimation 64
7.2 Hardware mechanism for the self-recovery of FPGA systems 66
7.2.1 Configuration check and recovery 66
7.2.2 Implementation of error-recovery mechanism 66
7.2.3 Operation of the internal recovery mechanism 67
7.2.4 Error-recovery-time comparison 68
7.2.5 Hardware-implementation comparison 69
7.2.6 Implementation of the internal error-recovery mechanism in TMR 69
7.2.7 Self-recovery architectures 71
7.2.8 Fault-emulation experiment 72
7.2.9 Reliability estimation 74

8 Conclusions 77

9 Acknowledgements 79

10 References 80

11 Bibliography 87

Abstract

SRAM-based FPGAs have become an attractive solution for many applications where a short development time, low-cost for low-production volumes, and in-the-field-programming ability are important issues. The flexibility of SRAM-based FPGAs comes from the adoption of a configuration memory that defines the operations of the circuit that the FPGA implements. It is therefore fundamental that the content of the configuration memory preserves the correct values during the FPGA operation. The main concern for the reliability and dependability of SRAM-based FPGAs are radiation-induced soft-errors that corrupt the configuration memory (produce bit-flips). These errors often occur in the space environment; however, because of increasing integration density they are also not uncommon at sea-level.

Different fault-tolerance techniques are being developed to increase the reliability and dependability of applications on FPGAs. These techniques function concurrently (on-line) with the system to monitor its operation. On-line testing techniques detect the errors in the system, error mitigation techniques are able to enhance the system to work despite faults, and error-recovery techniques recover the faults from the system. The goals of fault-tolerance techniques are to minimize the hardware, timing, and power overhead, and maximize the reliability of the system. This dissertation presents our advances in fault-tolerance techniques.

We have developed an on-line testing technique for an advanced encryption standard (AES) implemented on FPGA. This 32-bit AES core is the smallest reported AES core with error detection. The error detection is implemented in all the AES processes: encryption, decryption, and key schedule. Besides the on-line error-detection mode our AES core can also be tested in an efficient off-line self-test (BIST) mode. This novel smart BIST solution generates the random test vectors by performing the AES processes in a loop and uses the existing circuit of on-line error detection to analyze the outputs.

Additionally, we improved the existing error-recovery techniques. The first error-recovery technique is applicable for multiprocessor systems on FPGAs. We developed a software algorithm that can recover soft-errors from the FPGA configuration memory. The recovery algorithm for a single processor is already reported in the literature. However, the advanced feature of this algorithm is that it can adapt itself to another processor if the current testing processor is corrupted.

The second error-recovery technique is a hardware based error-recovery mechanism. This is the smallest and fastest controller that checks the configuration memory of the FPGA device and recovers the potential soft-errors. The error-recovery mechanism is small enough to be included in almost any FPGA design without the need to replace the FPGA device. We included the error-recovery mechanism in different self-recovery architectures with different levels of reliability.

All our developed fault-tolerance techniques were validated by fault-injection experiments. For this purpose we developed a fault-injection tool that automatically injects faults into the FPGA using a partial runtime reconfiguration.

Povzetek

Vezja FPGA osnovana na statičnem pomnilniku so postala najprimernejša platforma za vrsto sodobnih aplikacij, kjer so pomembni kratek čas razvoja, manjša cena pri majhnih količinah in zmožnost reprogramiranja vezja med delovanjem. Fleksibilnost vezij FPGA izhaja iz možnosti spreminjanja in prilagajanja konfiguracijskega spomina, ki določa funkcijo vezja. Da se funkcija vezja med delovanjem ohrani, moramo zaščititi konfiguracijski spomin pred zunanjimi vplivi. Glavna skrb pri zagotavljanju zanesljivosti vezij FPGA je sevanje. Delci z veliko energijo, ki naključno zadenejo vezje, lahko povzročijo spremembo konfiguracijskega spomina (povzročijo preklop bitov). Takšne napake imenujemo tudi napake SEU (ang. single-event upset). Napake SEU so zelo pogoste v vesoljskih aplikacijah, vendar se te napake zaradi vedno večje tranzistorske gostote modernih vezij pojavljajo tudi na zemeljski površini.

Obstaja veliko različnih testnih metod, ki povečajo zanesljivost aplikacij na vezjih FPGA. Te metode delujejo vzporedno z delovanjem aplikacije. Metode sprotnega testiranja skoraj v trenutku zaznajo napake v aplikaciji in omogočajo metodam sprotnega popravljanja napak, da le te odpravijo. Imamo pa tudi metode izogibanja napakam, ki omogočajo pravilno delovanje vezja kljub prisotnim napakam. Lastnosti dobrih testnih metod so, da pri čim manjši redundanci v smislu vezja, časa, ali porabe energije maksimalno povečajo zanesljivost vezja. V doktorski disertaciji predstavljamo naše rezultate na področju razvoja metod za sprotno testiranje in popravljanje napak SEU.

Razvili smo metodo za sprotno testiranje šifrirnega algoritma AES (ang. advanced encryption standard). To je najmanjše vezje šifrirnega algoritma AES z vgrajenim sistemom za sprotno odkrivanje napak. Odkrivanje napak smo implementirali za vse operacije algoritma: enkripcijo, dekripcijo in razširjanje ključa. Poleg sprotnega testiranja pa smo razvili tudi vgrajeni samodejni test BIST (ang. built-in self-test). Ta nova testna metoda izvaja operacije algoritma v zanki in uporabi že vgrajen sistem za odkrivanje napak za analizo pravilnega delovanja algoritma.

Izboljšali smo tudi metode za sprotno popravljanje napak. Razvili smo metodo za sprotno popravljanje napak na vezjih FPGA z več vgrajenimi procesorji. Izdelali smo program za odpravljanje napak v konfiguracijskem spominu FPGA. Podoben program za odpravljanje napak za delovanje na enem procesorju so razvili že drugi, mi pa smo dodali možnost, da ob odpovedi tega procesorja napako lahko odpravi drug delujoč procesor.

Druga metoda je doslej najmanjši in najhitrejši strojno implementirani mehanizem za popravljanje napak. Mehanizem sprotno pregleduje konfiguracijski spomin vezja FPGA in odpravlja morebitne napake. Ta mehanizem je dovolj majhen, da ga lahko dodamo skoraj vsaki aplikaciji na vezju FPGA, ne da bi morali vezje zamenjati z večjim. Za ta mehanizem smo predlagali različne možnosti uporabe, ki ob različni redundanci ponujajo različno stopnjo zanesljivosti.

Vse razvite testne metode smo preverili z eksperimenti pri katerih smo v sistem vnašali napake in opazovali odzivnost metod na le te. Posledično smo razvili tudi svoje inovativno orodje za avtomatsko vnašanje napak v vezja FPGA.

Abbreviations

ADC	=	analog-to-digital converter
AES	=	advanced encryption standard
ASIC	=	application-specific integrated circuit
ASSP	=	application-specific standard product
ATE	=	automated test equipment
BIST	=	built-in self-test
BRAM	=	block-random-access memory
CLB	=	configurable logic block
COTS	=	commercial off-the-shelf
CRC	=	cyclic redundancy check
DCM	=	digital clock manager
DSP	=	digital signal processor
DUT	=	device under test
ECC	=	error correcting code
EDC	=	error detecting code
FF	=	flip-flop
FIT	=	failures-in-time
FIFO	=	first-in first-out
FPGA	=	field-programmable gate array
FSL	=	fast simplex link
FSM	=	finite state machine
HDL	=	hardware description language
ICAP	=	internal configuration access port
IOB	=	input-output block
JTAG	=	IEEE 1149.1 standard test access port (or boundary-scan)
LET	=	Linear Energy Transfer
LUT	=	lookup-table
MBU	=	multiple-bit upset
MDM	=	MicroBlaze debug module
MPSOC	=	multiprocessor system on chip
MTBF	=	mean time between failure
MUX	=	multiplexer
NIST	=	National Institute of Standards and Technology
NOC	=	network on chip
OED	=	on-line error detection
PIP	=	programmable interconnect points
PRR	=	partial runtime reconfiguration
RAM	=	random-access memory
ROM	=	read-only memory
RS-232	=	serial communication standard
SDK	=	software development kit
SEC-DED	=	single-error correction double-error detection
SEE	=	single-event effect

SEFI	=	single-event functional interrupt
SET	=	single-event transient
SEU	=	single-event upset
SOC	=	system on chip
SOPC	=	system on programmable chip
SRAM	=	static random-access memory
SRL	=	shift register look-up table
TMR	=	triple modular redundancy
VHDL	=	VHSIC hardware description language
VHSIC	=	very-high-speed integrated circuit
VLSI	=	very-large-scale integration
XOR	=	exclusive or logic operation

1 Introduction

Field-programmable gate-array or FPGA devices are becoming the most suitable platform for implementing modern electronic systems due to their high level of reconfigurability, low cost and wide availability. FPGA devices are programmable logic circuits that can be programmed or reprogramed (configured or reconfigured) with almost any circuit or system. They are nowadays being used for coprocessors in high-performance systems to speed-up difficult tasks, for various embedded systems, systems on a chip, networks on a chip, or as a platform to design-circuit prototypes, etc. The FPGA devices are also increasingly being used in critical systems like space exploration missions, avionics, security systems, banking systems, secure servers, etc. These systems require the highest level of reliability, which is increased by including various testing techniques and error-recovery mechanisms.

Since the integration density of the VLSI chips is increasing; ensuring the reliability of the circuits is becoming much more challenging. The flaws in fabrication processes are usually detected by factory tests but the reliability of the circuits in operation is crucial. There are many negative external effects on integrated circuits, like power fluctuations, static charges, radiation, etc. These effects can cause hard or soft errors in the circuits. Soft-errors cause the wrong behavior of the circuit, which can be rectified by resetting or reconfiguring the system, while hard errors permanently damage the circuit.

The most common reliability concerns in modern systems are soft errors, which are usually caused by cosmic or artificial radiation. A high-energy particle can strike the surface of an integrated circuit, transferring enough energy to provoke a fault in the system. This effect is called a single-event effect (SEE). The SEE can cause a transient fault in a combinational logic of the circuit called a single-event transient (SET), or change the contents of a memory cell in the sequential part of the circuit or device memory, called a single-event upset (SEU). SRAM-based FPGAs are especially vulnerable to a SEU, hence it can change the configuration memory of the FPGA and consequently alter its basic functionality.

The radiation is very high in space where integrated circuits can experience several such faults per day. These faults cannot be prevented by physical radiation shields and the designers for space applications are using very expensive, radiation-hardened devices to cope with these effects. However, there is also a strong consideration to use commercial off-the-shelf (COTS) devices like SRAM-based FPGAs in space systems to minimize costs and development time.

Because of the atmospheric radiation shield the terrestrial applications are less exposed to such events. However, numerous experiments confirmed that the levels of radiation in the atmosphere and on the earth's surface also cause SEUs. Future high-density devices with low supply voltages will be even more vulnerable to the cosmic radiation produced by the sun. The use of on-line testing and fault-tolerance techniques will become indispensable.

The dissertation addresses problems and methods how to test, mitigate, and recover the above faults on an FPGA platform. On-line testing techniques detect the errors during the normal operation of the system. The on-line detection of errors contributes to a short

fault-detection latency, which is very important in order to prevent the fault from propagating further through the system. On the other hand, error-mitigation techniques can tolerate faults that occur during the system's operation. If a fault occurs in one part of the circuit, then a redundant part of the circuit is used to provide the correct and uninterrupted operation of the system. When a fault is detected inside a system it can be repaired (recovered) by error-recovery techniques.

The development of on-line testing and fault-tolerance techniques is strongly associated with the target device, and it requires a detailed analysis of the effects the upsets have on the device. The designer of an on-line testing for the particular target device has to find the best trade-off between hardware overhead and performance penalties, and the desired system reliability. To design on-line testing techniques for FPGA devices the techniques for standard integrated circuits are modified to cope with the distinct effects of a SEU on the programmable logic.

The most common techniques of on-line testing and mitigation are associated with circuit redundancy, such as duplication and comparison or triple modular redundancy (TMR). These techniques duplicate or triplicate the original circuit's design into identical modules and observe their outputs to detect the faulty behavior of the circuit. The design of the TMR technique in FPGA devices has to be carefully planned. The copies of the design modules have to be placed separately in the configuration memory and the internal signals have to be carefully routed. Following these precautions decreases the probability that a fault would affect multiple copies of design modules and produce failure. These methods are widely used in mission-critical systems, but they introduce a very high hardware overhead and increase power consumption.

Other circuit-redundancy techniques are based on codes. Error-detecting codes (EDC) to test the circuit or error-correcting codes (ECC) to mitigate the errors. The EDCs are mostly used to test memories or registers in the systems. The registers or memory words are extended by a number of check bits. The values of check bits are calculated from the rest of the data at the input (encoder) and the data are checked using the check bits at the output (checker). In FPGAs ECCs can be used to correct the contents of an internal memory (block RAM), sequential logic registers (flip-flops) or configuration memory.

Some on-line testing techniques use time or temporal redundancy. The basic principle of these techniques is to observe the output of the circuit at different times and compare the result. These techniques are used to detect a transient effect in a circuit (SET) and are not able to detect or mitigate permanent effects like a SEU.

The on-line error-recovery techniques of FPGA devices are used to recover errors from the configuration memories. The error recovery can be done simply by reconfiguring the whole device at a scheduled time without checking the system for errors; such an approach is also known as configuration "scrubbing". Other, more advanced, recovery methods employ a special error-recovery mechanism. The mechanism is independent of the user design. It uses the FPGA configuration interface to check the FPGA's configuration memory and clean upsets that occur. The device can be configured externally or internally, therefore we have internal or external recovery mechanisms. The external mechanisms are separate integrated circuits (microprocessor or FPGA) that are also subjected to radiation and vulnerable to SEE, hence radiation-hardened mechanisms are used, which increases the cost of the system. The internal error-recovery mechanisms are implemented in the FPGA logic at nearly no additional cost. However, the configuration memory of the mechanism itself can be upset, therefore it has to be as small as possible that the probability of SEU occurring in the mechanism is low. To additionally increase the reliability of the system the internal error-recovery mechanism should be protected by a fault-mitigation technique.

The dissertation is organized as follows. In chapter 2 initial aims and hypothesis are defined. The basic concepts required for understanding the dissertation are explained in chapter 3. The first part of the chapter addresses FPGA devices: FPGA internal architecture and organization of configuration memory are described. The second part of the chapter discusses radiation effects in integrated circuits and classifies soft-errors. Chapter 4 summarizes the state of the art techniques of on-line testing, mitigation, and error recovery. In Chapter 5 fault-injection techniques for integrated circuits are presented. Fault-injection experiments are important for evaluating the feasibility of fault-tolerance techniques. For the purpose of checking the reliability and fault coverage of our systems, we developed our novel tool for automatic fault injection using hardware emulation on FPGA.

The rest of the dissertation addresses the developed on-line testing and error-recovery solutions. On-line error detection of a small 32-bit AES core is introduced in Chapter 6. In Chapter 7 two approaches for error recovery are introduced. The first approach is a software-based recovery algorithm. This algorithm was extended from single-processor systems to multiprocessor systems. The latter is a small hardware-based recovery mechanism that can be added to any system on the FPGA. The mechanism has also been implemented in triple modular redundancy to increase the reliability.

2 Aims and Hypothesis

The purpose of the dissertation is to improve conventional on-line test methods for the systems on FPGA. Our aim is to develop methods that will reduce the hardware cost, shorten fault-detection latency, and increase the reliability of the systems.

Our first goal is related to the reliability of the hardware implementation of the Advanced Encryption Standard (AES). The AES is an encryption algorithm that is widely used in secure storage and communication. Different On-line Error Detection schemes for hardware implementations of the AES have recently been investigated. These schemes detect the erroneous operation of the algorithm and offer different levels of protection against fault-based side-channel attacks. The employed schemes use either temporal, functional, or information redundancy. Existing approaches focus on the individual AES processes (i.e., encryption, decryption, and key expansion). Besides, they are mostly built for complex hardware implementations that are not suitable for small embedded systems.

- We intend to develop an on-line test solution of the complete AES. In addition, the modular structure of the AES offers the means for the implementation of BIST. We shall organize individual parts into an efficient BIST with a low hardware overhead. The resulting AES core will be compact enough to fit into the smallest FPGAs and still offer acceptable throughput. We expect to achieve fault coverage that is better or at least comparable to the existing solutions.

Another goal is to improve the internal FPGA error recovery. The recovery technique recovers soft errors from the configuration memory of the FPGA. This method requires a recovery controller that checks the configuration memory and repairs the fault when it occurs. The controller is usually an embedded microprocessor which imposes a considerable hardware overhead. The current best solution is made by Xilinx, which uses a very small embedded microprocessor PicoBlaze. In this regard we shall pursue the goal of reducing the hardware overhead and reducing the fault-detection latency of the implemented error recovery.

- Our approach will employ the final state machine to control the process of error recovery. We expect that this solution will occupy fewer hardware resources and will have a shorter fault-detection latency.
- Our second proposed approach for configuration recovery will target multiprocessor systems on the FPGA. This approach will not be the optimal solution as regards fault-detection latency but will offer on-line checking of the multiprocessor system without many additional hardware resources. A novel feature of this approach is in an improved multiprocessor test task scheduling: in the case of the failure of the current testing processor, another processor takes over the testing and recovers the system.

Fault-detection and recovery techniques are normally evaluated using fault injection. For this purpose, two basic types of fault-injection techniques exist in practice: (software)

simulation and (hardware) emulation. Since none of the existing fault-injection tools included automatic identification of the employed FPGA resources, we decided to set up our own fault-emulation environment.

- In contrast to other state-of-the-art tools our tool should be able to determine the resources used by the device under test and generate the list of fault sources automatically. The faults will be injected into the FPGA by changing the original configuration memory through a partial runtime reconfiguration. The tool will be able to inject the fault, run and stop the device under test and determine the fault-coverage of the fault-tolerance scheme. In this way, a substantial speedup is expected.

3 Preliminaries

In this chapter we briefly refer to the main features of the Field-Programmable Gate Arrays that are necessary for the understanding of the developed error-detection and recovery techniques. The basic architecture of the FPGA circuit is explained along with the organization of its configuration memory. The partial runtime reconfiguration procedure and design flows are described.

In the second part of the chapter the reliability issues are addressed. The integrated circuits, especially the FPGAs, are susceptible to radiation-induced soft errors. Common fault models in integrated circuits and the reliability evaluations of the particular FPGA devices are presented.

3.1 Field-Programmable Gate Array

Electronic systems are nowadays designed in application-specific integrated-circuit (ASIC) or in field-programmable gate arrays (FPGA).

The ASIC system has a fixed factory-programmed structure that cannot be changed during the lifetime of the device (Smith, 2001). The specific purpose design enables high performance and power optimization. The systems are designed in hardware-description language (HDL) and produced by a semiconductor factory. The design process of ASIC is extremely expensive and its cost is high for small volume production. Modern ASICs include one or more microprocessors, memories, and other peripherals in a single chip for example to run a mobile phone, or other embedded system device.

FPGA is a multi-purpose programmable integrated circuit that can be configured (programmed) by the user (Maxfield, 2008). The system on the FPGA is designed in HDL and it can be easily synthesized and transferred to the chip, which allows quick and easy development. The FPGA circuits can be reprogramed many times, which makes them a perfect prototyping platform and also enables on-site hardware updates and repairs. Some FPGA devices can also be partially re-configured during runtime. The so-called partial runtime reconfiguration (PRR) allows changing a part of the circuit while the rest of the circuit is still working without interruption. This capability enables the design of self-reconfigurable systems, which can use the same hardware resources to perform different functions. This saves hardware resources and reduces the power consumption. PRR can also be used for injecting or repairing faults in the system during its operation.

The basic disadvantages of FPGAs against ASICs are lower clock rates (lower performance) and lower energy efficiency.

FPGA devices are produced by a number of semiconductor companies: Xilinx, Altera, Actel, Lattice, QuickLogic and Atmel. The types of FPGAs according to technology are:

- SRAM - based on static random-access memory (RAM). Volatile, requires external boot device (external non-volatile memory or external computer). The most versatile and the most commercially available type of FPGA. Enables reconfiguration and partial runtime reconfiguration.
- Antifuse - non-volatile, low power, radiation-tolerant and reliable. Is configured only

once. Highly expensive and designed for space applications.

- Flash – non-volatile. Flash-based FPGAs can be reconfigured by the end user. In comparison with the SRAM-based FPGA the flash-based FPGAs are more reliable, consume less power, but have lower density and lower performance.

In the rest of this section we focus on the structure and operation of the Xilinx SRAM-based FPGA with the emphasis on the Virtex 4 (Xilinx, 2008) and Virtex 5 (Xilinx, 2010a) FPGA families, which were used in the experiments described in the thesis. The majority of described features are common for all SRAM-based FPGAs, but some features are specific to Xilinx FPGAs.

3.1.1 Basic Architecture of a SRAM-based FPGA

The FPGA architecture consists of a network of configurable elements that enable the functionality of the FPGA (Xilinx, 2008; Xilinx, 2010a). There are three basic types of configurable elements:

- configurable logic blocks (CLBs),
- input/output blocks (IOBs),
- and block RAMs,

that are placed in separate columns and connected with internal routing resources. CLBs perform user-specified logic functions and sequential operations. The IOBs provide a programmable interface between the internal array of CLBs and the device's external package pins. Block RAMs provide an easily accessible internal memory. Besides the basic configurable elements, some FPGAs also include, clock management (DCM), digital signal processors (DSP), analog-to-digital converters (ADC) internal configuration interfaces (ICAP), etc. The basic architecture of a SRAM-based FPGA is depicted in Figure 1.

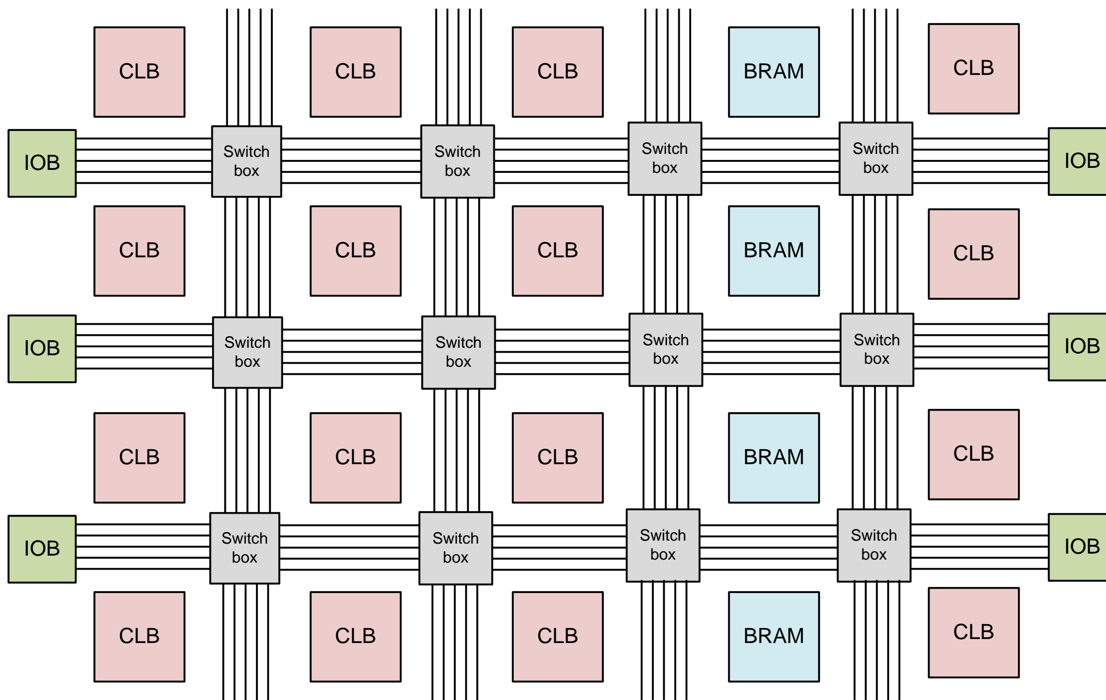


Figure 1: Basic Architecture of SRAM-based FPGA.

3.1.1.1 Configurable logic block (CLB)

CLBs are the main logic resource for implementing sequential as well as combinatorial circuits. The exact numbers and features vary from device to device. Every CLB consists of a certain number of slices. A slice is composed of a look-up table (LUT) with 4 or 6 inputs, some selection circuitry (MUX, inverters, buffers, etc.), and flip-flops. The LUT element is flexible and can be configured as a combinatorial logic, shift register, or RAM. Figure 2 contains the basic components of a slice:

- G [0-3] are inputs of LUT.
- INIT [0-15] are LUT content or initial values of LUT shift registers or LUT RAM.
- Flip-flop (FF) is reset by global set/rest (SR), which can be inverted SR_INV.
- Carry and control logic is composed of several multiplexers, which determine the internal routing between inputs and outputs.
- CIN and COUT are carry chain inputs and outputs.

Each CLB element is connected to a switch matrix to access to the general routing matrix. More architectural details can be found in the data sheet of aparticular FPGA device.

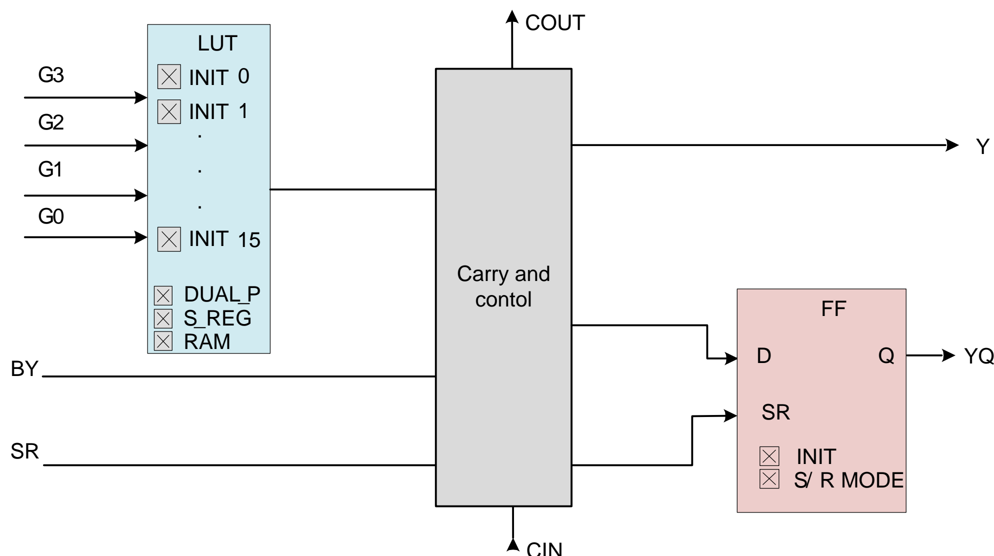


Figure 2: Basic structure of a slice in the configurable logic block.

Virtex 4 FPGA device: Virtex 4 CLB contains four interconnected slices, as shown in Figure 3. These slices are grouped in pairs. Each pair is organized as a column, and each pair in a column has an independent carry chain. The slices are connected to the general FPGA routing through the switch matrix. Virtex 4 slice has two four input LUTs and two D-type flip-flops.

Virtex 5 FPGA device: A CLB element in Virtex 5 FPGA contains a pair of slices. These two slices do not have direct connections to each other, and each slice is organized as a column as depicted in Figure 4. Each slice in a column has an independent carry chain. For each CLB, slices in the bottom of the CLB are labeled as SLICE(0), and slices in the top of the CLB are labeled as SLICE(1). The slices are connected to the general FPGA routing through the switch matrix. The Virtex 5 slice has four LUTs with six inputs and four D-type flip-flops.

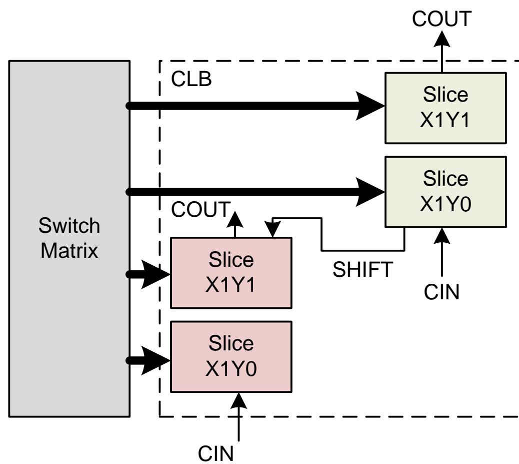


Figure 3: Structure of Virtex 4 CLB

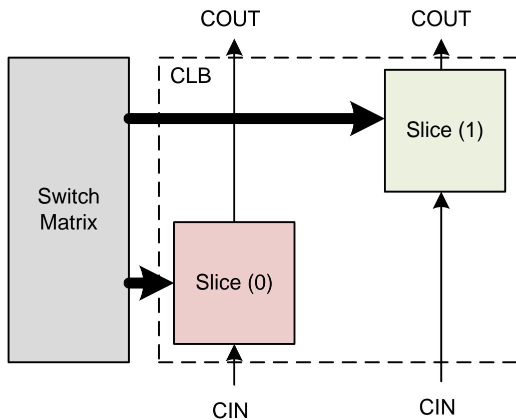


Figure 4: Structure of Virtex 5 CLB

3.1.1.2 Input output block (IOB)

User-configurable IOBs provide the interface between the external package pins and the internal logic. Each IOB controls one package pin, and can be configured for input, output, or bidirectional signals.
The IOB includes a direct input and a tri-statable output. The configuration options on the IOB include input inversion, output inversion, tri-state control inversion, a controllable slew-rate output, and a programmable delay to eliminate the input hold time when the input buffer directly sources a flip-flop. A pull-up or pull-down resistor can be activated for either inputs or outputs.

3.1.1.3 Internal memory (block RAM)

Block RAM is a configurable internal memory element that is easily and quickly accessible. A block RAM can be configured in different widths and sizes. Multiple RAM blocks can be connected into one larger memory. Besides as a single port, the block RAM can be configured in a dual-port mode. The two ports are symmetrical and totally independent, sharing only the stored data. The block RAM in Virtex 4 and Virtex 5 FPGA also has an optional error-correcting (ECC) mechanism. Another important feature of the memory is that it can be defined in the configuration bitstream and altered by dynamic partial reconfiguration. Input and output signals of the Virtex 4 and Virtex 5 FPGA dual-port block RAM are shown in Figure 5. The size of the Virtex 4 RAM block is 18 kbit

while Virtex 5 can select between 18 Kbit and 36 kbit RAMs.

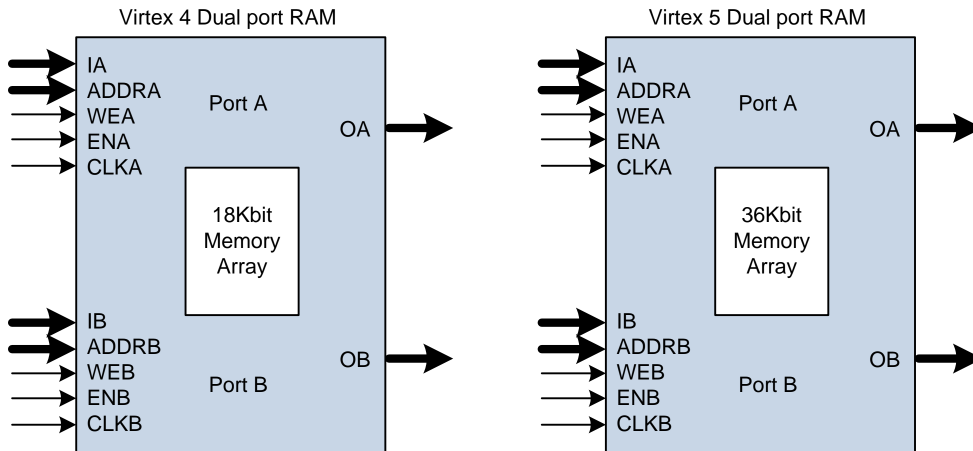


Figure 5: Connection signals of a block RAM in dual-port mode

3.1.1.4 Routing Resources

Routing resources are used to connect the CLBs, IOBs and other resources inside the FPGA device. The routing consists of switch boxes and programmable interconnect points (PIP). The switch box is used to connect the main wires. It is located at the intersections of the horizontal and vertical main wires and is comprised of a matrix of switches. The PIPs are used to connect a particular resource to the main wire. An example of routing is shown in Figure 6. An IOB is connected to the input CLB 1 through a PIP and the output of the CLB 1 is connected to the input of CLB 2 through a PIP and to the input of CLB 3 through a Switch box and a PIP.

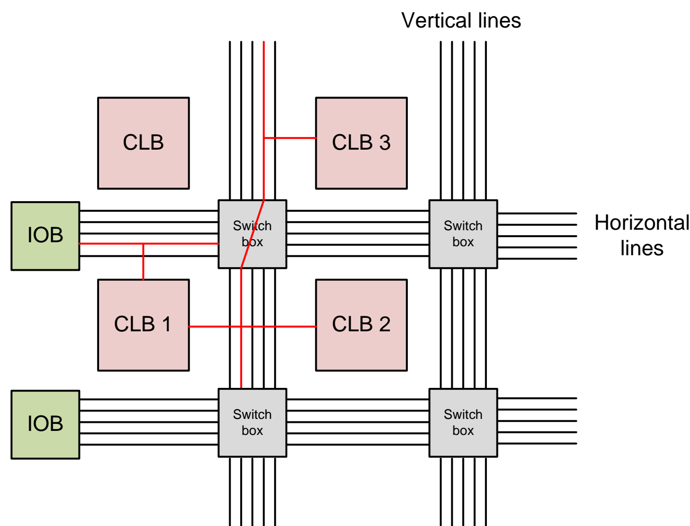


Figure 6: Internal routing example

3.1.2 Organization of FPGA configuration memory

The functionality of a SRAM FPGA is determined by the state of its configuration memory (SRAM cells) (Xilinx, 2009 and Xilinx, 2010b). The structure of the FPGA configuration is depicted in Figure 7. In the Xilinx FPGA the configuration memory is organized in a network of configuration frames that are laid out on a device according to their frame address. A configuration frame is the smallest reconfigurable part of the

FPGA. The size of a frame in Virtex 4, Virtex 5 FPGA is 41 words of 32-bits. The frame address is comprised of a *major frame address, block type, top/bottom bit, row,* and *minor address*. The block type has two types that are relevant for the user; the block types have a separate major address, see Figure 7. First is block type 0, which contains interconnect and configuration of configurable elements CLB, DCM ,DSP, IOBs, and block RAM. The second is block type 1, which holds the block RAM content.

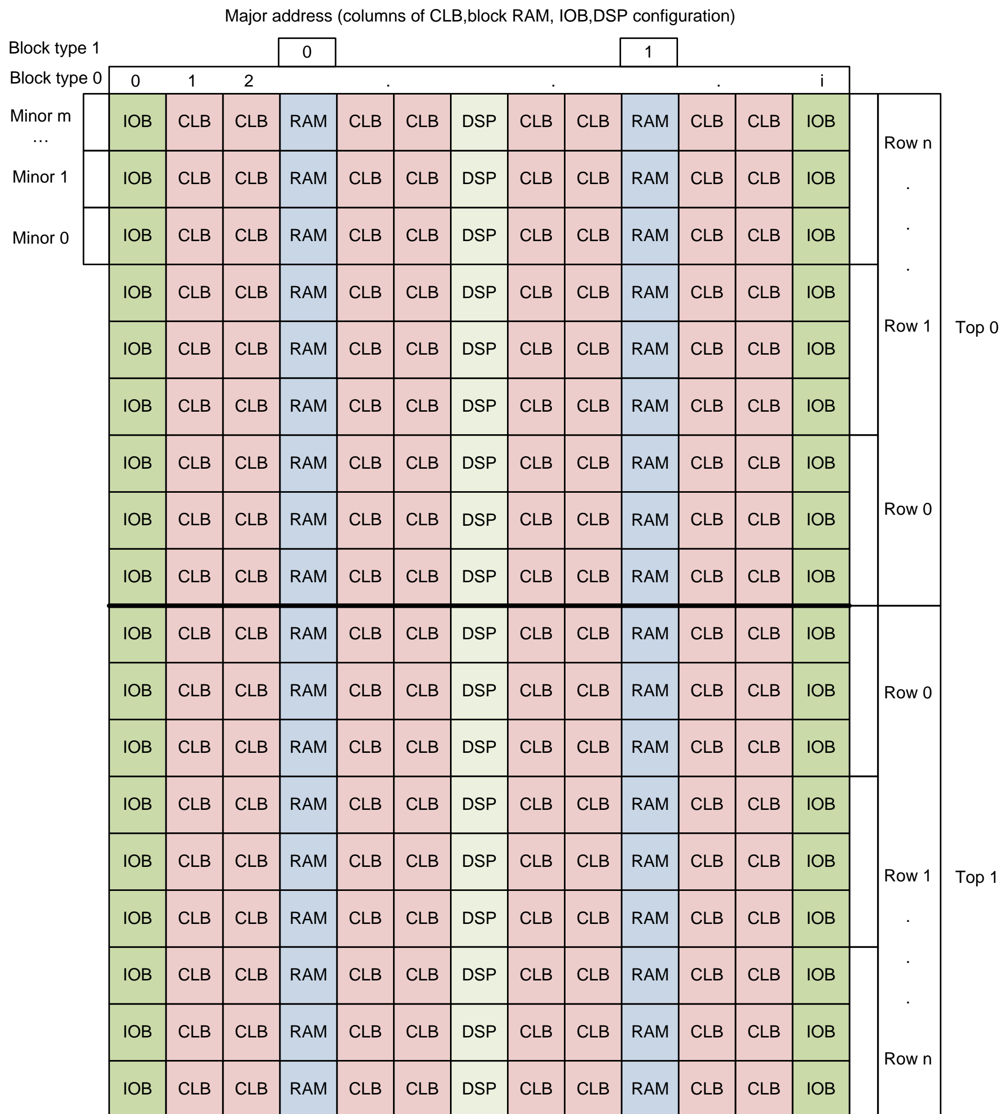


Figure 7: Configuration organization of configuration memory according to frame address

3.1.3 Available configuration interfaces

The configuration data is loaded into the device during power-up through different kinds of configuration interfaces, also known as configuration ports: Parallel port SelectMAP, serial joint-test-action-group (JTAG) and internal-configuration access-port (ICAP). The I/O structure of the configuration interfaces is shown in Figure 8 (Xilinx, 2009 and Xilinx, 2010b).

The Virtex 4 and 5 FPGAs are fully compliant to the IEEE 1149.1 standard, also called JTAG or Boundary-Scan. Boundary-scan is normally used for board-level and device testing. In addition to testing, Boundary-Scan offers the flexibility for a device to have its

own set of user-defined instructions. The FPGA uses a set of user-defined configuration instructions to perform serial configuration of the device. As shown in Figure 8a, JTAG has three inputs and one output:

- TMS is a control input to the TAP controller state machine,
- TDI is a serial data input to the boundary register chain,
- TCK or test clock is a clock signal input of the interface,
- TDO is a serial output from the boundary register.

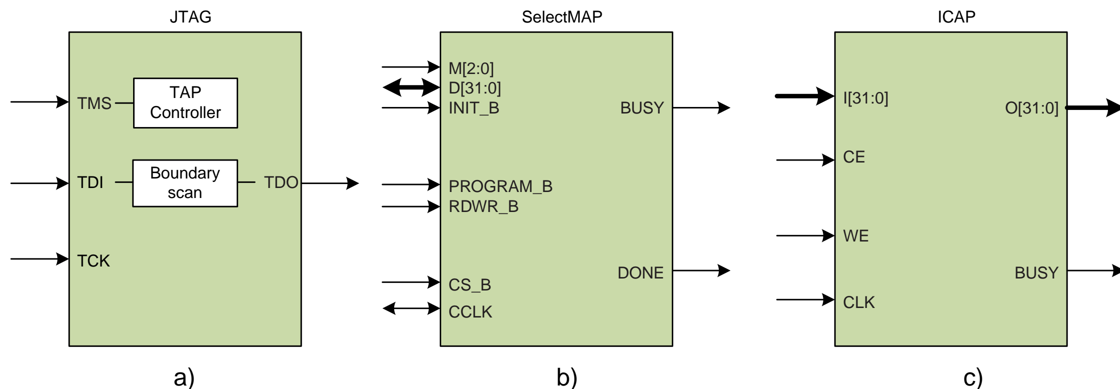


Figure 8: Configuration interfaces of Xilinx FPGAs

SelectMAP configuration interface provides an 8-bit, 16-bit, or 32-bit bidirectional data-bus interface to the configuration logic that can be used for both configuration and readback. The selectMAP can be interfaced as shown in Figure 8b. This interface supports Master, Slave, or multiple device mode (M[2:0]). The configuration control signals are:

- D[31:0] is a 32-bit bidirectional data bus,
- INIT_B input that can be held Low to delay configuration or output indicating whether a CRC error occurred during configuration,
- PROGRAM_B Active-Low asynchronous full-chip reset,
- RDWR_B determines the direction of the D[x:0] data bus,
- CS_B Active-Low chip select to enable the SelectMAP data bus,
- CCLK is a clock input in the slave mode and clock output in the master mode,
- BUSY indicates that the device is not ready to send readback data. The BUSY signal is only needed for readback,
- DONE is an Active-High signal indicating configuration is complete.

ICAP works in the same way as the SelectMAP configuration interface, except it is accessed internally, and ICAP has a separate read/write bus, as opposed to the bidirectional bus in SelectMAP. It allows the user to access configuration registers, readback configuration data, or partially reconfigure the FPGA after configuration is done. The interface of the ICAP library primitive is shown in Figure 8c. All the I/Os can be interfaced internally. It has four inputs and two outputs:

- I[31:0] ICAP write data bus,
- CE Active-low ICAP chip enable,

- WE determines read or write operation,
- CLK ICAP interface clock,
- O[31:0] ICAP read data bus,
- BUSY active high busy status. Used only in read operations.

3.1.4 Partial runtime reconfiguration of FPGA

Specific FPGA families allow performing partial reconfiguration, where a reduced bitstream reconfigures only a given subset of internal components (Vaidyanathan and Trahan, 2004). Partial Runtime Reconfiguration (PRR) is done while the device is active: certain areas of the device can be reconfigured while other areas remain operational and unaffected by the reprogramming. Partial reconfiguration is useful for applications that require loading different designs into the same area of the device or the flexibility to change portions of a design without having to either reset or completely reconfigure the entire device. With this capability, entirely new application areas become possible:

- in-the-field hardware upgrades and updates to remote sites,
- self-test and recovery of FPGA configuration memory,
- precise fault emulation,
- adaptive hardware algorithms,
- continuous service applications.

The primary advantages of run-time reconfiguration in devices are reduced power consumption, higher resource utilization, obsolescence avoidance, and flexibility. The costs of run-time reconfigurability are in the design and implementation complexity, both in architecture definition and in coding.

3.1.4.1 Types of partial reconfiguration

There are different ways to access and reconfigure the FPGA. According to the interface, through which the FPGA is reconfigured, we have two approaches:

- External reconfiguration. In this case, the reconfiguration is made externally, usually through a serial JTAG or parallel SelectMAP. So the device is totally dependent on the host, which takes the decision to reconfigure and provides the necessary data.
- Internal reconfiguration. In this case, the reconfiguration is performed in the system. The system neither needs an external source to reconfigure itself, nor to take the decision to reconfigure. It means that the system is fully autonomous. In the Virtex 4 and Virtex 5 family FPGAs, self-reconfiguration is done through the ICAP.

3.1.4.2 Implementation flows for a partial FPGA reconfiguration

There are three main flows for partial FPGA reconfiguration: difference-based, module-based, and frame-based (Lim and Peattie, 2002).

Difference based: This method of partial reconfiguration is accomplished by making a change to a design (normally done in Xilinx FPGA editor), and then by generating a bitstream based only on the differences in the two designs (Eto, 2007). Switching the configuration of a module from one implementation to another is very quick, as the bitstream differences can be much smaller than the entire device bitstream. This flow can also be used for modular design when there is no communication between modules.

Module based: Partial reconfiguration involves defining distinct portions of an FPGA design to be reconfigured while the rest of the device remains in active operation. These

portions are referred to as reconfigurable modules. The complete design is therefore divided into reconfigurable modules. Module-based partial reconfiguration is used when communication is needed between the modules.

In the older module-based partial reconfiguration flow (Virtex 2, Virtex 2 pro and also applicable to Virtex 4 and Virtex 5) a special bus macro is needed for the communication between reconfigurable modules. The bus macro provides a fixed bus of inter-design communication. Each time partial reconfiguration is performed, the bus macro is used to establish unchanging routing channels between modules, guaranteeing correct connections. A layout with three reconfigurable modules divided by bus macros is shown in Figure 9. For designs where the modules are completely independent (no common I/O except clocks) and there is no communication between modules, bus macros are not needed.

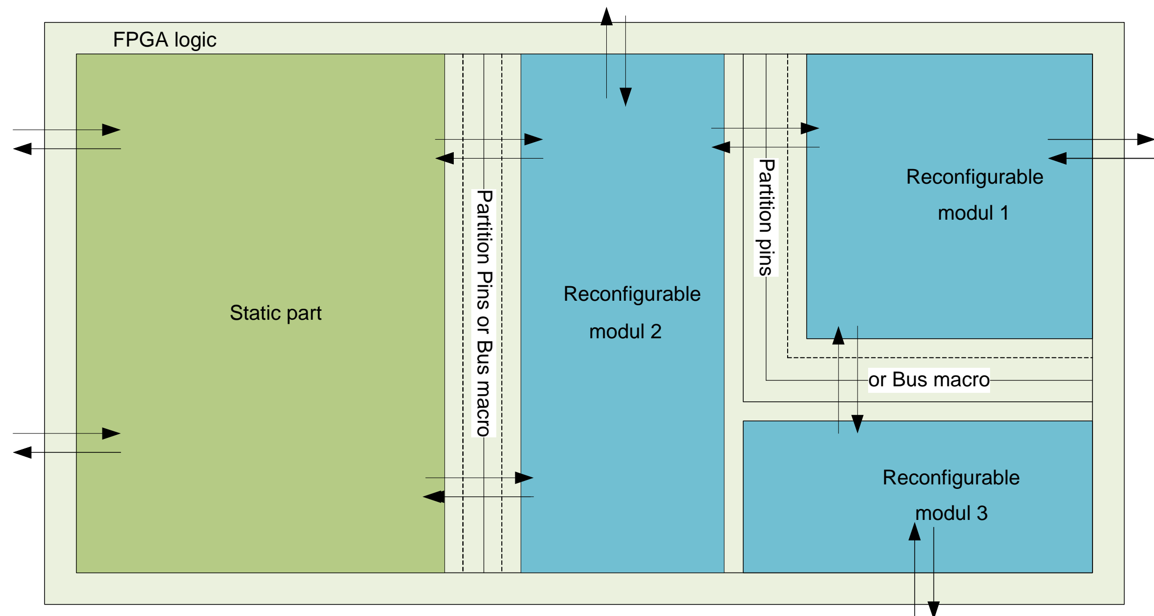


Figure 9: FPGA design layout with three reconfigurable modules

The most recent module-based partial reconfiguration flow is performed in Xilinx PlanAhead software (Xilinx, 2010c). The software defines Static and Reconfigurable Partitions (RP). Partitions provide common routing connections for reconfigurable modules through Partition Pins (PP). PPs are a replacement technology for bus macros from the previous modular partial configuration flow. The current implementation of PP requires proxy logic, which is a LUT1. The LUT1 is being inserted into the input and output paths of the RP, and it is recommended to register these inputs/outputs on both sides of the RP boundary. After creating partitions the designer adds the reconfigurable modules for each RP and, after verification, generates bit files required for partial reconfiguration in the hardware.

Frame based: This partial reconfiguration flow is the most challenging. The configuration designer defines or alters the configuration memory of the particular configuration frame. The designer has to have expert knowledge of the bitstream structure. This flow is used in fault-injection and error-recovery processes which will be further explained in sections 6 and 7.

3.2 Radiation effects in integrated circuits

As device dimensions continue to shrink integrated circuits are becoming much more

susceptible to radiation-induced soft errors. Soft errors are random "glitches" in a semiconductor device. These glitches are usually not catastrophic, and can normally be recovered by resetting the device. Soft errors are caused by a charged particle striking the semiconductor logic, memory or a memory-type element. Specifically, the charge generated by the interaction of an energetic charged particle with the semiconductor atoms corrupts the stored information in the memory cell – this phenomenon is called a Single Event Upset (SEU) – or has a transient effect on the semiconductor logic called a Single Event Transient (SET). These charged particles can come directly from radioactive materials, cosmic rays, or indirectly as a result of high-energy particle interaction with the semiconductor itself (Schrimpf and Fleetwood, 2004).

High-energy cosmic rays and solar particles react with the upper atmosphere generating high-energy protons and neutrons that shower to the ground. Neutrons are particularly troublesome as they can penetrate most man-made construction (a neutron can easily pass through five feet of concrete). This effect varies with both latitude and altitude. In London, the effect is two times worse than on the equator. At places with high elevation above the sea, the effect can be three or four times worse than at sea-level. In an airplane, the effect can be 100-800 times worse than at sea-level.

Other common sources of these errors are alpha particles, which are emitted by the trace amount of radioactive isotopes present in the packaging materials of integrated circuits. Bump materials used in the new flip-chip packaging technique have also been recently identified as containing significant alpha-particle sources.

Figure 10a shows the effect in the case of a heavy ion strike. Upon impact with the silicon, the heavy ion loses its energy and produces free electron-hole pairs, resulting in a dense ionized track in the local region (Messenger, 1982).

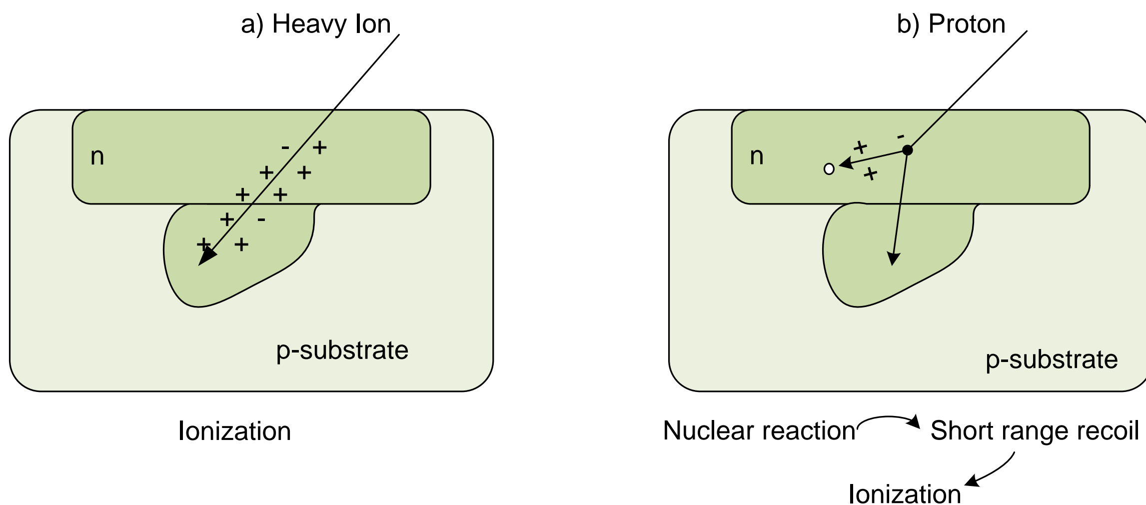


Figure 10: Charged particle striking the silicon circuit

Figure 10b shows an effect in the case of a proton or neutron strike. When a particle passes through the silicon surface it causes a nuclear reaction. The recoil also produces ionization. The ionization generates a charge deposition that can be modeled as a transient current pulse that can be interpreted as a signal in the circuit causing an upset. The current pulse at the particle strike can be modeled by a double exponential function from equation (1).

$$I_p(t) = I_0(e^{-t/\tau_\alpha} - e^{-t/\tau_\beta}) \quad (1)$$

where I_0 is approximately the maximum charge collection current, τ_α is the collection time constant of the junction, and τ_β is the time constant for initially establishing the ion

track. Figure 11 shows an example of a current pulse induced by a particle hit. The area under the curve corresponds to the amount of collected charge (Q).

The influence of radiation in the material is measured by the energy and flux of the particles that hit the material. The energy transferred to the device is called Linear Energy Transfer (LET). The minimal LET that can cause a SEU is called the LET threshold. There are many levels of robustness according to the amount of flux and energy transferred to the silicon that can keep the circuit operating properly. The probability of the particular particle causing an upset in the device is called a cross-section (σ). Consequently, the sensitivity of a device to an upset is measured in the cross-section in terms of the LET.

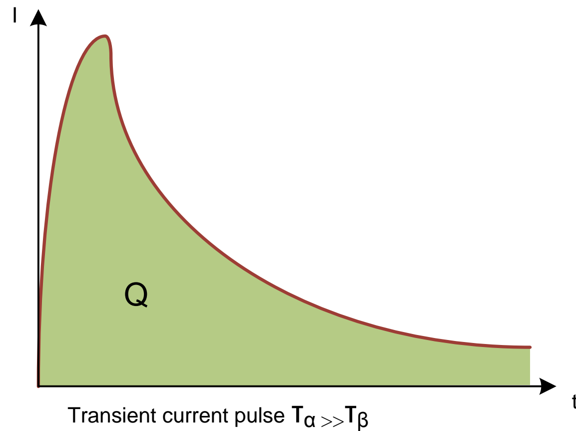


Figure 11: Transient current pulse induced by a particle hit

The SEU error rate or the reliability of a particular semiconductor device is stated in Failures In Time (FIT) or mean time between failures (MTBF). The FIT is the number of failures that can be expected in 10^9 hours of device operation and the MTBF is the mean time between two failures.

3.2.1 Soft errors in integrated circuits

A charged particle strike can affect the combinational logic or sequential logic of integrated circuits, as illustrated in
Figure 12 (Crain et al., 2001; Alexandrescu et al., 2002). Soft errors in integrated circuits are classified as single-event upsets (SEU), single-event transients (SET), and multiple-bit upsets (MBU).

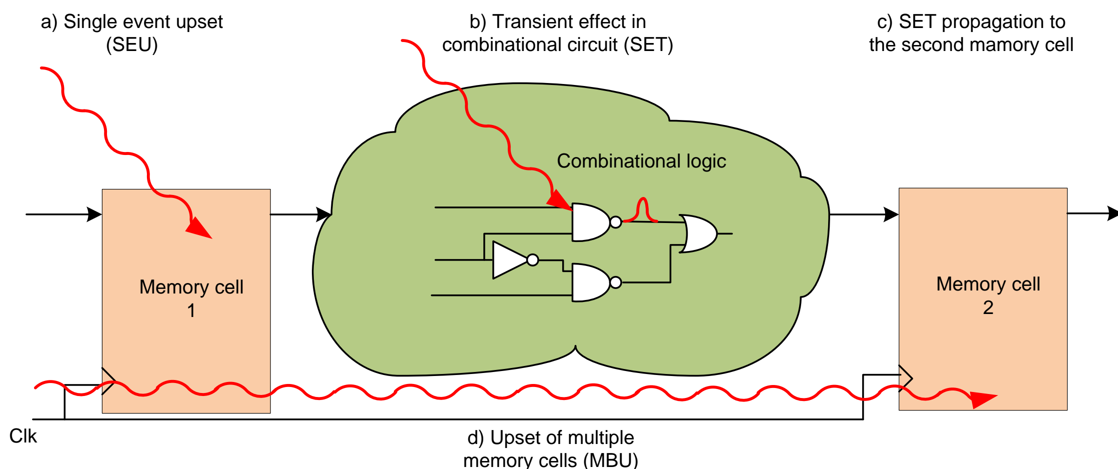


Figure 12: Soft errors in integrated circuits

A SEU occurs when a charged particle strikes a sequential memory cell and changes its state (

Figure 12a). For example, a typical SRAM memory cell is comprised of four transistors shown in Figure 13. A memory cell has two stable states that represent one bit of stored information. In each state two transistors are turned *off* (SEU target drains). When a charged particle strikes a drain in an *off* state transistor (Figure 13a), it can generate a transient current pulse to turn the gate of the opposite transistor *on*, which changes the state of the memory cell (Figure 13b).

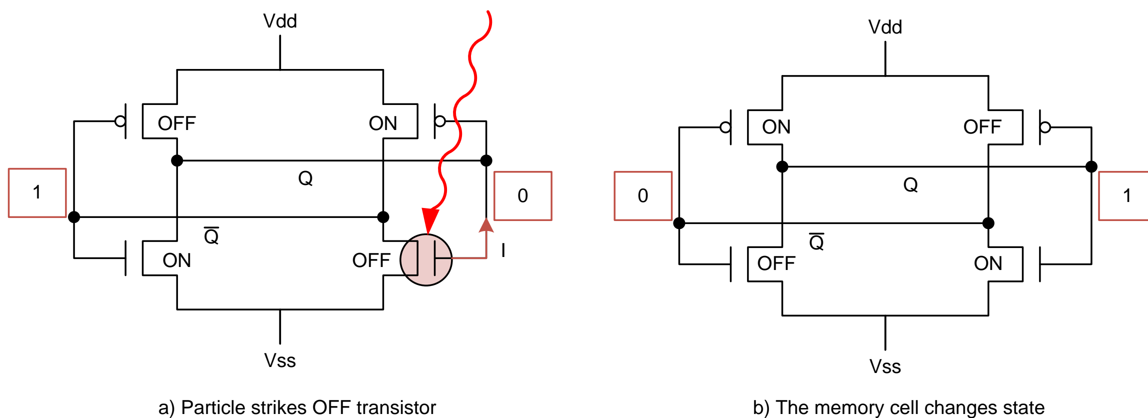


Figure 13: SEU in a SRAM memory cell

When a single particle hits a combinational logic of the circuit it generates a transient current pulse called a single-event transient (SET) (Leavy, 1991) (
Figure 12b). If the logic is fast enough to propagate a transient pulse, then the SET will appear at the second memory cell (

Figure 12c). Whether or not the SET gets stored in the memory cell depends on the duration of the pulse and the time of the rising edge of the clock.
MBU can occur when a single charged particle traveling through the semiconductor device at a shallow angle almost parallel to the surface of the die, simultaneously upsets two or more memory cells (

Figure 12d). Another possible occurrence of the MBU is when a SET in a combinational circuit propagates to two or more memory cells. The third type of MBU is when multiple particles strike multiple memory cells.

3.2.2 Soft errors in SRAM-based FPGAS

SRAM-based FPGA, as an integrated circuit, is also susceptible to radiation-induced soft-errors. The configuration memory of a SRAM-based FPGA is comprised of SRAM memory cells. A charged particle can cause a SEU in the configuration memory cell and consequently alter the FPGA functionality. A configuration bit is associated with a particular part of the FPGA. It can be a part of an internal memory of the device like a block RAM or flip-flop, or it can represent a functional part of the design, like a Configurable Logic Block (CLB), or internal routing (Rebaudengo, 2002).

The internal routing interconnects the CLBs, IOBs, and other functional blocks of the FPGA. The routing consists of switch boxes and wiring segments, shown in Figure 14a. The connections of the switch box and the wiring segments are determined by the logic state of their configuration bits. A SEU affecting these configuration bits could disconnect

the original CLB connection, or in another case, connect the wrong CLBs. For illustration, some typical faults are marked in Figure 14b.

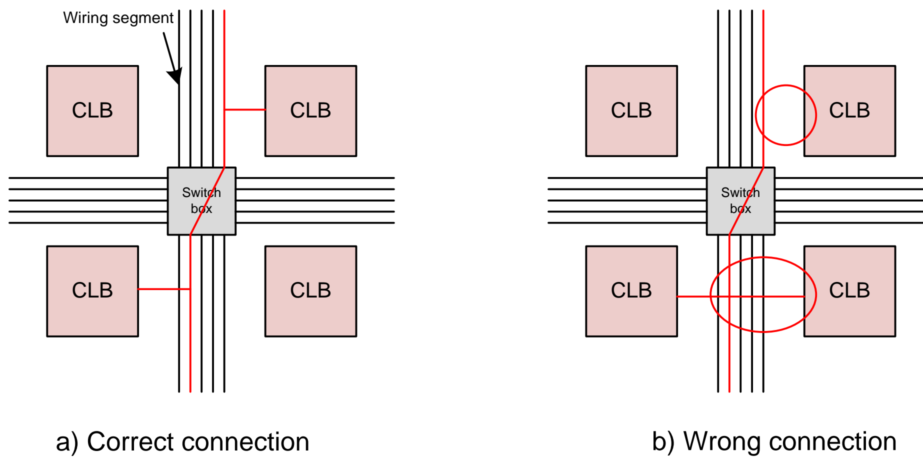


Figure 14: SEU in an internal FPGA routing

The simplified structure of a CLB is shown in Figure 15. The CLB in Xilinx FPGA consists of a number of look-up tables, flip-flops and internal carry and control logic. Candidates for SEU-induced faults in a CLB are:

- Look-up table bits. A SEU changes the logic function implemented by the look-up table (INIT [0-15]). A SEU can also change look-up table configuration bits and affect its functionality. For example, the bits that determine whether a look-up table resource is configured as a look-up table, as a dual port RAM (DUAL_P), or as a shift register (S_REG).
- Multiplexers and inverters inside the Carry and control logic. A SEU changes the internal routing of a CLB.
- Flip-flop configuration. A SEU changes the state of the flip-flop or changes the flip-flop mode (INIT, S/R MODE, etc.).

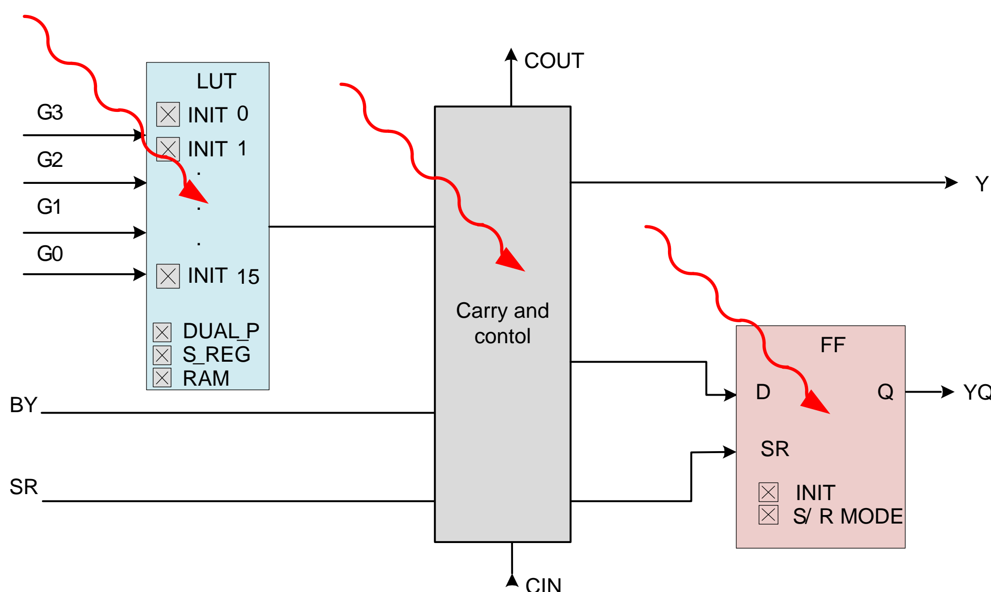


Figure 15: SEU in Configurable Logic Block

Note that a typical user design uses only a part of the whole FPGA configuration image. Therefore, a SEU in the redundant parts of the FPGA can be ignored. The faults in the configuration bits that are used to define the target design can affect the operation of the design. These configuration bits are considered to be potentially critical.

A SEU can also occur in a vital part of the FPGA configuration memory, which causes regional or device-wide failure. These faults are referred to as Single Event Functional Interrupts (SEFI). The observed SEFI in Virtex 4 and 5 devices occur on Power-On-reset (POR), Select Map or ICAP registers, Digital Clock Managers (DCM), and global signals (Global Write Enable, Global 3-state Control, GHIGH, etc.).

3.2.3 The reliability of Xilinx FPGA devices

The reliability of Xilinx devices is being evaluated in an ongoing Rosetta experiment (Lesea et al., 2005). Rosetta is a real-time experiment that measures the actual upset rate of Xilinx FPGAs due to cosmic-ray-induced atmospheric neutrons. Each Rosetta experiment consists of multiple sets of a 100 of Xilinx largest FPGAs of different technologies and is located at four locations at four different altitudes. Each group of 100 FPGA devices is composed of ten separate boards, with ten devices on each board. All 100 devices are wired into a serial JTAG test chain, so that the configuration memory (which includes the look-up table RAM, and the BRAM) can be written, read back, and verified. The current SEU error rates of the devices can be found in the annual Xilinx device reliability report (Xilinx, 2011). Table 1 shows the current atmospheric test results for different FPGA technologies. With increased transistor density the SEU rate of the FPGA devices does not decrease because of the more robust design of the configuration memory cells. Block RAM cells are small-size cells similar to a commercial SRAM. Therefore, the upset performance of the block RAM is not as robust as the configuration cells.

Table 1: Failures In Time (FIT) by FPGA technology

FPGA technology ->	150nm	130nm	90nm Virtrx4	65nm Virtex5
Configuration Memory				
Data Failure Rate (FIT/Mb)	401	384	246	151
95% confidence (FIT/Mb)	367 to 435	339 to 429	199 to 301	101 to 215
Block RAM				
Data Failure Rate (FIT/Mb)	397	614	352	635
95% confidence (FIT/Mb)	317 to 491	515 to 713	236 to 506	428 to 907

The Rosetta experiment demonstrates that:

- the FPGA structure by its nature is highly resistant to SETs due to the large capacitive loading of the signal paths. This loading is many times that of the loading in an application-specific integrated circuit (ASIC) or application-specific standard product (ASSP).
- in the majority of cases, a SEU only changes (flips) a single bit. Multiple-bit upsets (MBUs) due to a single ionizing particle almost never occur,
- there is a high probability that the upset bit in the configuration memory will not affect the functionality of the design because normally less than 10% of the configuration cells are significant to the design implementation,

- block RAM memory cells are more susceptible to SEUs than the configuration cells. For this reason Xilinx introduced an ECC option for BRAM in their Virtex 4 FPGA family (ECC is also available in newer families).
- flip-flops are least likely to suffer from a SEU. The flip-flop of a Virtex 5 device has approximately 0.06 FIT/Mb, which is equivalent to nearly 2 million years of MTBF for a medium-sized device.

From the experiment we can conclude that reliable FPGA applications should provide a single error-recovery strategy against SEUs in the configuration memory and block RAM. The mitigation strategies of flip-flops are normally omitted.

4 On-line testing and fault-tolerance techniques

As described in Chapter 3.2 integrated circuits, and especially SRAM-based FPGAs, are susceptible to radiation-induced errors. Therefore, it is imperative that the devices, where high reliability is required, are protected.

The first SEU protection measure was shielding, which reduces the particle flux to lower levels. Nowadays, with the technology evolution and scaling, devices have become more sensitive and the particles that escape shielding also cause soft-errors. Consequently, other means of SEU protection are used. The most common approach is to include on-line testing (Nicolaidis and Zorian, 1998) and fault-tolerance techniques (Lima et.al, 2006).

In an off-line electronic test the normal operation of the system is stopped and the system is stimulated using a predefined set of test vectors and tested using output analyzers. These test vectors can either be applied by an external tester or they can be stored on the chip and applied during the test mode. The latter technique is called a built-in-self-test or BIST. The on-line testing techniques, on the other hand, test the system during its operation. The test vectors are replaced by the operational input stimuli and the output analyzers are replaced by different on-line output observers (checkers). Fault-tolerance techniques that operate on-line are not only able to test the system, but also avoid errors and recover errors from the system. These techniques are called error-mitigation techniques and error-recovery techniques.

The key benefits of on-line testing and fault tolerance include low fault-detection latency, detection of transient faults, fault mitigation, and error recovery.

4.1 On-line testing techniques

On-line testing is performed while the circuit is performing its assigned task. Two types of on-line test are distinguished in the literature: the non-concurrent and the concurrent on-line test.

A non-concurrent on-line test is usually triggered in phases of system inactivity or in periodic and scheduled times when the normal function of the system is interrupted. Non-concurrent testing is used to detect permanent faults (SEU) and cannot detect transient faults (SET), whose effects disappear quickly. The authors in (Al-Asaad and Shringi, 2000) used scan chains to periodically check the system while (Yang et.al, 2008) used logic BIST. (Paschalis and Gizopoulos, 2005) implemented a periodic on-line test in an embedded microprocessor and (Aktouf et al., 1999) tested the resource that were currently in idle state. The non-concurrent on-line test is performed only virtually in parallel to system operation. Therefore, it is in some literature classified as an off-line technique. In the rest of the dissertation the on-line testing is considered as concurrent on-line testing technique.

A concurrent on-line test runs in concurrence with the system and does not interrupt its normal operation. The concurrent testing techniques use different kinds of redundancies to detect errors. Time redundancy is normally used to detect a SET in combinational circuits, while hardware redundancy is used to detect a SEU and a SET in sequential circuits. The on-line testing principle is depicted in Figure 16. Test vectors are generated

by the normal operational inputs. Besides the original circuit there is a redundant part of the circuit that produces additional encoded outputs. A checker is monitoring these outputs and thus performing the error detection.

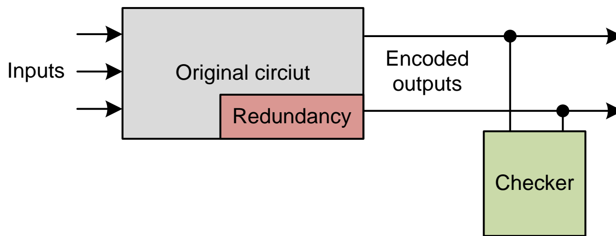


Figure 16: Concurrent on-line testing principle

A system protected with the on-line testing technique is also called a *self-checking system* (Nicolaidis and Zorian, 1998). The desirable goal of self-checking systems is to achieve the so-called *totally self-checking* property. This property requires that every fault in the system is detected before or at the time this fault produces an erroneous output. To achieve this goal the system has to meet the following criteria defined by (Carter and Schneider, 1968; Anderson, 1971):

- *Fault secure* criterion requires that any fault in the system produces erroneous outputs that can be detected at the output. This criterion assures that every single fault can be detected.
- *Self-testing* criterion requires that for each fault there is at least one input vector, occurring during normal operation of the circuit, which detects it. This criterion avoids the existence of redundant faults.

The circuit or system is thus totally self-checking if it meets these two criteria. For example, if a fault occurs in the totally self-checking system than it is detectable and if the MTBF is long enough then this fault will be detected before the next fault occurs.

In the rest of this chapter some basic techniques of concurrent on-line testing are presented with examples from the literature. The techniques are based on different kinds of redundancies.

Time redundancy techniques are normally used to check combinational circuits for temporal glitches in the logic called transient faults (SET). The principle of this technique is to check the output of the combinational circuit in two different moments and compare the results. The example of the technique is shown in Figure 17. The output of a combinational circuit is latched in two different times. The clock edge of the second latch is shifted by Δt, where Δt has to be shorter than the duration of the transient pulse generated by the particle strike. The outputs of the two latches are compared with a comparator that detects the pulse and reports the error.

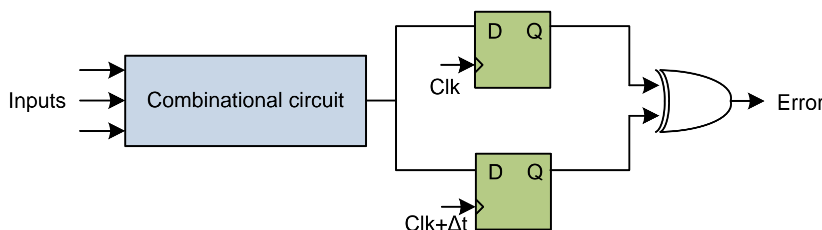


Figure 17: Time-redundancy technique to detect a SET in combinational circuits

In a special case of this method the Δt is equal to half a clock cycle, which means that the output is latched at the positive and negative edge of the clock cycle (double data-

rate). To test the encryption algorithm (Maistri et al, 2008) use both clock edges' sample data within the registers and compare the results.

The most straight forward hardware redundancy technique is duplication and comparison. The principle of the technique is shown in Figure 18. Two copies of the circuit run in parallel, receiving the same inputs and outputs of the both circuits, are compared by the comparator circuit. This technique increases the hardware cost by more than 100%.

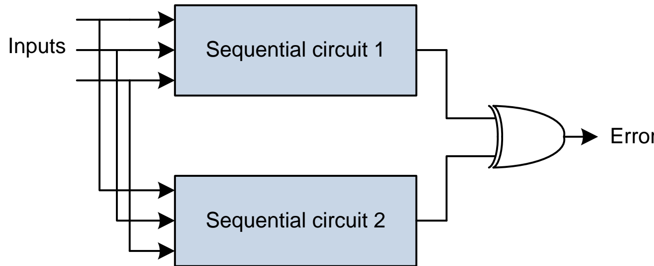


Figure 18: Duplication and comparison technique to detect SET and SEU in sequential circuits

Duplication and comparison is used in a variety of different systems. The on-line testing of embedded processor cores was improved by (Violante et. al, 2011). The processor cores are duplicated and the checker monitors whether the outputs of both cores match. If the outputs mismatch the processor context is restored from the previously saved states (checkpoint and rollback recovery method). The applicability of duplication and comparison in asynchronous circuits was investigated by (Verdel, and Makris, 2002). The testing of finite-state machines using a technique similar to duplication and comparison was proposed by (Drineas and Makris, 2003).

To reduce the hardware cost other, more elaborate techniques are employed. These techniques use error-detecting codes (EDC) with costs lower than the duplication. The EDCs are used in sequential circuits and in memories.

Parity code is the most simple and the least hardware consuming. An example of the technique is depicted in Figure 19. Protected registers inside the circuit are extended to accommodate the parity check bit. The parity bit of the registers is calculated at the input of the sequential circuit. Then the parity bit is propagated through the circuit and checked at the output. Note that the circuit designer has to ensure the correct propagation of the parity through the circuit.

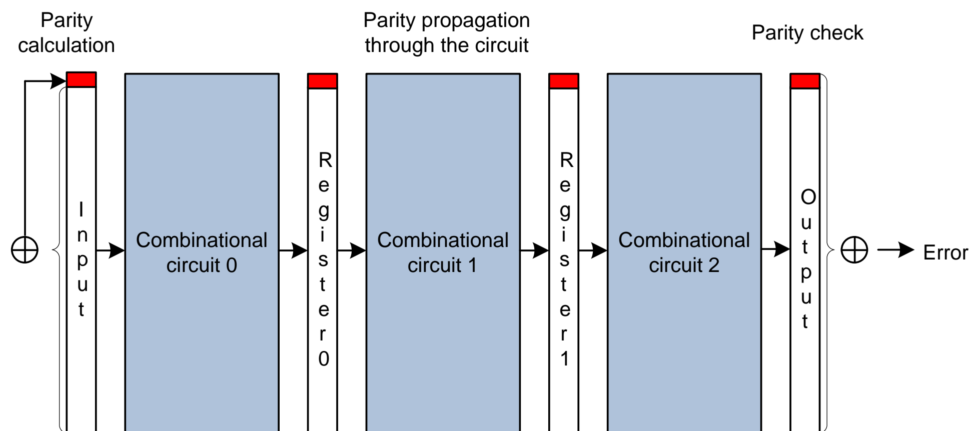


Figure 19: Error-detection technique using parity check

Dual-rail code is a variety of the duplication code. Check bits are equal to the complements of the information bits. The technique detects any errors affecting either the information part or its complement. But the technique is quite expensive since it duplicates the information.

Unordered codes are simply the codes in which no code word is contained in any other code word. The most used unordered codes are the *m out of n code* and the *Berger code*.

An m out of n code (Freiman, 1962) is composed of code words that have exactly m 1s. For example *2*-out-of-*4* codes are: 1001, 1100, 1010. This code is non-separable, which means that the information and check bits are merged.

The Berger code (Berger, 1961) is a separable unordered code. The check part of this code is the number of 0s inside information data. For example: if information is 10001010 then the check is 101. This code is an optimal separable unordered code. The number of check bits for *n* information bits is log *2(n+1).*

Arithmetic codes (Peterson, 1958) are also divided into separable and non-separable. The most used are the separable arithmetic codes. If the codes have a base A, information part X and check part X'. Then the check part X' is equal to X modulo A ($X' = |X|_A$). Arithmetic codes are interesting for checking arithmetic operations, because they are preserved under such operations. These codes are most often implemented as low-cost arithmetic codes where the check base A is equal to $2^m - 1$.

4.2 Error-mitigation techniques

For mission-critical systems it is sometimes not enough to only detect a fault, but also to operate in the presence of a fault which is possible by applying error-mitigation techniques to the system. These techniques are also based on different kinds of redundancies.

The error-mitigation technique based on time redundancy is shown in Figure 20. In this so-called full-time redundancy technique the output of the combinational circuit is latched at three different times. The clock edge of the second flip-flop is shifted by Δt and the clock edge of the third flip-flop is shifted by $2\Delta t$. The time Δt depends on the duration of the transient pulse (SET). The duration of the transient pulse is approximately Q_{col} / I_D, where Q_{col} is the collected charge during the charged particle strike and I_D is an average drain current. The outputs of the flip-flops (A, B, C) are connected to a majority voter circuit. The voter circuit outputs the result based on a majority vote. If one latch output is erroneous and the other two are correct then the circuit works correctly, as shown in the truth table in Figure 21. The majority voter is a combinational circuit and can be implemented using three two input AND gates and one three input OR gate. The schematic of the voter circuit is also presented in Figure 21. The hardware overhead of this method is in two extra latches and the voter circuit for each output signal and the timing delay is approximately $clk + 2\Delta t + t_v$, where t_v is the delay of the majority voter.

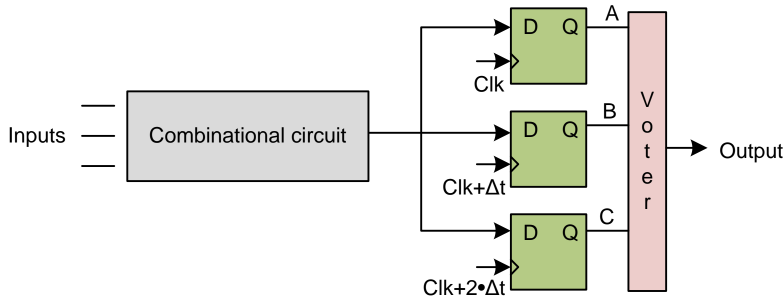
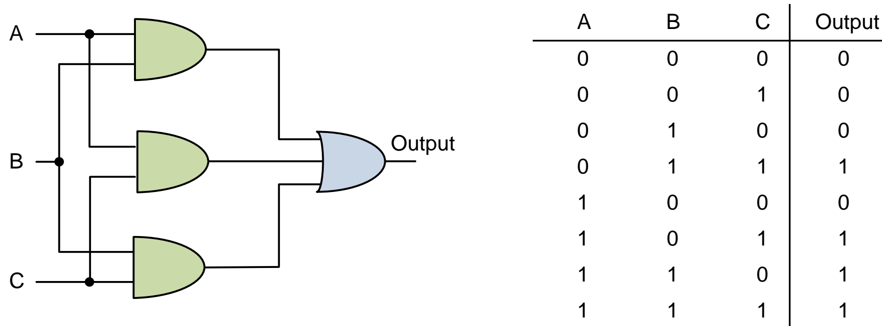


Figure 20: Time-redundancy technique to mitigate a SET in combinational circuits



A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 21: The majority voter scheme and truth table

The best-known hardware-redundancy mitigation technique is Triple Modular Redundancy (TMR). This technique is one of the *n*-modular redundancy techniques which were derived by (Von Neumann, 1956). The basic TMR technique triplicates the entire circuit into three modules and places the majority voter at the output of the modules, as shown in Figure 22. This method is effective against SETs and SEUs that occur in a single design module. However, if the upset occurs in the majority voter circuit the basic TMR is ineffective and a wrong value will be presented at the output. The hardware overhead of this method is three times the original design plus the voter circuit. While the hardware overhead is large, some have proposed partial TMR techniques which are focused only triplicating the specific sensitive logic (Pratt et al., 2008).

The basic TMR solution does not avoid the accumulation of upsets. The FPGAs cope with this problem by implementing an on-line error-recovery technique, which is explained in the next section.

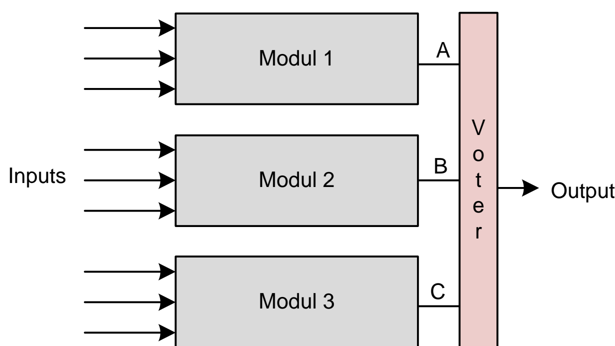


Figure 22: The architecture of the basic TMR technique

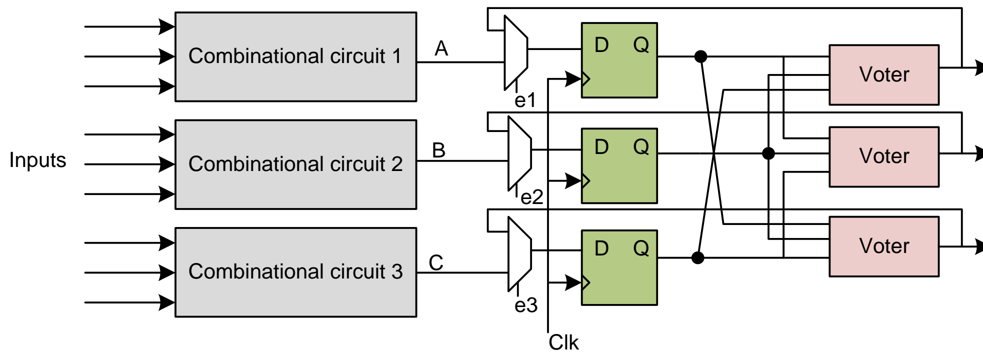


Figure 23: The architecture of TMR for the states recovery

A TMR method with states recovery was proposed by (Katz et al. 2001). The technique is depicted in Figure 23. The combinational and sequential logic of the circuit are protected separately. The combinational logic is triplicated. The sequential logic cells are triplicated and connected to three voters. The voters are fed back to the input of the flip-flop to correct the contents of the flip-flop in the case of an error. To implement the states recovery additional logic is needed for the generation of multiplexer control signals (e1, e2, e3).

To apply the TMR technique effectively in the FPGA device additional restrictions had to be considered. The triplicated modules had to be placed isolated from each other (different clock regions) and the internal signals had to be carefully routed to limit the possibility that an upset would affect more than one module. All the modules have to have separate clock and input signals. Routing the TMR design in the FPGA is a particularly hard problem. Various algorithms and design methods have been proposed to reduce the number of such errors (Lima Kastensmidt et al., 2005; Sterpone et al., 2006; Sterpone and Violante, 2008).

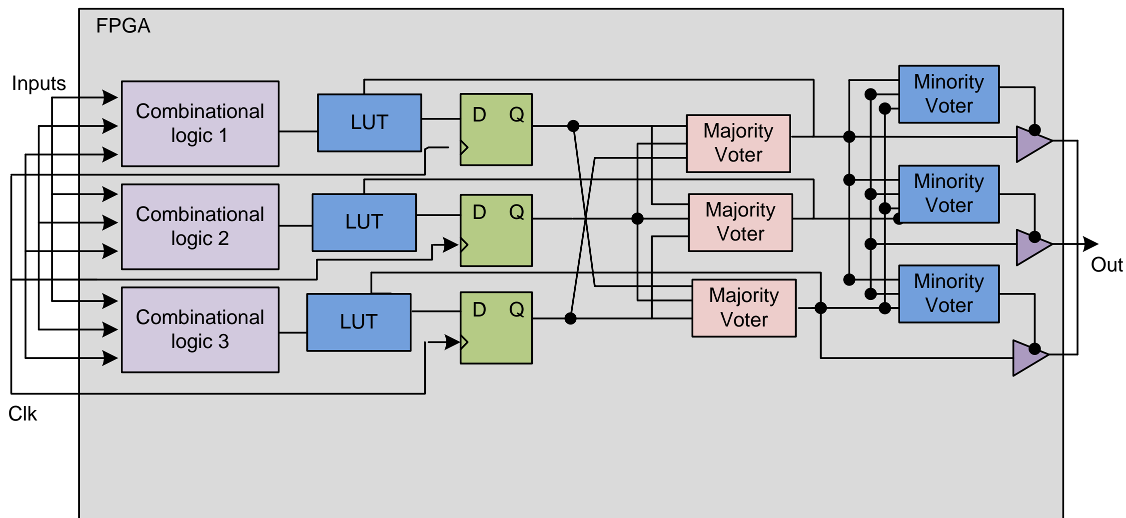


Figure 24: The architecture of Xilinx TMR

For Xilinx FPGAs a special hardened TMR architecture has been proposed in (Carmichael, 2000; Xilinx, 2007). This architecture can also be automatically generated by their tool (Xilinx TMRTool). The XTMR is exploited based on the states recovery TMR method and uses specific FPGA resources to implement the majority voters. The Xilinx TMR is depicted in Figure 24. :

- Inputs are connected outside of the FPGA and separated inside the FPGA
- Combinational logic is triplicated
- The sequential logic is implemented with majority voters and feedback loops
- Outputs are implemented using tri-state output buffers in combination with minority voters

Error-correcting codes (ECCs) are also used to mitigate the SEU in integrated circuits (Petersen, 1980). Many codes are used to protect the systems against single and multiple SEUs: Hamming codes (Hamming, 1953), Reed-Solomon codes (Reed and Solomon, 1960), BHC codes (Bose and Ray-Chaudhuri, 1960), etc. ECCs are mostly used to protect the memories and registers of the systems.

The most commonly used ECCs are Hamming codes. The Hamming single error-correction and double error-detection code (SEC-DED) is used in the systems where the probability of multiple errors is very low. The procedure of error correction in memories or registers in integrated circuits is presented in Figure 25. At the input the Hamming check bits are calculated added to the original word to build the code word (Hamming encoder) and at the output the *syndrome value* is calculated, which determines the state of the error (single or double) and the location of the single error in the code word.

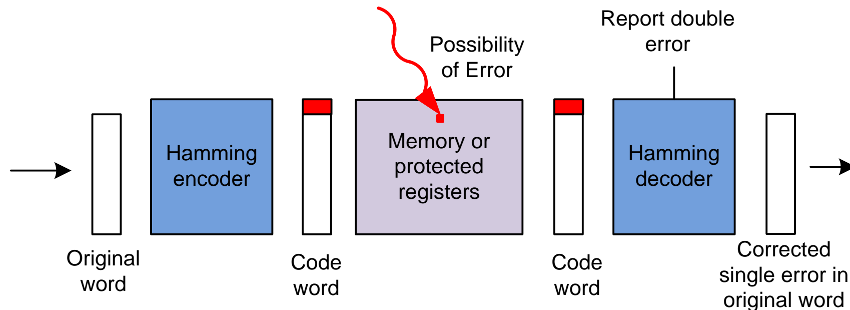


Figure 25: Hamming code error-correction implementation

The Hamming SEC-DED code satisfies the relation (2), where k is the number of check bits for a single error detection, m is the length of the original data word and $m+k+1$ is the length of the code word.

$$2^k \geq m+k+1 \tag{2}$$

For example, if we have 8-bit data then we need 4 check bits plus one overall check bit for double-error detection.

The encoder for an 8-bit word is shown in Figure 26a. The Hamming check bits *p0-p4* are located at positions 0, 1, 2, 4, and 8 in the code word respectively. The check bits *p1-p4* are determined as an even parity of bits at the positions marked in grey. *p0* is an even parity of the entire code word used to detect double errors. The calculation of even parities can be performed by XOR operation and implemented using XOR logic gates.

The decoder for an 8-bit word is shown in Figure 26b. The decoder calculates the error *syndrome*. The syndrome bits (*b0-b4*) are determined as an even parity of the bits at positions marked in grey. The syndrome value has the following options:

- (*b0-b4*) = 0, no error occurred.
- $b0 = 1$ and $(b1\text{-}b4) \neq 0$, single error occurred. The syndrome bits *b1-b4* point to the location of the error within the code word.
- $b0 = 0$ and $(b1\text{-}b4) \neq 0$, double error occurred.

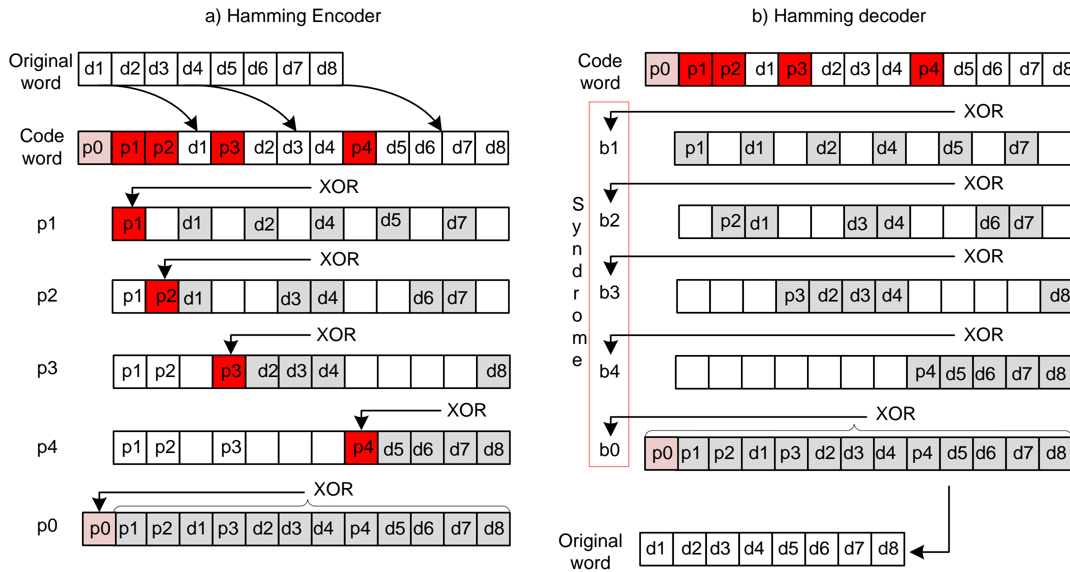


Figure 26: Hamming code encoder and decoder implementation

4.3 Error-recovery techniques in SRAM-based FPGAs

The configuration memory of the FPGA determines the functionality of the FPGA. The original configuration is downloaded onto an FPGA at the power-up. During the operation of the device a SEU can occur, which changes the configuration of the FPGA. The error-recovery techniques are used to recover the original state of the configuration memory.

Simple error-recovery techniques (scrubbing techniques) only periodically reconfigure the whole device from the external memory. The external memory has to be radiation hardened and reliable to assure the correctness of the original ("Golden copy") configuration memory. The scrubbing period has to be adjusted to be less than the estimated mean time between two SEUs. The architecture of this technique is shown in Figure 27a.

More advanced error-recovery techniques check whether the configuration memory is correct during the normal operation of the user application and can be regarded as on-line recovery techniques. These techniques require an error-recovery mechanism that monitors the configuration memory and in the case of an error recovers only the faulty bits using the partial runtime reconfiguration of the FPGA.

The recovery mechanism uses different ECCs to check the configuration memory. In (Carmichael and Tseng, 2009) (Heiner et al., 2008) and (Chapman, 2010) a Hamming code is used. The Hamming check bits are stored within the configuration of the Xilinx FPGAs. (Asadi and Tahoori, 2005) used a Cyclic Redundancy Check (CRC) of the configuration memory. The CRC values are stored separately inside the internal memory of the FPGA.

Depending on which FPGA configuration interface is used to reconfigure the device, the recovery techniques are classified as either external or internal. The principle of external recovery techniques is shown in Figure 27b. The external techniques use one of the external configuration ports (i.e., JTAG, SelectMap). The external recovery technique requires a reliable recovery mechanism. (Hulme et al. 2004) proposed a radiation-hardened processor to control the recovery process, (Asadi and Tahoori, 2005) used a small auxiliary FPGA to check the main FPGA for errors, and (Berg et al., 2005)

implemented a controller in ASIC.

An external recovery mechanism produces extra implementation costs. Hence, the Internal SEU-recovery techniques were proposed. The internal recovery technique is shown in Figure 27c. They use internal configuration interface (for example, the internal-configuration-access-port-ICAP). The internal recovery controller is implemented in the FPGA along with the user application. It has to be small, fast and reliable. (Carmichael and Tseng, 2009), (Heiner et al., 2008), and (Chapman, 2010) use an embedded microprocessor (PicoBlaze) as a configuration controller.

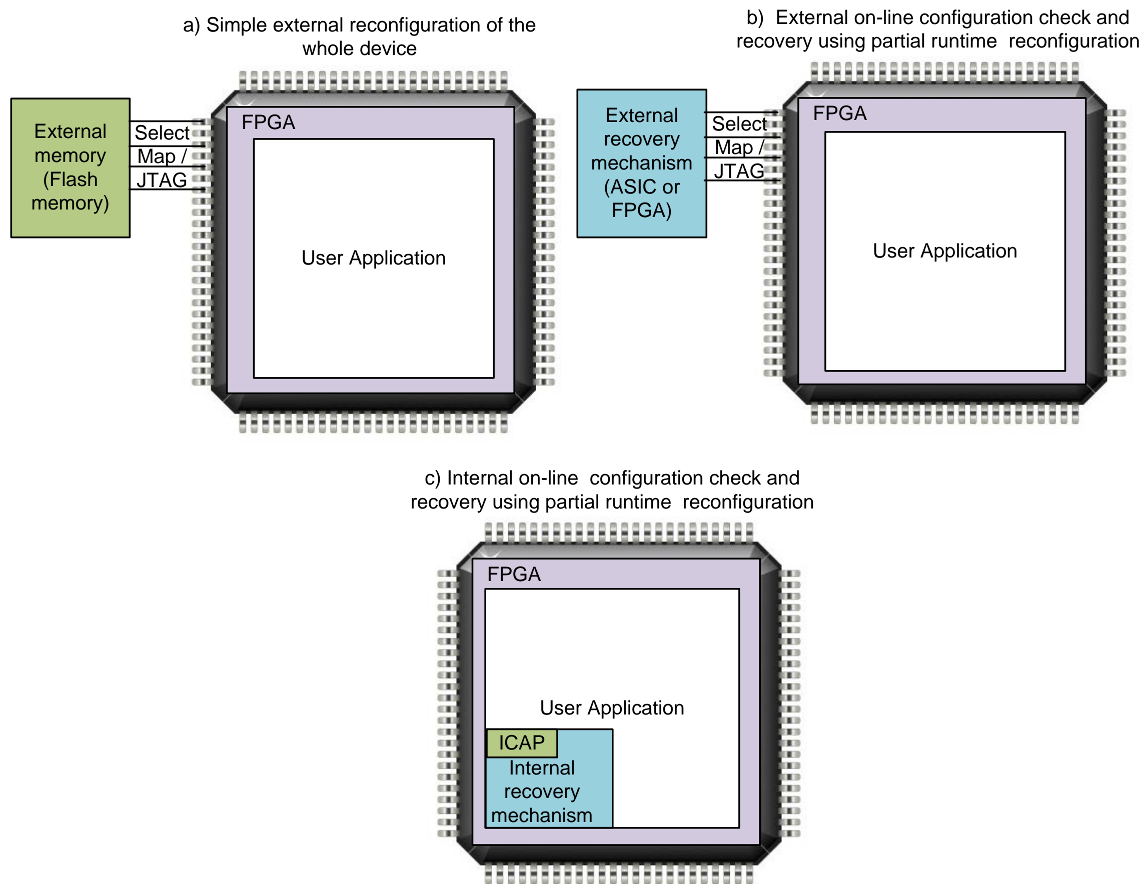


Figure 27: Error-recovery techniques

5 Verification of on-line testing and recovery solutions

To determine the efficiency of the developed on-line testing and recovery solutions we need to create an experiment where we insert a fault in the protected circuit or system and analyze its behavior when subjected to a fault. This process is called fault injection. Various fault-injection techniques have been proposed (Benso and Prinetto, 2003) and can be categorized as physical fault injection, fault injection with HDL simulation, and fault injection with emulation on FPGA (fault emulation).

5.1 Physical fault injection

The physical fault-injection approaches apply once the system or circuit is already available (fabricated circuit). Such approaches include pin-level fault injection, memory corruption, heavy-ion injection, laser fault injection, power-supply disturbances, or software fault injection. An example of a physical fault-injection setup is depictured in Figure 28. A fault is injected using an artificial radiation source. The operation of the tested integrated circuit is monitored by a computer or automatic-test equipment (ATE). This fault-injection technique provides the accelerated radiation test results that are a good approximation to the real environment. The disadvantage of this method is that it is expensive and also it is extremely difficult to inject a fault into a specific target in the system. The method is only suitable for a statistical analysis of the fault coverage and not for determining the complete fault coverage of the system.

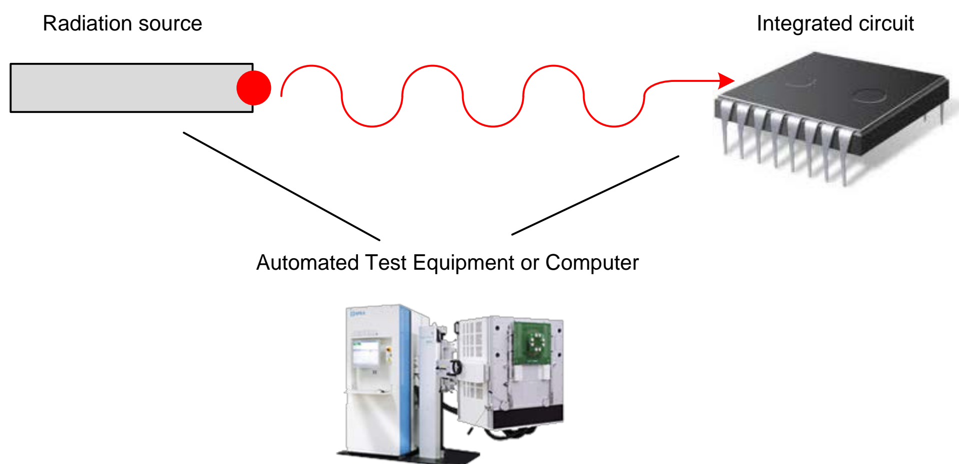


Figure 28: Fault-injection procedure with artificial radiation

5.2 Fault injection with HDL simulation

To avoid later fabrication costs the fault-injection experiment should be applied early in the design process. The proposed approach is to inject faults in high-level models of the circuit or system (most often, VHDL, or Verilog description of the circuit) and evaluate the impact of faults on the circuit behavior using the Hardware Description Language (HDL) simulations. The main drawback related to the use of simulations is the huge

amount of time required to run the experiments when many faults have to be injected into a complex circuit. This fault-injection technique also enables fault injection at lower design levels if the HDL code can be translated into a lower level description of the system.

The fault-injection procedure is controlled by a computer program, as shown in Figure 29. Each fault is injected by changing the HDL code and the response of the faulty circuit is simulated in the HDL simulator.

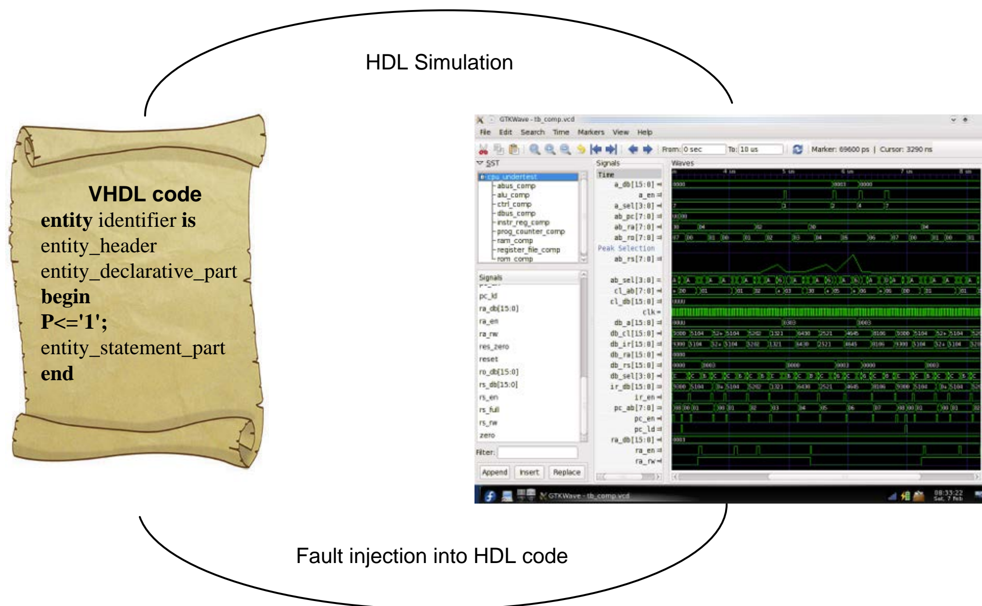


Figure 29: Fault-injection procedure with HDL simulation

5.3 Fault injection with emulation on FPGA

To cope with the time limitations imposed by the HDL fault simulator, it has been proposed to take advantage of hardware prototyping, using FPGA-based hardware emulation.

FPGA is programmed by assigning values to its configuration bits. A group of configuration bits is called a configuration *bitstream*. The faults are injected by changing the original configuration bitstream. The basic fault-emulation procedure is shown in Figure 30. A computer first injects a fault into the bitstream, then it reconfigures (full or partial reconfiguration) the FPGA device with a faulty bitstream and monitors the circuits' response during the emulation process.

The most straightforward approach to hardware fault emulation is based on modification of the high-level description of the system (often assumed to be in VHDL) (Asadi et al., 2004), (Alderighi et al., 2003). The faulty bitstreams are prepared using the manufacturer's synthesis tools and the full configuration of the FPGA is used to inject the fault. A design flow of this method is shown in Figure 31a. A fault is injected into the VHDL code. Then the VHDL is synthesized and a netlist is generated. After placement and routing of this netlist, a bitstream is obtained that can be downloaded to the FPGA. The faulty application is then executed and the results are analyzed. In this approach the synthesis, placement, and routing has to be made for each fault separately and the whole configuration has to be downloaded to the FPGA device. This method is therefore very time consuming but does not require a knowledge of the bitstream structure.

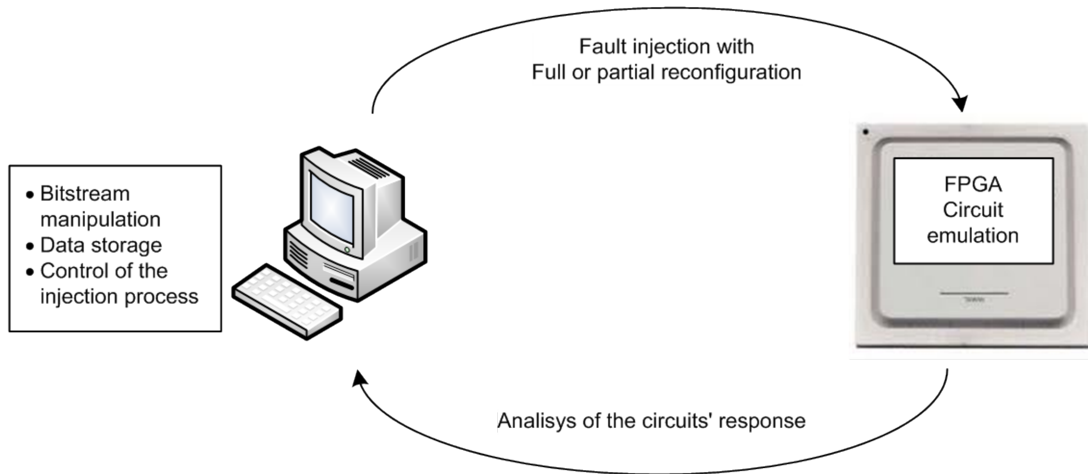


Figure 30: Fault-emulation procedure

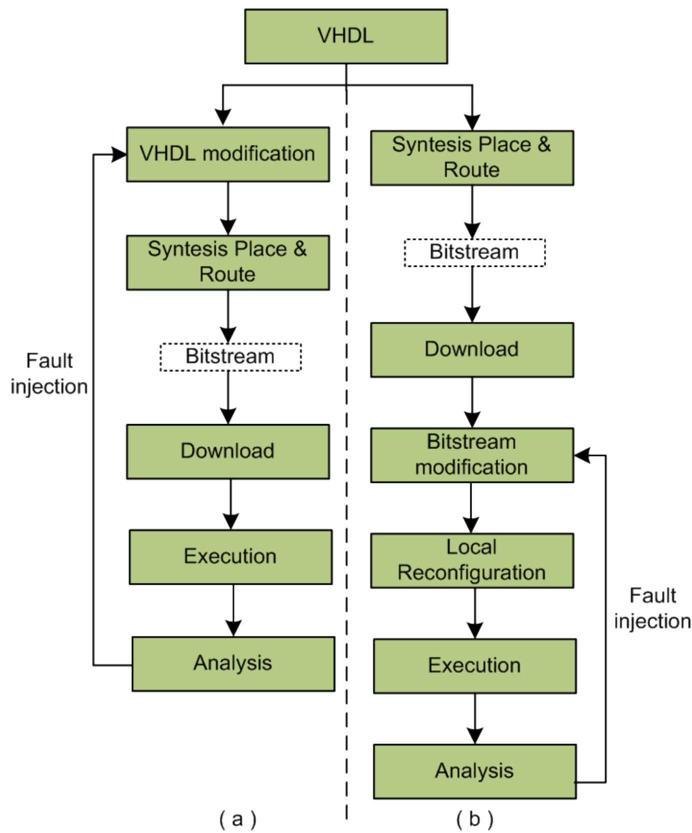


Figure 31: Fault-emulation design flow: a) VHDL modification emulation flow and b) Bitstream emulation flow

More advanced approaches are bitstream based and use a partial runtime reconfiguration to inject faults (Leveugle, 2001; Antoni, 2002; Leveugle and Antoni, 2003; Kafka and Novak, 2006; Sterpone and Violante, 2007). The design flow of this method is shown in Figure 31b. The fault injection is performed directly in the bitstream obtained after the synthesis, placement, and routing of the initial circuit and not by VHDL description modifications. While the bitstream modification must be made for each injected fault, the synthesis, placement and routing are performed only once. Since these design steps can be very time consuming, avoiding their repetition reduces the fault-injection time greatly. For each injected fault configuration, the bitstream has to be

modified and downloaded to the FPGA emulator, using the runtime reconfiguration. This may be done by a global reconfiguration of the device. However, in practice, only a small number of bits have to be changed in a bitstream to inject a fault; therefore, we can use the partial reconfiguration capabilities of the device.

The bitstream-based approaches differ in terms of which controller is used for the reconfiguration. The approaches of (Leveugle, 2001; Antoni, 2002; Leveugle and Antoni, 2003) use an external computer, while the approaches of (Kafka and Novak, 2006; Sterpone and Violante, 2007) use an internal embedded microprocessor.

The design of this technique requires an extensive knowledge of the FPGA bitstream structure. The information about devices' bitstreams are not provided by the FPGA manufacturers due to their protection of intellectual property. For Xilinx devices a JBits tool was provided for bitstream manipulation. The JBits application programming interface (API) is a Java-based tool set that allows designers to reconfigure a particular target in the Xilinx FPGA through the Select Map interface without any knowledge of the bitstream structure. The JBits software supports only Xilinx Virtex 2 and older devices; therefore, for newer FPGAs a different approach is necessary.

5.4 Our fault-emulation approach

To verify our on-line testing and recovery techniques we developed our own fault-emulation tool. For the systems on FPGA circuits the fastest and the most precise fault-injection technique is a hardware fault emulation. But this technique is also very hard to design. Every device-under-test (DUT) has to be analyzed separately and the faults have to be inserted into the low-level resources of the FPGA. These fault sources are very hard to determine without an expert knowledge of the FPGA structure. To make the design of fault-emulation experiment easier our tool can automatically determine the fault sources of the particular DUT.

The fault-emulation tool was developed on a Virtex 4 FPGA (also applicable for Virtex 5 and Virtex 6 FPGA with minor modifications). Our fault-emulation approach automatically extracts fault sources from the DUT and injects the faults using a partial reconfiguration of the FPGA (without the use of JBits tool) (Legat et al., 2009; Legat et al., 2010).

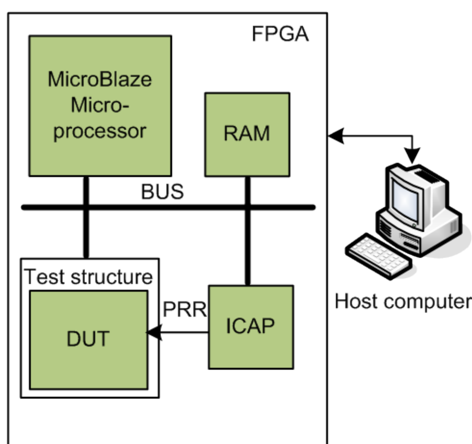


Figure 32: Basic hardware architecture of the proposed approach

An embedded microprocessor in the FPGA controls the fault-emulation procedure. The fault injection is performed using partial reconfiguration of FPGA through the internal-configuration-access-port (ICAP). The architecture of the proposed approach is

depicted in Figure 32. The host computer is used to automatically extract fault sources of the device-under-test (DUT) from the FPGA design. The fault sources are then sent to the embedded microprocessor on the FPGA. Fault injection and fault detection are performed by the embedded microprocessor and the results are sent back to the host computer. The process of automatic fault injection and the detailed structure of the tool are described in Chapter 6.6

6 On-line testing techniques

On-line testing techniques are being applied to critical parts of integrated circuits to detect errors during normal operation. This low-latency fault detection is necessary to prevent the faults from propagating through the system and to trigger the error-recovery procedures. Apart from a low fault-detection latency there are other requirements for good on-line testing techniques. These requirements are a high fault coverage with a low hardware and timing overhead. On-line testing is a traditional research field with established general approaches. However, their implementation in practice satisfying the above requirements is not a trivial task, but rather subject to tough optimization problems and innovative solutions.

The problem addressed in this dissertation was to develop an on-line error-detection solution for a 32-bit advanced-encryption-standard (AES) core that can be used in safety-critical applications. This is the smallest AES core on a FPGA with the on-line error-detection (OED) reported in the literature. In contrast to other solutions that focus on the individual AES processes (i.e., encryption, decryption, and key expansion) we performed an on-line test of the complete AES. Furthermore, an efficient BIST method for the AES core using our fault-detection scheme is introduced. This implementation is compact and has a high data throughput, which really demonstrates its efficiency trade-off. This work was published in (Legat et al., 2011a)

6.1 Advanced Encryption Standard

Since 2001, when the AES was approved by the National Institute of Standards and Technology (NIST, 2001), the AES has been one of the most commonly used cryptographic algorithms. The AES is a block cipher, which means that it encrypts or decrypts one block of data at a time. It takes a fixed block of data on the input and transforms it into an output block of the same size. For the transformation, a second input called the secret key is required. Since the AES is a symmetric cipher the same key is required for both the decryption and the encryption processes. While the original Rjindael algorithm allows different block and secret-key lengths, NIST restricts the size of the AES data blocks to 128 bits and allows just three key lengths 128, 192 and 256 that correspond to three encryption strengths.

The AES algorithm operates on a 4 x 4 array of bytes called a state. At the beginning of the encryption, the cipher key is added to the state. Next, 10 round transformations are performed. A round transformation is composed of four basic operations:

- Sub-bytes (SB),
- Shift-rows (SR),
- Mix-columns (MC),
- Add round key (AK).

In the last round, the MC operation is omitted. The process is shown in Figure 33. The resulting encrypted 128-bit block is called the cipher text.

The decryption process is similar to the encryption process. The cipher-text is stored in the state. Next, the inverse operations *Inv. sub-bytes* (ISB), *Inv. shift-rows* (ISR), *Inv. mix-*

columns (IMC) and *Add round key* (AK) are repeatedly performed in the reverse order. After the last inverse round transformation, the content of the state is a plain-text.

The key expansion process can be performed prior to, or simultaneously with, the AES encryption or decryption. The round keys are determined using the following operations:

- Rot-word (RW),
- Sub-word (SW),
- Add round constant (AR).

The initial round key is equal to the encryption key. The next round keys are expanded from the current round key using the RW, SW and AR operations.

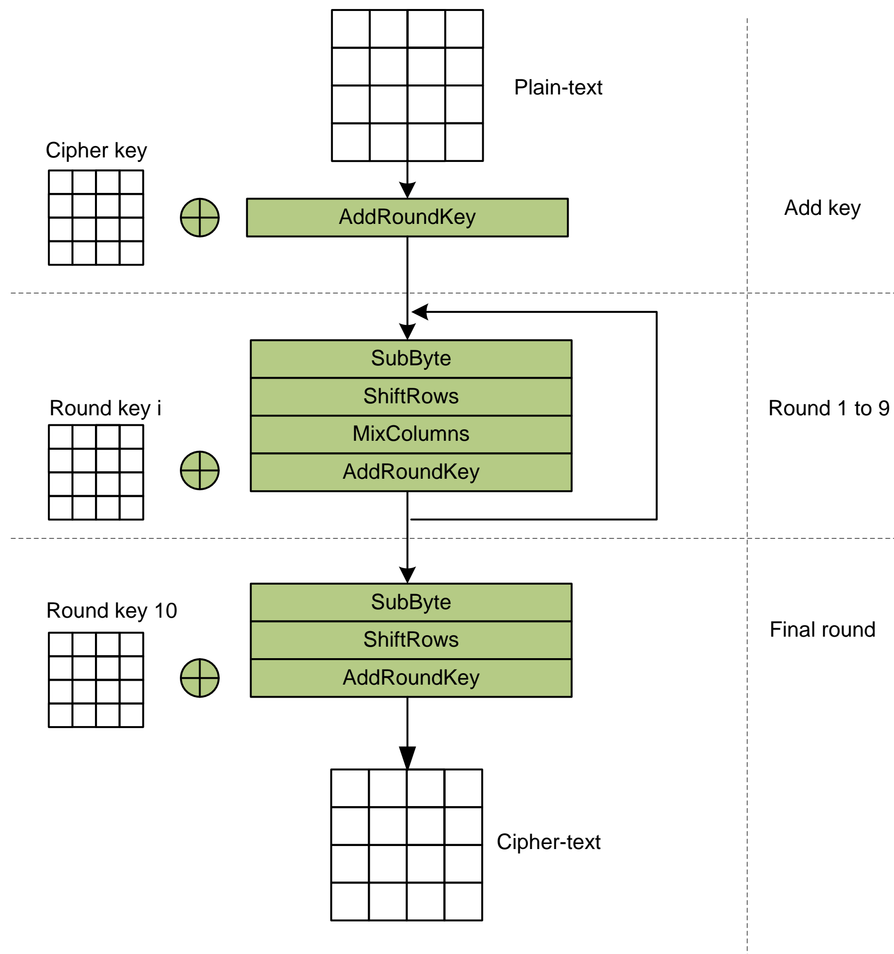


Figure 33: The data flow for the AES encryption algorithm

6.2 State of the art

Various hardware implementations of the AES algorithm were proposed; some of them were designed for ASIC and others exploit specific features of FPGA devices. The implementations also vary in terms of the length of the data word that they operate on. The most common are 128-bit implementations, which transform the whole AES state at a time. Pipelined implementations (Hodjat and Verbauwhede, 2004; Jing et al., 2007; Zhang and Parhi, 2004) achieve the highest data throughput, whereas 32-bit implementations (Gaj and Chodowiec, 2003; Rouvroy et al., 2004) use approximately four times fewer hardware resources than 128-bit implementations, but increase the encryption time as a result. The trade-off between resources and encryption time makes

such implementations suitable for small, cryptographic, embedded applications. In general, 8-bit implementations are the least hardware demanding (Feldhofer et al., 2004), but they achieve poor data throughputs.

Different OED schemes for various hardware implementations of the AES are currently being investigated. The employed schemes use one of the following: temporal, functional, or information redundancy. The temporal or time redundancy schemes execute encryption operations several times and compare the results (Karri et al, 2001; Karri et al, 2002; Maistri et al, 2008; Satoh et al, 2008).

These methods are only effective against temporal faults and they significantly reduce the data throughput. The functional redundancy schemes duplicate the various functional parts of the AES circuit and compare their outputs with the outputs of the corresponding parts in the operation (Di Natale et al., 2009; Joye et al., 2007). They have a larger hardware overhead due to the duplicated parts of the circuit and the extra control logic necessary to perform the comparisons.

The approaches that use information redundancy are usually based on error-detecting codes. The on-line testing schemes that use codes add extra bits of information to the original data. These codes are modified throughout the algorithm data path and their validity is checked at designated points.

The simplest of the error-detection codes are the low-cost parity codes. The parity prediction detects single and odd numbers of faults. The parity modification in different AES transformations was first studied in (Bertoni et al., 2002; Bertoni et al., 2003). The authors proposed parity prediction in different AES transformations in the encryption and decryption process. They added one bit per byte and implemented the S-box as a 512×9 bit memory. Bertoni et al. also analyzed the single- and multiple-fault coverage. They injected faults directly into the data block and studied the fault coverage of single and multiple errors. This is an abstract fault model and does not consider the different hardware-related fault sources. Another low-hardware-overhead solution of on-line error detection was presented in (Wu et al., 2004). It uses only one parity bit per 128-bit state and implements the S-box as a 256×9 bit memory. The solution was implemented in a FPGA to investigate the hardware overhead; however, no results in terms of fault coverage were presented. A similar AES parity-check solution is considered in (Ocheretnij et al., 2005)

The authors assessed the hardware overhead of different ASIC S-box implementations. They proposed a BIST method for the AES with concurrent checking. They injected stuck-at faults into the encryption data path and achieved a 76% fault coverage. Some authors focus on checking the S-boxes. The parity prediction depends on whether the S-box is implemented in the memory or as a combinational circuit. In (Bertoni et al., 2002; Bertoni et al., 2003; Wu et al., 2004) the parities were pre-calculated and stored in the memory. The parity-check schemes of the S-boxes implemented as a combinational circuit were investigated in (Mozaffari-Kermani and Reyhani-Masoleh, 2008; Mozaffari-Kermani and Reyhani-Masoleh, 2009; Shee-Yau and Huang-Ting, 2006). Truth tables are used in (Di Natale et al., 2007) to predict the input and output parities of the S-box.

More complex error-detection codes provide a better detection of multiple faults and some also offer single-fault correction, but occupy more hardware resources than simple parity prediction. The error detection based on CRC was presented in (Yen and Wu, 2006), while Hamming codes were investigated in (Mathew et al., 2008). A comparison between the Hamming and Reed-Solomon error-correcting codes was conducted in (Moratelli et al., 2008).

6.3 32-bit AES architecture

Our AES implementation is based on the architecture described in (Rouvroy et al., 2004). The AES round is broken up into a number of smaller, 32-bit-wide operations, which reduces the hardware cost by a factor of four, while it increases the encryption time by the same factor. Four clock cycles are required to complete each round transformation and consequently 44 clock cycles are needed to encrypt one block of data. This AES implementation uses specific hardware resources of the FPGA, notably the shift-register look-up tables (SRLs), the look-up tables (LUTs), and the dual-port RAM blocks.

In contrast to the conventional implementations, the order of operations for the round transformations during the encryption and decryption in the adopted architecture (Rouvroy et al., 2004) is modified as follows: SR, SB, MC and AK. The SB and SR operations are reordered compared to the AES standard. The reason for changing the order is that the SB and MC are combined into a single operation, which simplifies the implementation, as explained later in this chapter. The order of SB and SR can be changed because SB operates on a single byte, and the SR operation reorders the bytes without altering them.

The SR operation is performed by SRL using a dynamic variable access. The results from the AK operation in the previous round are shifted into the four 8-bit-wide shift registers in four consecutive cycles to store the whole 128-bit state into the shift register. The appropriate state bytes are selected from the shift register using dynamic variable access. For instance, in the first quarter of the round, bytes $s_0^i, s_5^i, s_{10}^i, s_{15}^i$ are selected. The SB, MC and AK operations are performed on selected bytes. The results, which represent the next state bytes $s_0^{i+1}, s_1^{i+1}, s_2^{i+1}, s_3^{i+1}$, are then stored in the shift register for the next round, as shown in Figure 34. The second, third and fourth quarters of the round are managed accordingly.

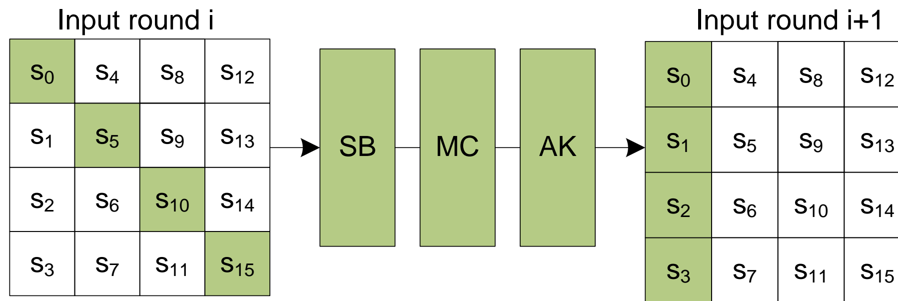


Figure 34: The first quarter of the round memory access

Following the definition of the SB and MC operation, the column of the next state is determined from a shifted column of the current state:

$$\begin{bmatrix} s_0^{i+1} \\ s_1^{i+1} \\ s_2^{i+1} \\ s_3^{i+1} \end{bmatrix} = \mathrm{T0}\left(s_0^i\right) \oplus \mathrm{T1}\left(s_5^i\right) \oplus \mathrm{T2}\left(s_{10}^i\right) \oplus \mathrm{T3}\left(s_{15}^i\right) \quad (3)$$

The SB and MC operations are combined and represented by the four tables T_0 to T_3 with 256 x 32-bit data:

$$T_0(s) = \begin{bmatrix} '02' * SB(s) \\ SB(s) \\ SB(s) \\ '03' * SB(s) \end{bmatrix}$$

$$T_1(s) = \begin{bmatrix} '03' * SB(s) \\ '02' * SB(s) \\ SB(s) \\ SB(s) \end{bmatrix}$$

$$T_2(s) = \begin{bmatrix} SB(s) \\ '03' * SB(s) \\ '02' * SB(s) \\ SB(s) \end{bmatrix} \quad (4)$$

$$T_3(s) = \begin{bmatrix} SB(s) \\ SB(s) \\ '03' * SB(s) \\ '02' * SB(s) \end{bmatrix}$$

where *s* is a state byte.

The process of decryption performs the inverse transformations ISR, ISB, IMC and AK. As in the encryption process, similar tables $IT_0(s)$ to $IT_3(s)$ are defined for the combined ISB and IMC operations.

The expanded keys are pre-computed and stored in a block RAM. The order of the RW and SW key expansion operations is changed. The SW is performed using the SB look-up table from the encryption round. Since the SB element of the $T_0(s)$ to $T_3(s)$ tables is duplicated, one SB element can be replaced with the ISB element, which is used in the SW operation for the inverse key expansion. The RW and AR key expansion operations are implemented using:

- a 32-bit shift register (SRL) to store the temporary key values,
- a Linear Feedback Shift Register (LFSR) to compute the round constants (RCONs),
- an appropriate signal wiring for the RW operation.

Tables CT_0 to CT_3 are defined to implement the SB, MC and SW operations used in the encryption, decryption and key-expansion process. These tables contain elements of the encryption T_0 to T_3, the decryption $IT_0(s)$ to $IT_3(s)$ and the key expansion SB, ISB. An example of CT_0 is shown below.

$$CT_0(s) = \begin{bmatrix} 2 * SB(s) & E * ISB(s) \\ SB(s) & 9 * ISB(s) \\ ISB(s) & D * ISB(s) \\ 3 * SB(s) & B * ISB(s) \end{bmatrix} \quad (5)$$

The CT_0 (s) is stored in the FPGA 18-Kbit block RAMs, which are configured as dual-port ROMs. It is easy to see from (2) that $CT_{1\text{-}3}$ are permutations of the table CT_0. Therefore, they do not need to be stored separately. They are acquired by shifting bytes of CT_0. This implementation of the CT tables is called the CT-box. Two dual-port ROM blocks are required to access the values of the CT-box from all the four bytes together (one CT-box uses 16 Kbits).

The complete 32-bit AES architecture from (Rouvroy et al., 2004) is depicted in Figure 35. Additional switching logic and multiplexers are required to select the encryption, decryption, and key expansion data path.

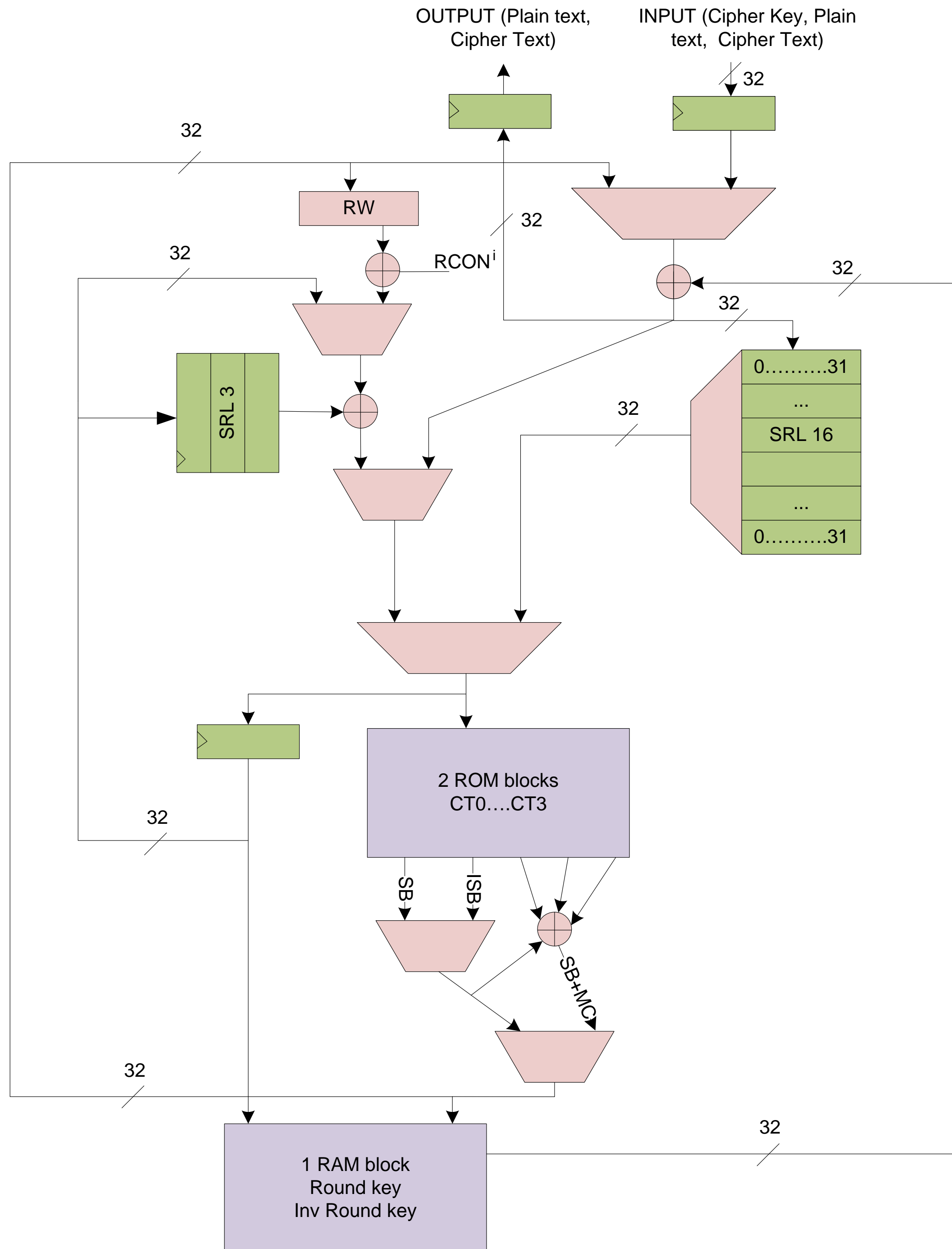


Figure 35: 32-bit AES architecture (Rouvroy et al., 2004)

6.4 The AES algorithm with on-line error detection

Our parity error-detection scheme is specially designed for the 32-bit FPGA implementation described in Section 6.3. It exploits the properties of the FPGA resources to minimize the hardware overhead. A parity check is performed in all the AES processes: encryption, decryption and key expansion. The AES implementation operates on a 32-bit word, which represents one state column at a time. A minimal hardware overhead would be achieved if only one parity bit per column was added.

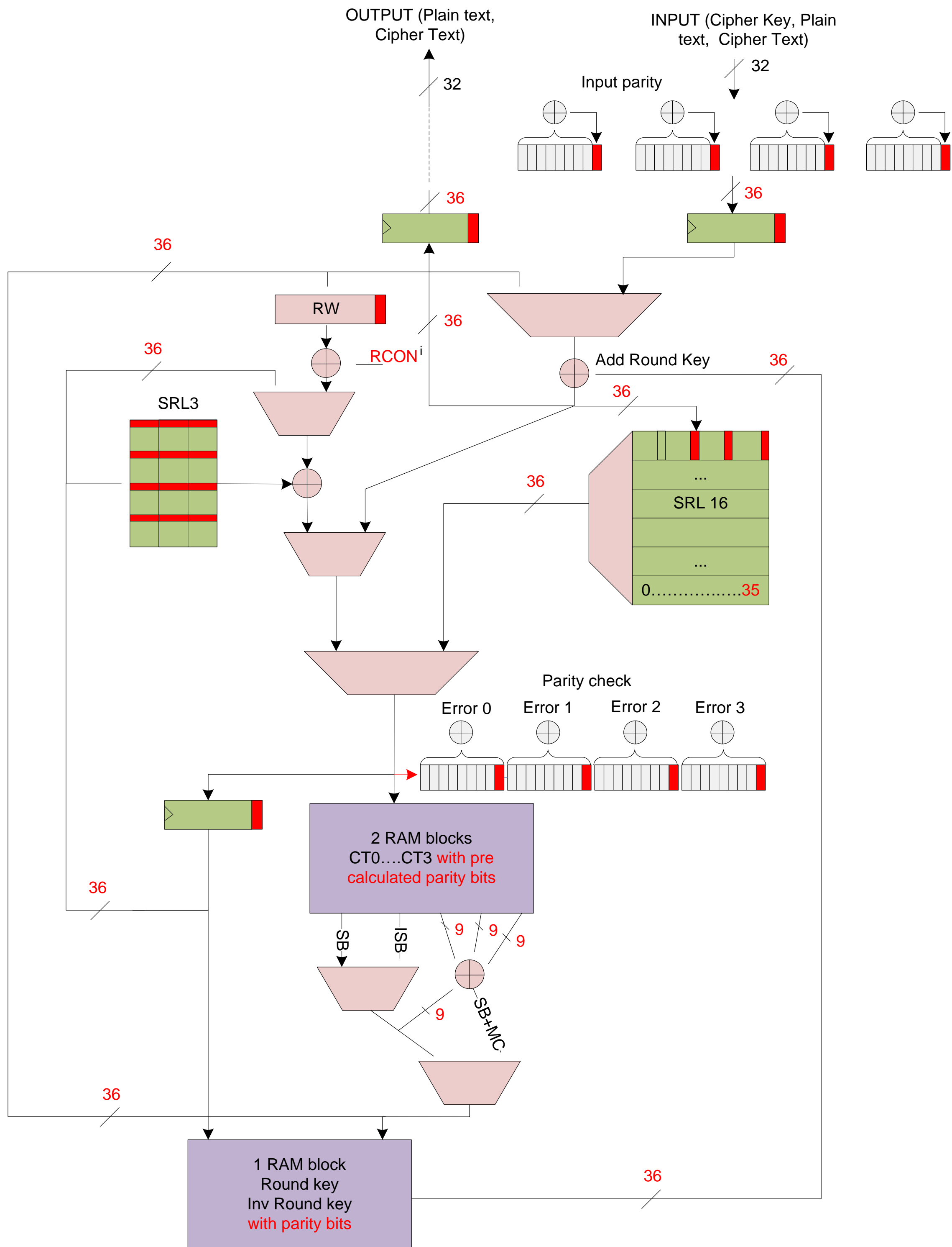


Figure 36: AES architecture with parity check

However, during the SR operation one byte is taken from each column of the state, as shown in Figure 34. This manipulation makes the computation of parity at the column level rather complicated; hence we added one parity bit to each state byte. Additionally, one parity bit per byte increases the error-detection capability of multiple faults in different bytes, which was experimentally analyzed in (Bertoni et al., 2003).

The parities of the 32-bit input word are determined by exclusive-or (XOR) gates. One parity bit is computed for each byte of the 32-bit word, resulting in a 36-bit word. The

input parities are modified according to the AES operations and checked at the output.

Figure 36 illustrates the modifications of the original AES architecture. Additional parity bits are clearly marked. The parity modification of the AES operations in the encryption and decryption are as follows:

- Shift-rows (SR) – The parity bit calculation of the shift rows operation is straightforward. The parity bits are shifted along with their accompanying bytes and are not modified. The shift register is extended from 32-bit to 36-bit.
- Substitute-bytes (SB) and Mix Columns (MC) - The SB and MC operations are performed by an 8-bit input 32-bit output look-up table CT-box, which is implemented in the ROM. For every byte in the 32-bit × 512 CT-box ROM a parity bit is pre-computed and stored, resulting in a 36-bit-wide word. At the output of the CT-box ROM, the parity bits are XOR-ed in parallel with the CT-box bytes. To store a CT-box with the parity bits no additional hardware is required. The 36-bit × 512 CT-box occupies 18 Kbits of memory, which is exactly the size of one FPGA Block ROM, which is already assigned by the original AES architecture.
- Add round key (AK) - During the AK operation the word of the round key is XOR-ed with the state word. Because the parity is a linear operator the four parity bits of the state can be XOR-ed with the adjacent four parity bits of the stored round key.

The additional hardware required to implement the parity check of the key-expansion process is highlighted in Figure 36. Four parity bits are computed out of the 32-bit cipher key word using XOR trees, resulting in a 36-bit-wide word. The parities are modified according to the key-expansion operations and stored in the RAM block with the round keys. The parity modifications of the key-expansion operations are:

- Substitute word (SW) - The SW operation uses the SB and ISB parts of the CT-box. The SB and ISB parts contain pre-computed parity bits.
- Rotate word (RW) - During rotation of the word the parities are rotated together with the bytes.
- Add Round constant (AR) - The AR operation performs the XOR operation between the first key column, the rotated last key column and the round constant. To include parities in this operation, a formula to calculate the parity of the round constants has to be derived. According to (NIST, 2001), the round constant $RCON(i)$ is defined over $GF(2^n)$ using the following equation:

$$RCON(i) = x^{(i-1)} mod\ G, \quad (6)$$

where $G = x^8 + x^4 + x^3 + x + 1$

The $RCON(i)$ can be given as a polynomial of degree 7: $RCON(i) = \sum_{k=0}^{7} a_k \cdot x^k$, with $a^k \in \{0,1\}$. Equation (4) can be written recursively: $RCON(i+1) = x \cdot RCON(i)\ mod\ G$. This expression can be simplified to:

$$RCON(i+1) = a_7 \cdot G + x \cdot RCON(i) \quad (7)$$

Let us denote the parity of the current round constant as $p(RCON(i))$; then the parity of the next round constant $RCON(i+1)$ is:

$$p\big(RCON(i+1)\big) = a_7 + p\big(RCON(i)\big). \quad (8)$$

During the AR operation, the parities of the 2 key columns are XOR-ed with the parity

bits of the round constants. The parity is checked using the XOR operation over each data byte and its parity bit, as depicted in Figure 36. In this implementation of the CT-box the parities are not carried from the input to the output of the CT-box. The CT-box parities are pre-computed, appended to the original values, and stored in the ROM. To check the AES data path and to ensure the correct CT-box input bytes, a parity check has to be performed at the input of the CT-box lookup table.

6.4.1 Fault masking in the CT-box

The implementation of a CT-box in a dual-port ROM can limit the error-detection capability of the employed parity scheme. When a single fault occurs in a dual-port ROM the same fault can be accessed from both ports and can either become a double fault in different bytes or it can be masked out with the XOR operation. In the latter case the fault would not be detected by the parity scheme. An example of fault masking in the CT-box is shown in Figure 37.

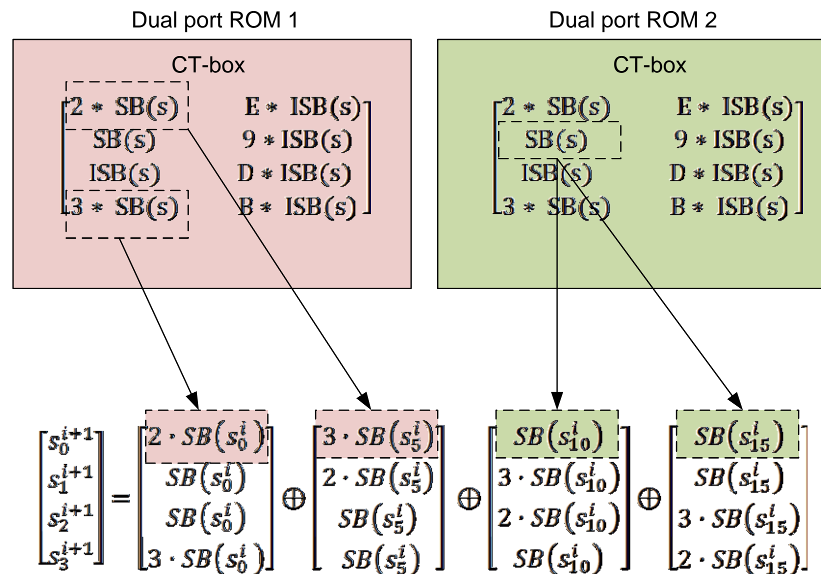


Figure 37: Masking of a single fault in a dual-port ROM

When we combine equations (3) and (4) we obtain the matrix equation in Figure 37 that determines the first column of the next state. From the given equation we can observe that elements $2 \cdot SB(s)$ and $3 \cdot SB(s)$ appear only once in the computation of the new column, while element $SB(s)$ appears twice. Therefore, a fault in either the $2 \cdot SB(s)$ or $3 \cdot SB(s)$ element results in an error of the new state byte and is detected. Note, however, that a single fault in the $2 \cdot SB(s)$ or $3 \cdot SB(s)$ element of a CT-box ROM might still result in several errors in a new column, but in separate bytes. On the other hand, a fault in the $SB(s)$ element could be masked out. This situation arises when both $SB(s)$ elements participating in the calculation of the next column byte reside in the same dual-port ROM. For instance, if we look at the first row of the equation in Figure 37 we can see that if $s_{10}^{i} = s_{15}^{i}$ and the part of the dual-port ROM 2 that corresponds to $SB(s_{10}^{i})$ is faulty, the error is cancelled out. This problem would not occur if there were two *SB* instances in the CT-box implementation, as is implied by the mix-column operation. However, a duplicate *SB* instance was replaced with the *ISB* element, which is needed for the expansion of the inverse key. Separate storage of the *ISB* elements would occupy

more FPGA resources. However, this problem could also be circumvented by using a separate ROM for each CT-box implementation, but on the other hand that would require four ROMs instead of only two.

From the equation in Figure 37 it is clear that the $SB(s)$element always appears in two consecutive CT-boxes, and thus if each consecutive CT-box is implemented by different dual-port ROMs the single fault in the ROM results in a single fault in the state byte. This solution is shown in Figure 38.

The fault-masking problem does not occur in the decryption. The elements $E \cdot SB(s)$, $9 \cdot SB(s)$, $D \cdot SB(s)$ and $B \cdot SB(s)$ are used to compute the next state column. A single fault in either of the elements results in a single error in one or several separate bytes of the new column, which is detected.

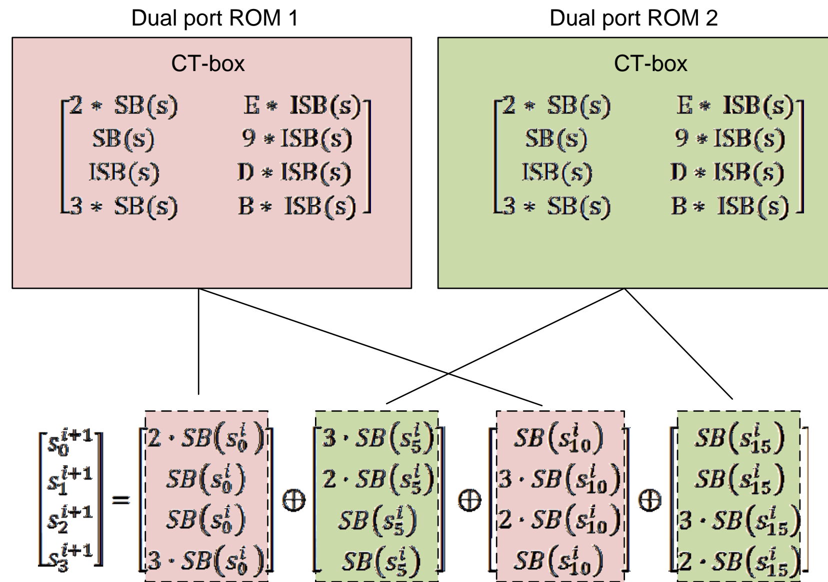


Figure 38: Correct implementation of CT-boxes to prevent fault masking

6.4.2 Hardware implementation

The AES with OED architecture was designed in VHDL. The VHDL code was synthesized using Xilinx ISE 11.5 software. To estimate the hardware overhead for different parts of the error-checking method, the synthesis of the plain AES, the AES with parity prediction without parity check, and the AES with a full error-detection mechanism were implemented. Table 2 summarizes the implementation details.

A substantial part of the hardware overhead is used for the input parity calculation and parity check. The AES with parity prediction occupies 11% more slices than the original AES core, while the slice overhead of the complete error-detection scheme with parity check is 16%. The throughput of the AES encryption is practically the same. For the implementation of the error detection no extra clocks are needed. The difference of 2.4% in the data throughput is due to the difference in the circuit's maximum clock frequency. The lower maximum clock frequency is the result of a slightly longer critical path. A significant advantage of this AES parity-check implementation is that we do not need any extra FPGA RAM blocks to implement the parities of the CT-box and the round keys since there is just enough space inside the already-used RAM blocks to store them. The complete AES core with the OED occupies 37% of the Xilinx Spartan 3 XC3S50 FPGA slices.

Table 2: Comparison of the implementation results

	Our 32-bit AES All AES processes			128-bit AES (Wu et al., 2004) Encryption only	
	AES	AES with parity	AES with OED	AES	AES with OED
	Spartan 3 XC3S50			Virtex XCV1000	
Slices	247	274	287	1815	1950
Flip Flops	73	81	81	823	830
4 input LUTs	339	383	403	3629	3899
BRAM blocks	3	3	3	8	9
Slice overhead	/	10.9%	16.2%	/	7.4%
BRAM overhead	/	0%	0%	/	11.1%
Clock frequency(MHz)	103.6	102.8	101.2	56.4	53.0
Throughput (Mbps)	301.4	299.1	294.4	656.2	616.7
Throughput reduction	/	0.77%	2.4%	/	6.4%
Through./Area (Mbps/Slices)	1.22	1.09	1.03	0.36	0.32

So far, to the best of our knowledge, no other 32-bit AES implementation in a FPGA with on-line error detection has been reported. For the reader's convenience, a 128-bit AES implementation (Wu et al., 2004) with similar functionality is included in Table 2. While a direct comparison of the two designs in terms of the employed resources is not really meaningful, the throughput versus area can be regarded as a realistic common criterion, and in this case our implementation outperforms the other by a factor of 3.

6.5 BIST method for the AES algorithm with error detection

The cryptographic algorithms exhibit a good statistical property for pseudorandom testing (Schubert and Anheier, 2000), (Di Natale, 2010). We used this property to make an efficient BIST with a low hardware overhead. The idea is to run the encryption, decryption, and expand the key processes with parity-error detection in a number of iterations. The output of the previous iteration is fed into the input of the next iteration. In this way the pseudorandom input test patterns are generated. The parity check is used as a test of the AES core. Therefore, a separate output analyzer is not required.

The faults in the AES can occur in the CT-box ROM or in the rest of the data path, comprised of multiplexes, logic elements, and shift registers. A fault can be detected with a parity check only when it is activated. In order to make the BIST efficient we analyzed which parts of the AES circuit are employed during a particular AES process. The detectable faults in the data path, apart from the CT-box, are activated in the few iterations of the BIST because the corresponding parts of the data path are used in each iteration. On the other hand, only a portion of the CT-box is used in each BIST iteration. We examined how the individual AES process uses a particular part of the CT-box and how many times it accesses it. Figure 39 denotes the parts of the CT box used by the individual AES processes. The analysis shows that:

- In the encryption process the $2 \cdot SB(s)$, $3 \cdot SB(s)$ and $SB(s)$ parts are accessed 16 times per round during the first 9 rounds. In the last round, only $SB(s)$ is accessed.
- In the decryption process the $E \cdot ISB(s)$, $9 \cdot ISB(s)$, $D \cdot ISB(s)$ and $B \cdot ISB(s)$ parts

are accessed 16 times during the first 9 rounds. In the last round, only $ISB(s)$ is accessed.

- In the expansion of the round keys the $SB(s)$ part is accessed 4 times, while expanding any of the 10 round keys. During the expansion of the inverted round keys the $ISB(s)$ is used 8 times, while expanding the inverted round keys 0 and 10. On the other hand, $E \cdot ISB(s), 9 \cdot ISB(s)$, $D \cdot ISB(s)$ and $B \cdot ISB(s)$ are used 4 times per round during the expansion of the inverted keys 1 to 9.

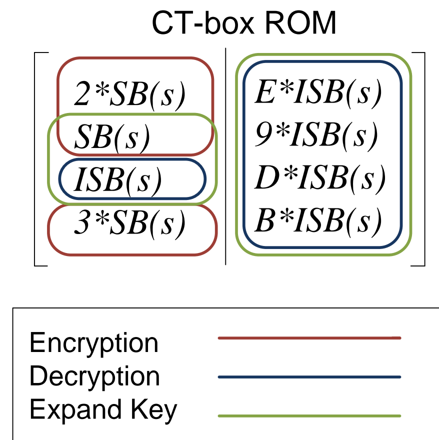


Figure 39: Parts of CT-box ROM accessed during the encryption, decryption and key schedule.

Table 3 summarizes the accesses to the particular part of the CT-box for each AES process.

Table 3: Frequency of the accessed CT-box ROM parts

	Encryption	Decryption	Expand Key
2*SB(s)	144	0	0
SB(s)	160	0	40
ISB(s)	0	16	8
3*SB(s)	144	0	0
E*ISB(s)	0	144	36
9*ISB(s)	0	144	36
D*ISB(s)	0	144	36
B*ISB(s)	0	144	36

According to Table 3, the $ISB(s)$ part is accessed only 24 times – 16 times during decryption and 8 times during key schedule process – which is considerably less than the other CT-box parts. We included an extra decryption and key expansion process to the BIST iteration to increase the probability of activating faults in the critical $ISB(s)$ part of the CT-box.

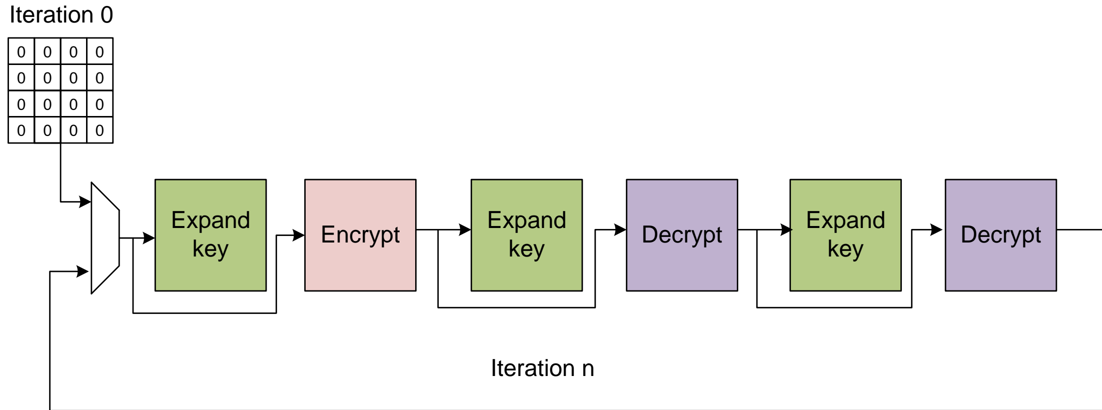


Figure 40: BIST data flow

The BIST data flow is shown in Figure 40. The BIST consists of one encryption, and two decryption AES processes, performed consecutively with interlacing key-expansion processes. Due to a good statistical property of the AES algorithm (Schubert and Anheier, 2000; Di Natale, 2010) the initial value of the BIST can be arbitrary. In our case it was set to zero to simplify the design. Since the key expansion process does not produce output values its input values are used as the input to the following encryption/decryption process. The output of the encryption/decryption process is fed into the next key expansion process. The output of the previous iteration is the input of the next iteration.

The implementation details of the AES BIST are shown in Figure 41. Apart from the AES core with error detection the BIST architecture comprises one FIFO, two multiplexers, and a simple control logic. The control logic initiates the BIST (*Init* signal MUX1) and schedules the AES process (Enc/Dec/Key signal). A four-stage FIFO is used to temporarily store the output vector of the previous AES process that is fed into the input of the next AES process. In the case of the key expansion process the AES core does not produce a relevant output so the FIFO feeds itself (controlled by the key signal). The control logic also stops the BIST at the detection of an *Error* or at the maximum number of iterations.

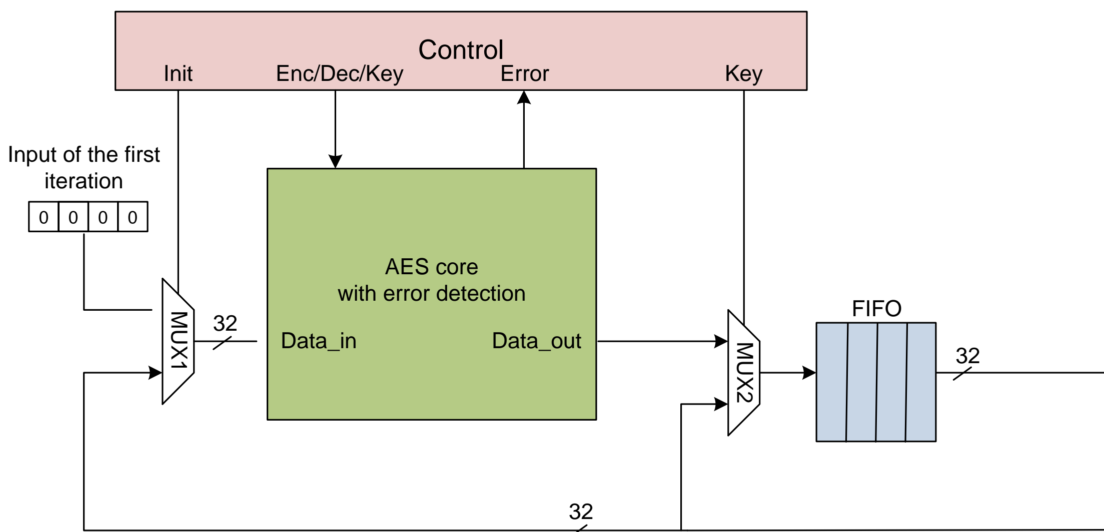


Figure 41: BIST architecture

6.5.1 Hardware implementation results of the BIST

The additional hardware required to implement the AES BIST's control logic and registers is presented in Table 4.

Table 4: Implementation of the proposed BIST on the Xilinx Spartan 3 XC3S50-4

	BIST hardware
Slices	80
Flip Flops	16
4 input LUTs	81 + 32 as shift registers
RAM blocks	0

The AES BIST is small enough to fit into the smallest of FPGAs. The complete implementation of the AES including the BIST occupies 47% of the low-cost Xilinx Spartan 3 XC3S50 FPGA resources.

6.6 Our fault-emulation tool

To evaluate the fault coverage and fault-detection latency of our on-line error-detection techniques we designed a fault-emulation tool (Legat et al., 2009; Legat et al., 2010). The detailed hardware-software architecture of our fault-emulation tool is shown in Figure 42. The list of fault sources is generated from the low-level HDL description. It is transferred to the embedded system memory through an rs-232 serial interface. The MicroBlaze embedded processor analyzes the list of fault sources and controls the fault-emulation process. The faults are injected into the DUT (AES core) by partial reconfiguration through the Internal Configuration Access Port (ICAP). A custom control core is used to start, pause, and stop the test procedure.

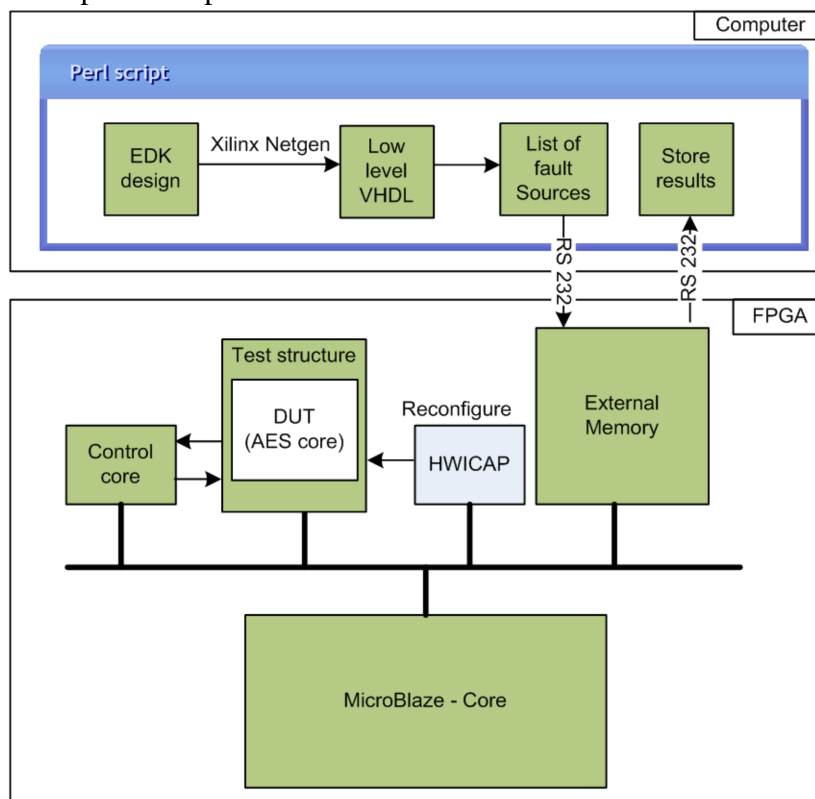


Figure 42: Hardware-software architecture of the fault-emulation tool with partial reconfiguration

6.6.1 Determining the fault sources inside the DUT

To use our method, the HDL description of DUT, Xilinx ISE, and EDK tools are required. A processor system with a DUT test structure core and an ICAP core is designed in the EDK. The list of fault sources is extracted from the DUT module using automated PERL script that performs the following tasks:

- The placed and routed processor design including the DUT is translated to the HDL description of low-level FPGA functional blocks that are directly mapped to the FPGA. This translation is done using the Xilinx tool Netgen. The low-level HDL description consists of basic VHDL functional blocks describing FPGA resources such as RAMs, LUTs, LUT shift registers, multiplexers and flip-flops (Simprim library).
- The low-level HDL description of the design is parsed. The HDL functional blocks that are part of the DUT are extracted using the name of the top entity of the DUTs HDL description. Other similar methods usually need the Plan Ahead tool to place the DUTs core separately from the rest of the design, but this is not necessary for our method.
- A list of fault sources is generated from the extracted HDL functional blocks. The list includes FPGA resources, their locations on the FPGA device, which are the same as in Xilinx FPGA Editor, and initialization parameters. In the case of a LUT with fewer than 4 inputs some inputs of the slice LUT on the FPGA device are not used. This information is passed to the embedded microprocessor using the bit mask. This bit mask instructs the processor to inject faults that correspond to the used input bits. An example of a HDL functional block of a LUT is shown in Figure 43. In this example the input ADDR 3 is not used and the bit mask ‘0001’ is added to the fault list. This bit mask tells the embedded processor to inject faults only in the last 8-bits of the LUT truth table.
- The list is sent to the FPGA via a serial connection.

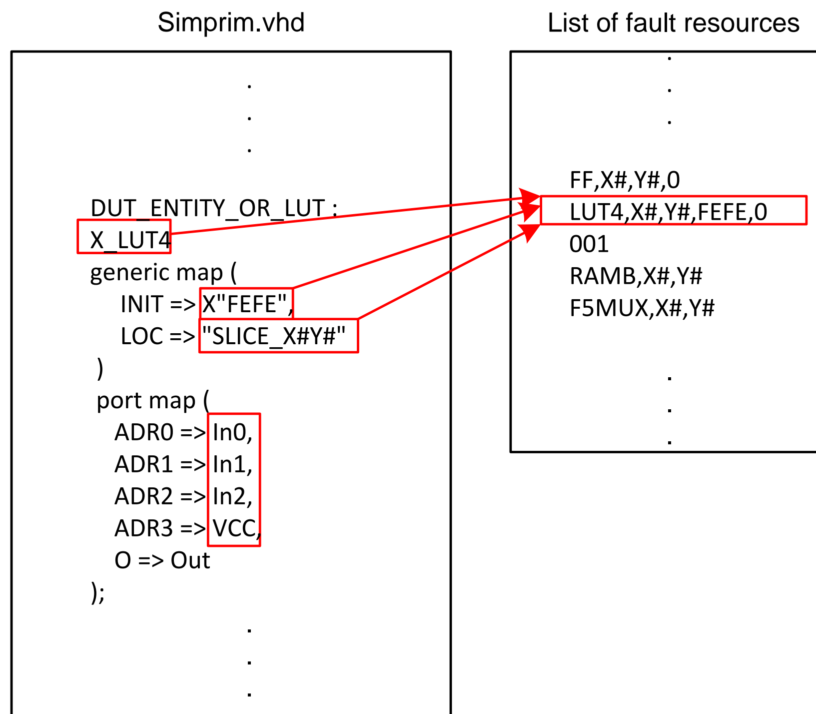


Figure 43: Extracting information from 4 input LUT functional blocks

6.6.2 Fault-injection flow using the HWICAP core on the Virtex 4 FPGA

Faults are injected by the MicroBlaze embedded microprocessor using the HWICAP (hardware ICAP) processor core. The reconfiguration was performed following the Xilinx configuration guide (Xilinx, 2009) and using some of the HWICAP functions. Information about the structure of the configuration memory can be found in Section 3.1.2 .

The greatest challenge we had to face during the fault-injection procedure was to find which bit inside the configuration frame corresponds to a particular part of the FPGA resource. The required information about the internal structure of the configuration frame for some FPGA resources had to be reverse engineered because it is not documented by Xilinx.

The fault-injection procedure starts by determining the frame address of the particular FPGA resource from its location in the list of fault sources. It continues by reading a frame, changing necessary bits inside the frame, and reconfiguring the frame. After the test of the DUT the frame is reconfigured to its original state. A detailed description of fault injection in particular FPGA resource is presented next.

6.6.2.1 Configurable logic block

The CLB frames can be read out or reconfigured using the Xilinx GetClbBits or SetClbBits C functions. The slice location X#,Y# from the fault list is also transformed by the provided functions. The CLB consists of 4 slices. A slice has two LUTs, two flip-flops and routing between the resources. A simplified slice structure with half of the elements is shown in Figure 44.

LUT: The configurable bits of a LUT are depicted in Figure 44. The INIT parameter holds the initial values of the LUT truth table. Other configurable parameters determine how a particular LUT instance is used as a shift register, dual port RAM, or LUT.
LUT bit-flip fault: The faults in the LUT can be injected before the start of the test procedure because the LUT is a part of the configuration and is not changed during the DUT's operation. The INIT parameter is read, a bit or multiple bits are flipped, the INIT is reconfigured, and the DUT is tested. After the test the INIT is reconfigured back to the original.

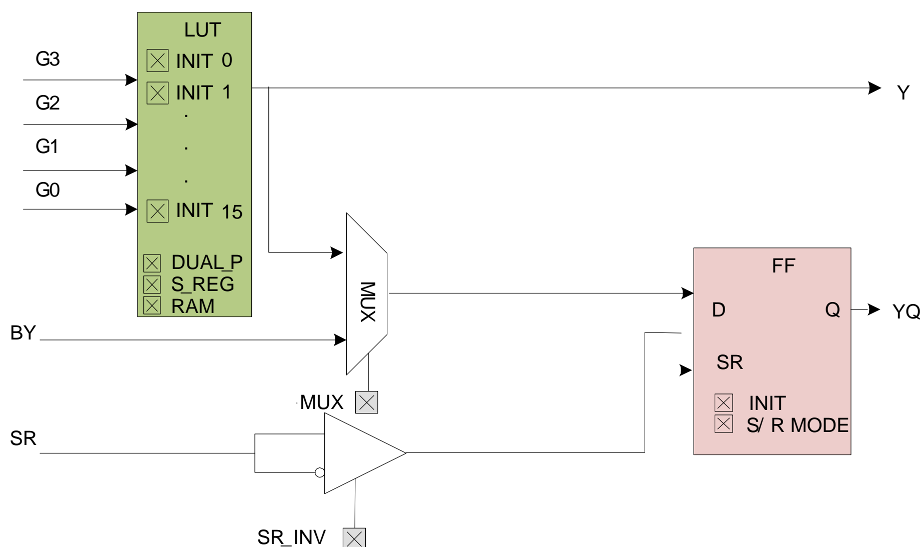


Figure 44: Simplified Virtex 4 slice structure (G side)

LUT stuck-at fault: Stuck at faults at the LUT inputs or output are emulated by properly reconfiguring the INIT parameter. Here are two examples. To emulate a stuck at 0 at the LUT output the INIT should be reconfigured to all 0s. Another illustrative example of the stuck at input G2 of a three-input LUT is shown in Figure 45. The injection procedure is the same as with the LUT bit-flip faults.

Fault-Free LUT

G2	G1	G0	Y
0	0	0	0
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	1

X"FE"

G2 stuck-at-0

G2	G1	G0	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1

X"EE"

Figure 45: Stuck at 0 fault on the G2 input

LUT shift register fault: LUT can be configured as a 16-bit shift register (SRL16). The fault injection has to be performed during the DUT's operation. The device is paused, the current content of the shift register is read, and reconfigured with the flipped bit. The DUT test procedure can then continue with the injected fault.
LUT RAM: LUT can be configured as a 16-bit dual-port RAM. In this case the faults are injected in the memory cells and the injection procedure is the same as with the SRL16.

Flip-Flop: Configurable bits of a flip-flop (FF) are shown in Figure 44. The INIT holds the initial state of the FF. The S/R MODE bit determines the value of FF on reset. The SR signal is a global FF reset which resets all the FFs. The FFs in a slice can be reset locally by inverting the SR_INV multiplexer. To capture the contents of the flip-flops into the configuration, or update the flip-flops with the content of the configuration, two complementary commands are issued:

- GCAPTURE loads the current value of the FF into its INIT configuration bit,
- GRESTORE updates the FF content with the value of the INIT configuration bit.

For more information of how these commands are issued refer to the Xilinx Virtex 4 configuration guide (Xilinx, 2009).

Flip-flop bit-flip fault: A bit-flip fault is injected during the operation of the DUT. The DUT is paused, the GCAPTURE updates the INIT of the FF. The INIT bit of the selected FF is then flipped and restored (GRESTORE). The test procedure continues with the injected fault.
Flip-flop stuck-at fault: Stuck-at fault can be injected when the tested device is paused. First, the S/R MODE bit is configured to the desired value to be stuck-at, and then the SR_INV bit multiplexer select bit is set to the opposite line. This holds the FF in the reset state and it is stuck-at the value of the S/R MODE bit. The limitation of this method is that both FFs of the slice are connected to the SR_INV multiplexer. Therefore, the stuck-at fault is injected in both FFs.

6.6.2.2 Block RAM

The faults are injected in the BRAM content using a partial reconfiguration. The reading and reconfiguring of the BRAM contents are performed by low-level DeviceReadFrame and DeviceWriteFrame C functions. The X#, Y# location of a particular BRAM instance from the fault list has to be translated to its frame address. The X# Y# location determines the major frame address that contains the contents of 4 BRAMs, and also selects the particular BRAM among them. The content of those 4 BRAMs spans over 64 frames identified by a minor frame address.

Notice that the BRAM content frame has special save bits that by default prevent changes of the content. In order to rewrite the BRAM frame content these bits have to be set to '0'. The offsets of these save bits in the Virtex 4 FPGA are (the ordering of frame words is from 0 to 41):

- top side of device: 136, 456, 808, 1128,
- bottom side of device: 184, 504, 856, 1176.

BRAM bit-flip fault: The fault is injected when the DUT is paused. The BRAM frame is read and saved to a buffer, bits are flipped, the proper save bits are set to '0', and the buffer is written back to the BRAM frame. After the fault emulation is completed the fault-free state is restored.

6.7 Fault-emulation results

The fault coverage of the proposed BIST architecture of the 32-bit AES core with the OED was assessed by our fault-emulation tool. The fault sources inside the AES design were automatically determined and a fault list was made. Faults were injected in the block RAMs and look-up tables. For each injected fault the BIST was run for several iterations. The fault-emulation results are presented in Table 5.

Table 5: Injected faults and fault coverage

	Injected Faults	Detected Faults	Fault coverage
BRAM (CT-box ROM)	36864	36864	100.0 %
BRAM (expanded keys)	3168	3168	100.0 %
LUT (AES combinational logic)	7124	6298	88.4 %

6.7.1 BRAM faults

Two RAM blocks are used to implement the CT-box that contains 1024 36-bit words. Thus the number of all the bit-flip memory-cell faults is 36,864. During each BIST iteration, some faults are detected, while others remain undetected. Figure 46 shows the percentage of detected BRAM faults during the first 100 iterations. We can see from the graph that the BIST detects most of the BRAM faults in the first 30 iterations. All the faults are detected in 83 BIST iterations.

The third RAM block contains the stored expanded keys. After the key-expansion process 3168 faults were injected into the memory cells containing the expanded keys and the inverse expanded keys. All the faults inserted into the keys are detected in a single BIST iteration. The first key-expansion process of the BIST iteration is followed by the encryption process, where all the expanded keys are detected and the second key expansion is followed by the decryption process, where all the inverse expanded keys are detected by the parity checker.

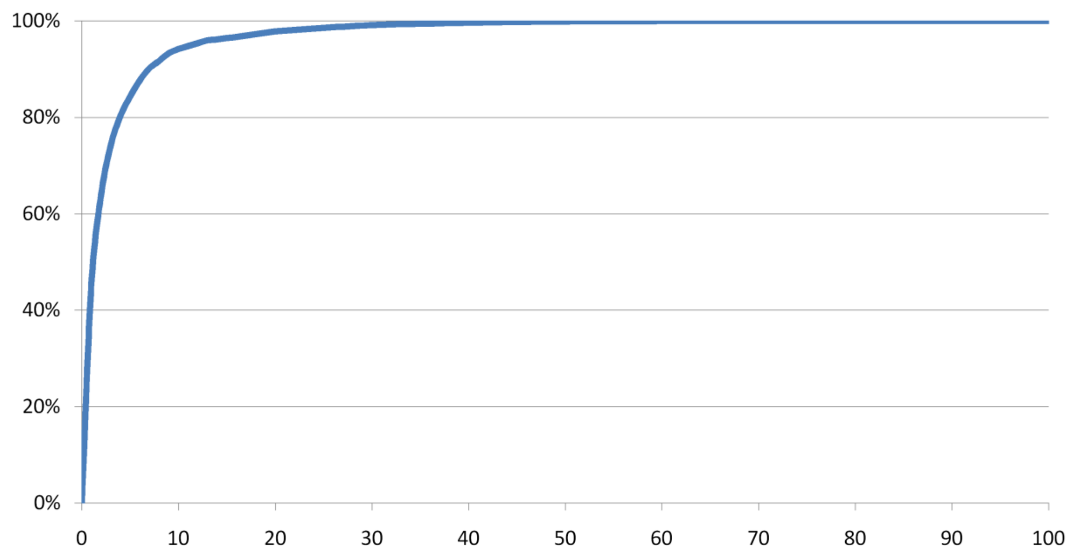


Figure 46: Fault coverage of CT-boxes versus the number of BIST iterations

6.7.2 LUT faults

The lookup tables contain the combinational part of the AES circuit. During an analysis of a low-level HDL description of the AES, 354 LUTs were found in the AES data path. A fault list was generated taking into account the LUT bit-flip faults and the LUT I/O faults. The fault list was reduced taking into account the fault collapsing of the equivalent faults. A total of 7124 LUT faults were injected into the AES core. Of those, 4164 were LUT bit-flip faults and 2960 were LUT I/O faults. The AES BIST detected 88.4% of the injected LUT faults. Figure 47 shows the fault coverage versus the BIST iterations. The experiment shows that the majority of the faults were detected in the first few BIST iterations.

The 826 faults that were not detected by the parity check also did not affect the output of the AES after 100 BIST iterations. The undetected faults in the LUTs were further examined:

- A logic function of the AES circuit implemented in the LUT might contain don't care values x. A flip of the bit associated with x does not affect the output of the AES and this injected fault is redundant.
- Faults were also injected into the parity-checker part of the circuit. Some faults in the parity checker cannot be detected because they are not activated by the AES BIST, for example, a stuck-at 0 at the output of the parity checker error signal cannot be detected, either by examining the output of the parity checker or by checking the AES output.

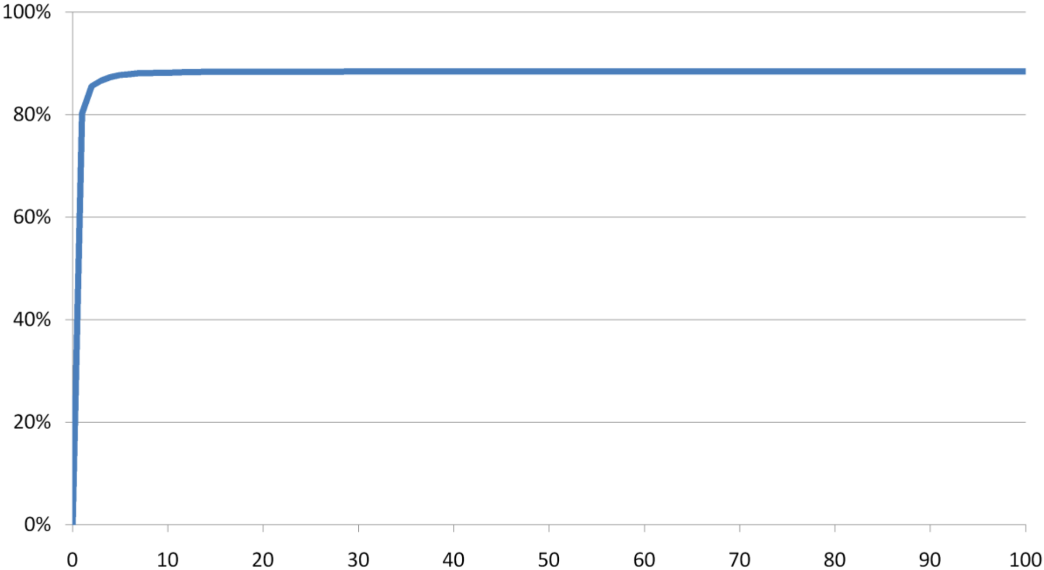


Figure 47: Fault coverage of the AES logic in the LUTs versus the number of BIST iterations

7 Error-recovery techniques

Mission-critical systems on SRAM-based FPGAs require means to quickly recover radiation-induced soft-errors from the configuration memory. In this chapter we describe our two on-line error-recovery techniques.

The first is a recovery technique for multiprocessor systems on reconfigurable chips (Legat et al., 2011c). This is the first error-recovery technique that exploits the advantage of multiple processor cores. In the case of the failure on one processor another processor is able to recover the system. With different test-scheduling options the technique offers a choice between the reliability and availability of the system, and it requires a minimal amount of extra hardware resources.

The second recovery technique is a small error-recovery mechanism that can be used to convert any system into a reliable self-reparable system (Legat et al., 2011b). The salient feature of our internal recovery mechanism is that it occupies the least reported hardware resources and is portable to different FPGAs. The reliability of the mechanism was tested in different recovery architectures, including implementation of the mechanism in the TMR and monitoring by an external watchdog timer.

7.1 Self-recovery of embedded multi-processor systems on the FPGA

This method can be applied to a multiprocessor system implemented on the FPGA. It is effective against soft errors in the configuration memory (i.e., the errors caused by high-energy radiation, also known as Single Event Upsets). The system tests itself without a visible downtime to the end user. The recovery algorithm checks the configuration memory of the SRAM-based FPGA through the internal-configuration-access-port (ICAP) and repairs the faulty configuration bits through a partial reconfiguration (Legat et al., 2011c).

In our work the internal recovery technique from (Heiner et al., 2008), (Chapman, 2010) is extended from single to multi-core systems on a chip. The recovery process is controlled by one of the processor cores, while others perform their normal operation. The advantage of our technique is that the error-recovery algorithm is able to adapt itself: when a fault causes erroneous behavior on the current testing processor it can be recovered by another working processor. Our error-recovery algorithm has also been implemented in a case study and evaluated by a fault-injection experiment (Legat et al., 2011b).

7.1.1 Required hardware platform

Our technique is designed for a multiprocessor SOC on FPGAs. The required hardware infrastructure is depictured in Figure 48. The hardware requirements are:

- a multi-core system with *n* embedded microprocessor cores,
- a local memory for each processor core,
- a partial reconfiguration interface that the processor cores have a common access to,

- a mechanism for mutual exclusion to enable one process to gain exclusive access to a particular shared peripheral,
- an external watchdog timer.

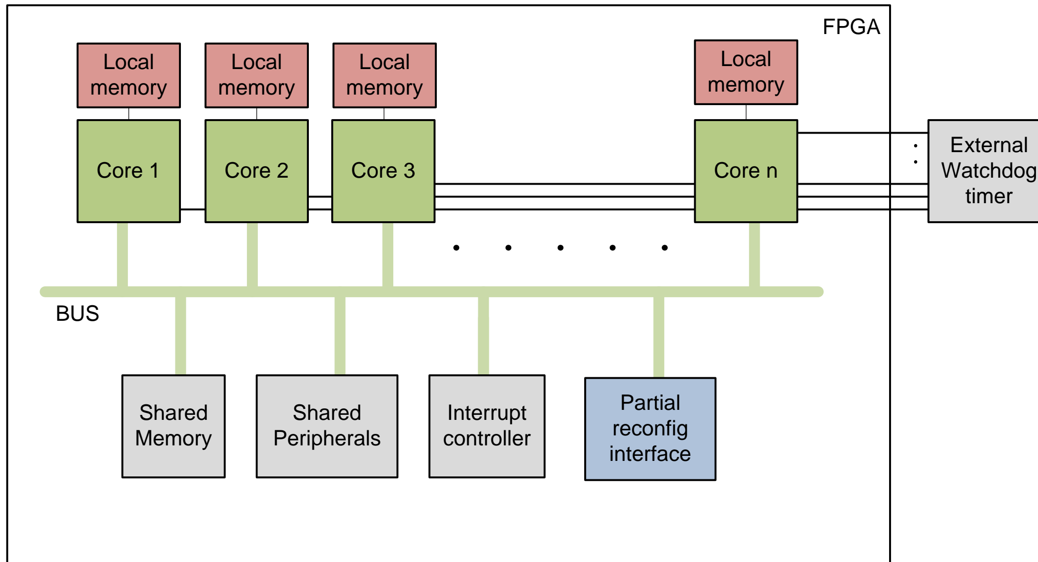


Figure 48: Required hardware architecture

7.1.2 Error-recovery technique

The basic procedure of our error-recovery technique is that one of the processor cores scans the configuration frames and performs a reconfiguration in the case of detected faults (first-pass recovery). If the processor core itself is affected by a SEU, another processor core takes the role and performs the reconfiguration (extended recovery).

During normal system operation the error-recovery algorithm runs in parallel with the target system application. The time intervals of the test execution depend on the target application since testing may slow down the overall system operation and may have an noticeable impact on the power consumption. As stated in the introduction, FPGA circuits are vulnerable to a SEU. A fault may occur in any configuration memory cell at any time. Consequently, a processor core which checks the configuration memory is also subject to SEU induced faults. In order to solve the problem, the processor core always performs a self-test after the configuration check. We assume that if the processor core that checks the configuration core is corrupted, it will never report the result of the self-test as fault-free. A processor self-test can be accomplished in a number of ways. In our implementation we followed the approach proposed in (Wegrzyn et al., 2009).

In the following we give more details about test scheduling, error detection, and error recovery.

7.1.2.1 Test scheduling

The test-scheduling algorithm can adjust the test period and select the processor core on which the test will run. A shorter test period results in shorter fault-detection latency, but decreases the available processor resources for the end user. The shortest latency is achieved if the test runs continuously.

Configuration memory check and recovery runs on one processor, while other processors perform operations of the target application. In our implementation, the

processor that currently performs the algorithm is selected in a round-robin fashion. Other test-scheduling techniques are possible at the operation system level.

7.1.2.2 Configuration check and recovery

Error detection and correction is performed through the partial reconfiguration interface, which in our case is the ICAP core. The ICAP core enables an embedded microprocessor to read and write the FPGA configuration memory in runtime. The processor core reads the configuration memory frame by frame and checks the integrity of the current configuration frame. From 12 ECC bits and 1300 data bits it calculates the syndrome value. Table 6 provides a decomposition of the syndrome value and its corresponding error status.

In Virtex 4 and 5 FPGA devices an error-correcting code (ECC) facility that automatically calculates the syndrome value is used.

Table 6: Syndrome value and the corresponding error status

Syndrome Bit 11	Syndrome Bit 10 to 0	Error status
S[11]=0	S[10:0]=0	No error
S[11]=1	S[10:0]≠0	Single bit error - S[10:0] is the location of the error
S[11]=1	S[10:0]=0	Single bit error in the last parity bit
S[11]=0	S[10:0]≠0	Double bit error, not correctable

7.1.2.3 Error-recovery algorithm

The process of configuration readback and error-correcting runs on the selected processor. The test algorithm scans through all the configuration frames. It calculates and examines the ECC syndrome value of a particular frame and in the case of a fault takes appropriate actions. The recovery algorithm pseudo code is shown in Algorithm 1.

A fault can manifest in different ways. Possible scenarios of the errors and their recovery are:

- An error occurs on non-selected processors, or other peripherals. The selected processor corrects the fault.
- A double error occurs on non-selected processors, or other peripherals. The fault can be detected but cannot be corrected. The entire system is reconfigured.
- An error stops the selected processor. After a time-out, the watchdog timer triggers the next processor which tries to correct the fault.
- The selected processor reports a faulty self-test result. The processor stops. After a time-out, the watchdog timer triggers the next processor, which tries to correct the fault.
- An error affects the clock routing. The watchdog timer detects the fault. The fault is uncorrectable. The entire system is reconfigured.
- An error affects the global signals and configuration registers of the FPGA. The watchdog timer detects the fault. The fault is uncorrectable. The entire system is reconfigured.

Algorithm 1: Error-recovery algorithm pseudo code

```
FOR each_frame DO
  read_frame(frame_address)
  calculate_syndrome()
  IF syndrome = single_error THEN
   locate_error()
   reconfigure_frame()
   report_single_error()
  ELSE IF syndrome = double_error THEN
   report_double_error()
   stop()
  ELSE
   next_frame()
  END IF
END FOR
IF self_test() = OK
   reset_watchdog()
   next_processor(processor number++)
ELSE
   report_selftest_fail()
   stop()
END IF
```

7.1.3 Practical application of the error-recovery method

The proposed error-recovery technique was studied on a dual MicroBlaze processor design. Both processors have access to external memory. Apart from sharing the external memory, the two processors have their own local memory. Each processor has an interrupt controller assigned to it to handle various interrupts. The processors have a common access to the XPS_hwicap core. This Xilinx core is used for internal access to the configuration. It enables readback of the configuration memory and the partial reconfiguration. The processors use a Xilinx XPS_Mutex core to synchronize the access to the shared peripherals. The design also has an external hardware watchdog timer connected to the processors through the fast-symplex-link (FSL). Figure 49 shows a block diagram of the dual-processor system.

We implemented the soft-core multiprocessor design on a Virtex 5 FPGA device. The hardware components were assembled, compiled, and downloaded to the board using the Xilinx EDK tool. The software was developed in the Xilinx SDK tool and debugged using a hardware debugger interface via an MDM core.

The test algorithm runs on one processor at a time and is triggered by interrupts. The interrupts trigger the test algorithm in two ways. In a normal way, the previous processor finishes the check and self-test correctly, and triggers the interrupt on the next processor. In an urgent way, the watchdog timer triggers the interrupt. The watchdog timer is reset by the selected processor at the end of the check if the self-test is successful. If a fault causes an error on the processor and it fails the self-check or stalls during the check, the watchdog timer is not reset in time and it triggers an interrupt on the next working processor, which tries to correct the fault.

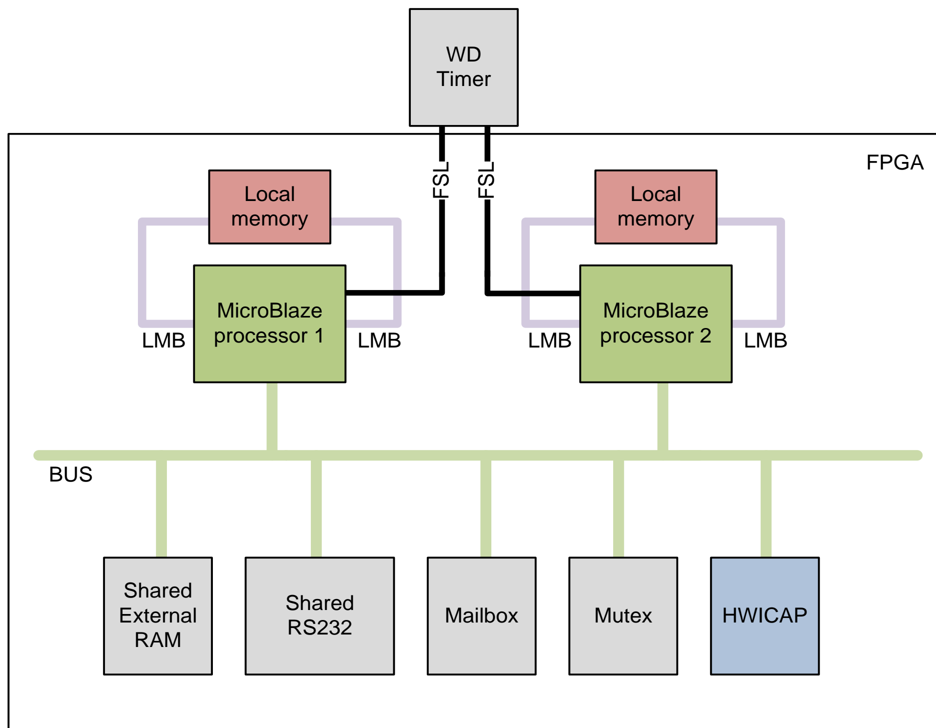


Figure 49: Block diagram of the dual-processor system

7.1.3.1 SEU fault-emulation experiment

A fault-emulation experiment was performed to assess the performance of the error-recovery technique. Fault emulation was assisted by an external computer. The faults were injected using a partial runtime reconfiguration through the JTAG configuration interface. The system on the FPGA was monitored using the RS-232 serial communication.

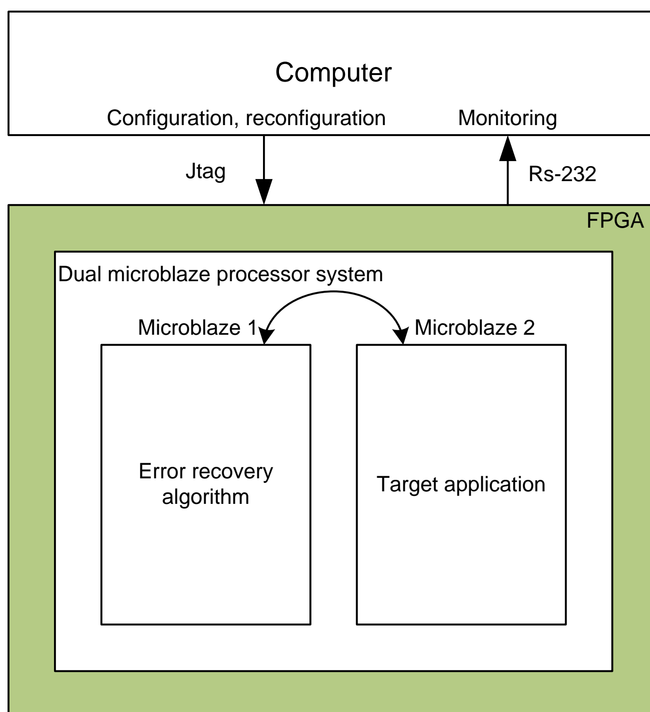


Figure 50: The structure of the fault-emulation system

For each injected fault, a system self-test was performed in order to determine whether the injected fault was located in the FPGA configuration memory used for the target application or not. In this particular case, a system self-test can be regarded as the actual target application. Next, for the same injected fault the proposed error detection and recovery algorithm was performed in a round-robin fashion. If the system on the FPGA had not recovered from the fault, the whole configuration was reconfigured before the next injected fault. The structure of the fault-emulation system is shown in Figure 50.

Random faults were injected into the system-on programmable-chip (SOPC) using an external computer. The time period between two injected faults was large enough for the system to finish the error recovery and test program on both processors.

7.1.3.2 Fault-emulation results

Table 7 presents the results of the fault-emulation experiment. A total of 5000 random single bit flip faults were injected into the system. 247 of them (i.e., 4.9 %) affected some critical location and caused a fault in the tested system. Some 188 out of 247 (i.e., 76 %) were detected and recovered by the error-recovery algorithm, while the remaining 59 caused the system to fail. From the 188 recovered faults, 61 faults caused a failure on the current testing processor and the system was recovered by the other working processor.

The obtained fault-emulation results were compared to other reported solutions. The techniques described in (Heiner et al., 2008), (Chapman, 2010) use the same principle of error detection and correction. The recovery controller is a single embedded microprocessor (PicoBlaze). The recovery method depends on the correct operation of the processor controller: if a fault causes the processor to fail the system cannot be recovered. Our technique has the advantage that it can adapt itself to another working processor and recover the system. As shown in Table 7, 247 faults affected the system under test. The system recovered from 127 out of 247 faults affecting the system by the first processor unit checking the FPGA configuration (the same techniques as employed in (Heiner et al., 2008), (Chapman, 2010)) and from an additional 61 faults after switching from a stalled (faulty) processor to another processor.

Table 7: Fault-emulation results

Scenario	Faults
All injected random SEU faults	5000
Faults that affected the system	247
System recovered First pass recovery Extended recovery	188 127 61
System did not recover	59

7.1.3.3 Reliability estimation

The reliability estimation of our dual-processor system is shown in Table 8. The hardware design of our system was implemented on Virtex 5 xc5vlx50t. According to the Rosetta experiment (Lesea et al., 2005) a Virtex 5 device has a nominal susceptibility to SEU of 151 FIT / Mb. The xc5vlx50t has approximately 11.37 Mb of relevant configuration cells. Therefore, this FPGA has a nominal susceptibility of 1717 FIT or a MTBF of approximately 66 years.

The dual-processor system occupies 5913 slices, which is 82% of the Virtex 5 xc5vlx50t slices. To assess the susceptibility of our system we made an analysis of the critical bits of the design with the Xilinx Critical bit report (Chapman, 2010) included in the Xilinx ISE tool. The tool estimated that 16% of the relevant FPGA configuration bits

were potentially critical to our design. Hence, the maximum susceptibility to a SEU of our design is 275 FIT.

Our fault-injection experiment indicated that only 4.9 % of the 5000 random faults affected the system. Hence, the critical bit count is smaller and the nominal susceptibility of our system without the error-recovery algorithm is around 84 FIT. The error-recovery algorithm corrected 76% of the faults that affected the system. Therefore, our SOPC with error detection has a nominal susceptibility to SEU faults of 20 FIT, which corresponds to 5707 years between two failures (MTBF).

Table 8: SEU reliability estimation of our dual-processor system

System	FIT (Faults/10^9h)	MTBF (Years)
Virtex 5 (xc5vlx50t)	1717	66
Our SOPC (Xilinx tool estimation)	275	415
Our SOPC (Experiment)	84	1358
Our SOPC with error recovery (Experiment)	20	5707

7.2 Hardware mechanism for the self-recovery of FPGA systems

We developed a small and fast internal error-recovery mechanism (Legat et al., 2011b). The mechanism is implemented by finite-state-machine (FSM) logic, which gives it an advantage against other reported solutions (Heiner et al., 2008; Chapman, 2010). The mechanism occupies the smallest portion of the FPGA configuration and has the fastest error recovery time. The efficient implementation of the error-recovery mechanism is very important, since a smaller design has a lower probability of the SEU affecting the correct operation of the mechanism.

7.2.1 Configuration check and recovery

The error detection and correction technique is similar to the technique described in Chapter 7.1.2.2 except that the configuration control is done with a small hardware mechanism and not by a multiprocessor program.

The configuration frames can be read (readback) or written (reconfigured) in runtime using the ICAP. Each configuration frame in the Virtex 4 and Virtex 5 FPGA contains 12 parity bits. These are the parity bits of the Hamming Error-Correction Code (ECC). The error within a frame is determined by a syndrome value, which is calculated from the 12 parity bits and the other 1300 bits of the read frame data. Table 6 provides a decomposition of the syndrome value and its corresponding error status. The first 11 bits of the syndrome value S[10:0] identify the location of a single erroneous bit within the frame (including the errors in the parity bits), while the last bit of the syndrome value S[11] indicates the double error in the frame. Virtex 4 and Virtex 5 FPGA have an embedded ECC circuit that calculates the syndrome value during each frame readback.

7.2.2 Implementation of error-recovery mechanism

The hardware architecture of our error-recovery mechanism is depictured in Figure 51. It consists of an ICAP device, frame ECC device, dual-port block RAM, and control logic.

The FPGA device is configured by writing the configuration commands into the configuration registers. The Xilinx Virtex 4, 5 Configuration User Guides give a detailed description of the register types and commands to perform the configuration operations.

The ICAP device has direct access to the configuration registers. The error-recovery mechanism uses the ICAP to read and write a configuration frame. The readback and reconfiguration operations are performed using an appropriate sequence of 32-bit configuration commands sent to the ICAP input. These commands are predefined and stored in the internal memory of the FPGA block RAM.

The frame ECC device is used to detect and locate errors inside the FPGA configuration frame. It works in parallel with the ICAP device. While the ICAP device reads the particular frame, the frame ECC uses the frame data to compute the syndrome value.

The internal block RAM is used to store the configuration commands and to buffer the frame data during the error correction. The block RAM is also susceptible to a SEU. To protect the integrity of the block RAM content in the Virtex 4 and Virtex 5 FPGAs, the block RAM ECC option is enabled. The block RAM can be configured as a single 512 × 64-bit RAM with Hamming error correction, using an extra eight bits in the 72-bit-wide RAM. The eight protection bits are generated during each read operation to correct any single error, or to detect any double error. The status output indicates the error status. In the case of a single error the logic does not correct the error in the memory array, but the output decoder only presents the corrected data on the output. The ECC in the Virtex 4

FPGA uses two vertically adjacent 18-kb block RAMs, while the Virtex 5 FPGA uses one 36-kb block RAM.

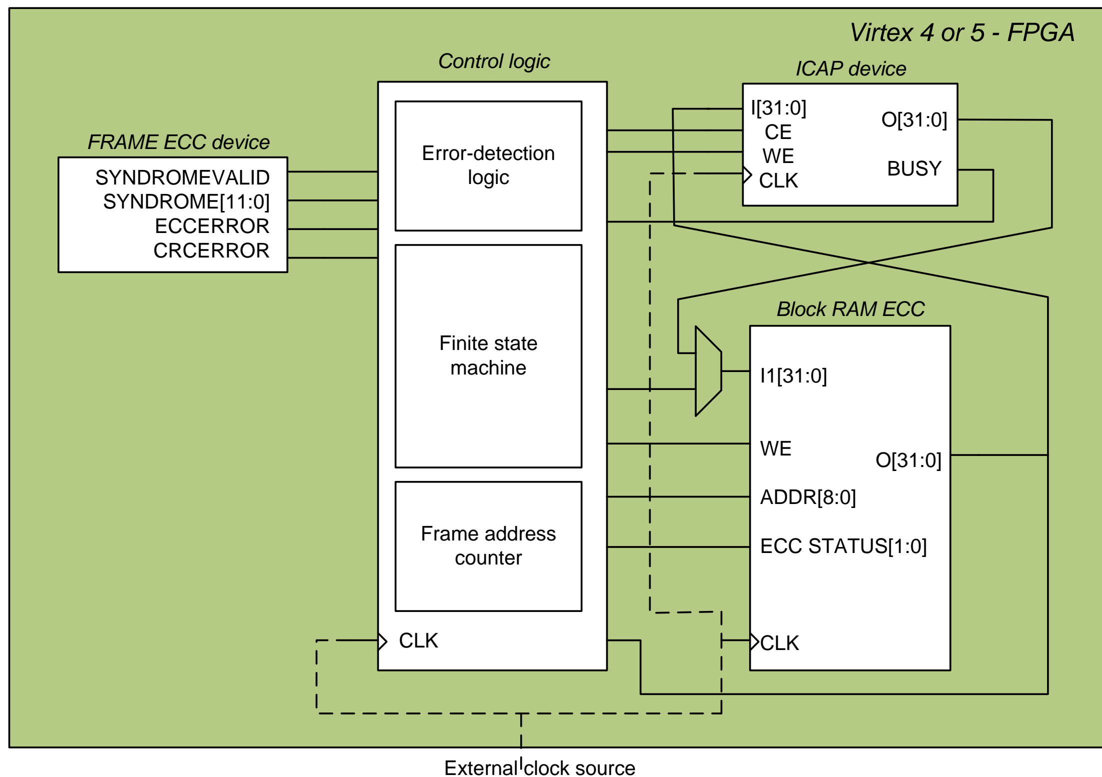


Figure 51: Hardware architecture of the error-recovery mechanism

The control logic manages the error detection-and-correction process. The controller is composed of a Finite State Machine (FSM) which uses Frame address counter, and an Error detection logic. The Frame address counter consists of separate counters for the major address, minor address, row, and top/bottom bit that form a 32-bit frame address. When the next frame address is required the counters are incremented. When the frame address reaches the last frame, the counters are reset back to their initial values pointing to the first frame. The Error-detection logic determines the location of a single error within the frame. It examines the syndrome value output from the ECC device and calculates the block RAM address and the bitmask used for the error recovery.

7.2.3 Operation of the internal recovery mechanism

The state machine of the internal recovery mechanism is shown in Figure 52. The recovery process begins with a *Start state* where the signals are initialized. Then the device configuration readback is started in the Initiate *readback state*. In the *Check frame state* the syndrome value of the current frame is checked through the frame ECC device by the Error-detection logic. The correct frame address is maintained by incrementing the Frame address counter for every pulse of the SYNDROMEVALID signal. Depending on the number of faults inside a single frame, the following scenarios are possible:

- In the case of a single fault the single error is detected and the device readback is stopped. The erroneous frame is read and stored in the block RAM (*Read frame state*). The faulty bit determined by the Error-detection logic is corrected and the frame is reconfigured (*Correct frame state*). After the reconfiguration the frame is rechecked in the *Check frame state* and if the error has been recovered the mechanism

restarts the device readback (*Initiate readback state*).

- In the case of a double fault the double error is detected. The mechanism switches to the *Stop state*. The recovery process is stopped and the double error, with its corresponding frame address, is reported.
- If more than two faults occur in one configuration frame, the operation of the mechanism becomes unreliable. It may happen that the faults are undetected (fault masking) or wrongly identified and corrected as a single fault. Empirical evidence (Lesea et al., 2005), however, suggests that the probability of a multiple fault in a frame is extremely low and can be neglected.

When the last frame is read, the mechanism goes back to the *Initiate readback state* and restarts the device readback.

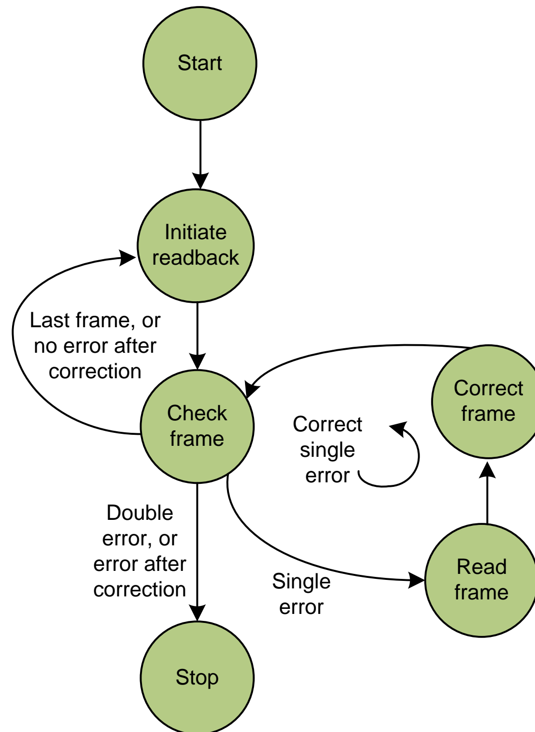


Figure 52: Simplified FSM of Virtex 5 recovery controller

7.2.4 Error-recovery-time comparison

The error-recovery time is an important feature of the error-recovery mechanism. A shorter recovery time means a lower probability that a SEU-induced fault will affect the operation of the target application. The error-recovery time is the sum of the error-detection time and the error-correction time. The worst-case error-detection time is the period in which the mechanism checks all the configuration frames and its time depends on the size of the FPGA device. Our error-recovery mechanism first initiates the configuration readback and then checks each frame in 41 clock cycles, which is one 32-bit word per clock cycle. When a single error occurs, our mechanism determines the erroneous frame, corrects the fault, and reconfigures it in 210 clock cycles.

Table 9 shows a comparison of the error-recovery times with other reported implementations.

The recovery mechanism for Virtex 4 was compared with the report of (Heiner et al., 2008). The error detection is performed by a PicoBlaze processor through a device readback. Their mechanism achieves a comparable error-detection time, but when the error is detected they do not stop the readback and they do not have the information about

the frame address of the erroneous frame. Hence, they have to perform another check on a frame-by-frame basis, which results in a very long error-correction time.

Virtex 5 devices have an autonomous dedicated circuit (readback CRC) that performs a continuous readback of the device. The recovery mechanism for Virtex 5 (Chapman, 2010) observes the readback CRC circuit to detect the errors. The error-detection time of the mechanism is the same as in our case; however, their correction time is longer (~12500 clock cycles). The recovery is controlled by an 8-bit PicoBlaze processor. The frame read and reconfiguration is four times slower due to the 8-bit processor bus.

Table 9: Error-recovery time comparison

Time is stated in clock cycles	Virtex 4 (XC4VLX15)		Virtex 5 (XC5VLX30)	
	Our Mechanism	Heiner et al. 2008	Our Mechanism	Chapman 2010
Worst-case Error-detection time	147650	~150000	226115	226115
Worst-case Error-correction time	210	~2400000	210	~12500
Worst-case Error-recovery time	147860	~2550000	226325	238615

7.2.5 Hardware-implementation comparison

The internal error-recovery mechanism was implemented in the Virtex 4 and Virtex 5 FPGAs. In Virtex 4 it occupies 156 slices, 118 flip-flop registers, and 1 block RAM. If the block RAMs are protected by an ECC, two block RAM instances are occupied. In the Virtex 5 the recovery mechanism occupies just 72 slices, 115 Flip-flop registers and 1 block RAM. A total of 36 kb of block RAM is used with the block RAM ECC. The implementation results can differ slightly when different options are selected in the synthesis tools.

Table 10 shows a comparison of the used hardware resources with two other reported scrubber implementations. Both of them use an embedded microprocessor to control the recovery process, resulting in significantly higher hardware overheads. The microprocessor indeed offers some additional options, like control or debug through an RS232 interface and fault injection. However, these options are not required for the actual recovery process and may even reduce the reliability of the system.

Table 10: Hardware implementation comparison

	Virtex 4		Virtex 5	
	Our Mechanism	Heiner et al. 2008	Our Mechanism	Chapman 2010
Slices	176	736	72	172
Flip-flops	118	680	115	321
Block RAM	2 (ECC)	2	1(ECC)	1

7.2.6 Implementation of the internal error-recovery mechanism in TMR

An internal error-recovery mechanism is also susceptible to SEUs. A critical fault in the mechanism could cause a system-wide corruption of the configuration data. Therefore, it is essential that the logic of the mechanism is protected by some SEU-mitigation technique.

The original implementation of the error-recovery mechanism is small and the block RAM is protected by an ECC. To further increase the reliability of the mechanism we implemented the mechanism in TMR. The hardware architecture of the TMR is depicted in Figure 53. The TMR is applied to the control logic and the block RAMs (with the ECC). The majority voter is placed at the inputs of the ICAP device. The outputs of the ICAP device and the Frame ECC device are fed back to the inputs of the triplicated design modules. These triplicated design modules are clocked by separate synchronous clock signals.

To apply the TMR technique effectively in the FPGA device additional restrictions had to be considered. The triplicated modules had to be placed in such a way that they were isolated from each other and the internal signals had to be carefully routed to limit the possibility that an upset would affect more than one module.

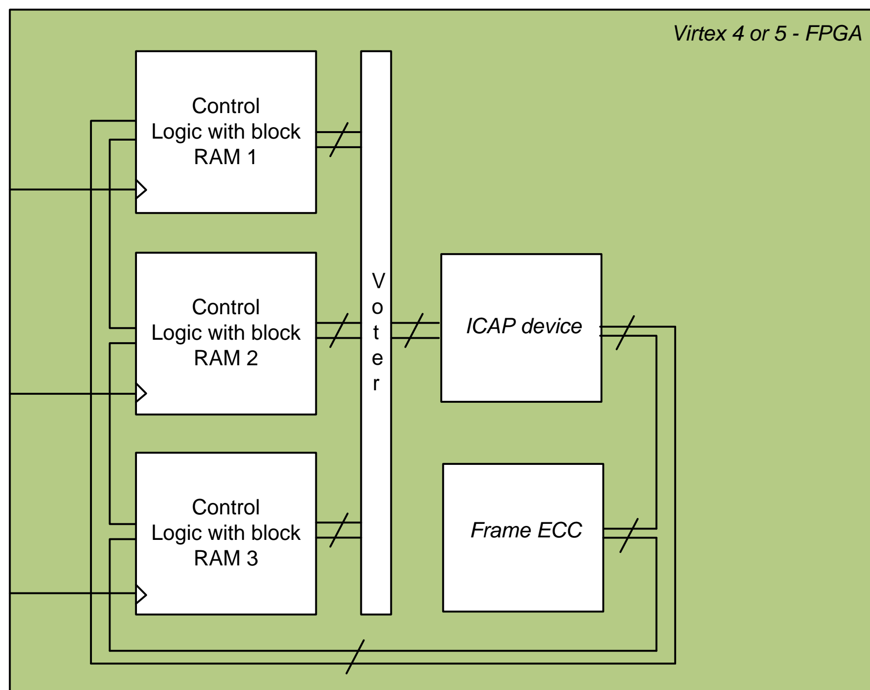


Figure 53: Hardware-architecture block diagram of error-recovery mechanism implemented in TMR

The hardware-implementation results of the TMR scrubber in Virtex 4 and Virtex 5 are shown in Table 11. The TMR increases the number of occupied resources by more than three times. (Heiner et al., 2008) also implemented the scrubber for Virtex 4 in the TMR. They triplicated the PicoBlaze controller and the block RAM. However, their hardware design is considerably larger.

Table 11: Hardware implementation of the error-recovery mechanism in TMR

	Virtex 4		Virtex 5
	Our Mechanism	Heiner et al. 2008	Our Mechanism
Slices	671	1308	321
Flip-flops	354	1082	345
Block RAM	6(ECC)	6	3(36 kb ECC)

7.2.7 Self-recovery architectures

The proposed error-recovery mechanism can be employed in different configurations of self-recovery architectures, with different internal and external hardware redundancies and different levels of reliability. We tested the following four architectures:

Recovery system with internal recovery mechanism: The most basic error-recovery architecture is shown in Figure 54 (A). It contains only the internal error-recovery mechanism, which runs at the same time as the target application. It corrects single faults and detects double faults in the configuration memory. This recovery architecture is suitable for systems with limited resources. The internal scrubber occupies less than 1 percent of the slices of the smallest Virtex 5 FPGA. This error-recovery architecture does not have any external recovery procedure. The system fails if an upset occurs on a critical bit of the internal error-recovery mechanism and it cannot recover itself from the SEFI.

Recovery system with internal recovery mechanism and external watchdog timer: Figure 54 (B) shows the block diagram of an error-recovery architecture with internal and external recovery procedures. The error-recovery architecture consists of:

- the FPGA, that contains the internal error-recovery mechanism and the user application, which run concurrently,
- the external watchdog timer, which monitors the vital signals of the internal error-recovery mechanism,
- the external non-volatile memory, which holds the original (golden) configuration data.

When the external watchdog timer detects the wrong operation of the internal error-recovery mechanism, it recovers the system from the external memory. The recovery from the external memory is also triggered by the double error signal from the error-recovery mechanism. This self-recovery architecture is effective against a SEFI. In order to provide an efficient operation of the watchdog timer the monitored signals of the internal error-recovery mechanism must be carefully selected. In some circumstances the failure of the internal error-recovery mechanism remains undetected and the system fails to recover. An experimental case study is given in the next section.

Recovery system with internal recovery mechanism in the TMR: The recovery architecture with an internal scrubber in the TMR is shown in Figure 54 (C). This system is vulnerable to the upsets occurring in the small voter circuit or to upsets that affect the operation of two modules at the same time. This architecture does not have any external recovery procedure and it is not effective against a SEFI. It is suitable for reliable systems that have enough available internal FPGA resources and are not able to include any additional external circuitry.

Recovery system with internal recovery mechanism in TMR and external watchdog timer: This architecture offers the highest level of reliability. The architecture combines the internal scrubber in the TMR with the external watchdog timer. Figure 54 (D) shows a block diagram of the architecture. The watchdog timer monitors the vital signals of the internal scrubber and recovers the system if it detects a failure. This system can be recovered from most of the failures, including the failures on the majority voter of the TMR scrubber and a SEFI.

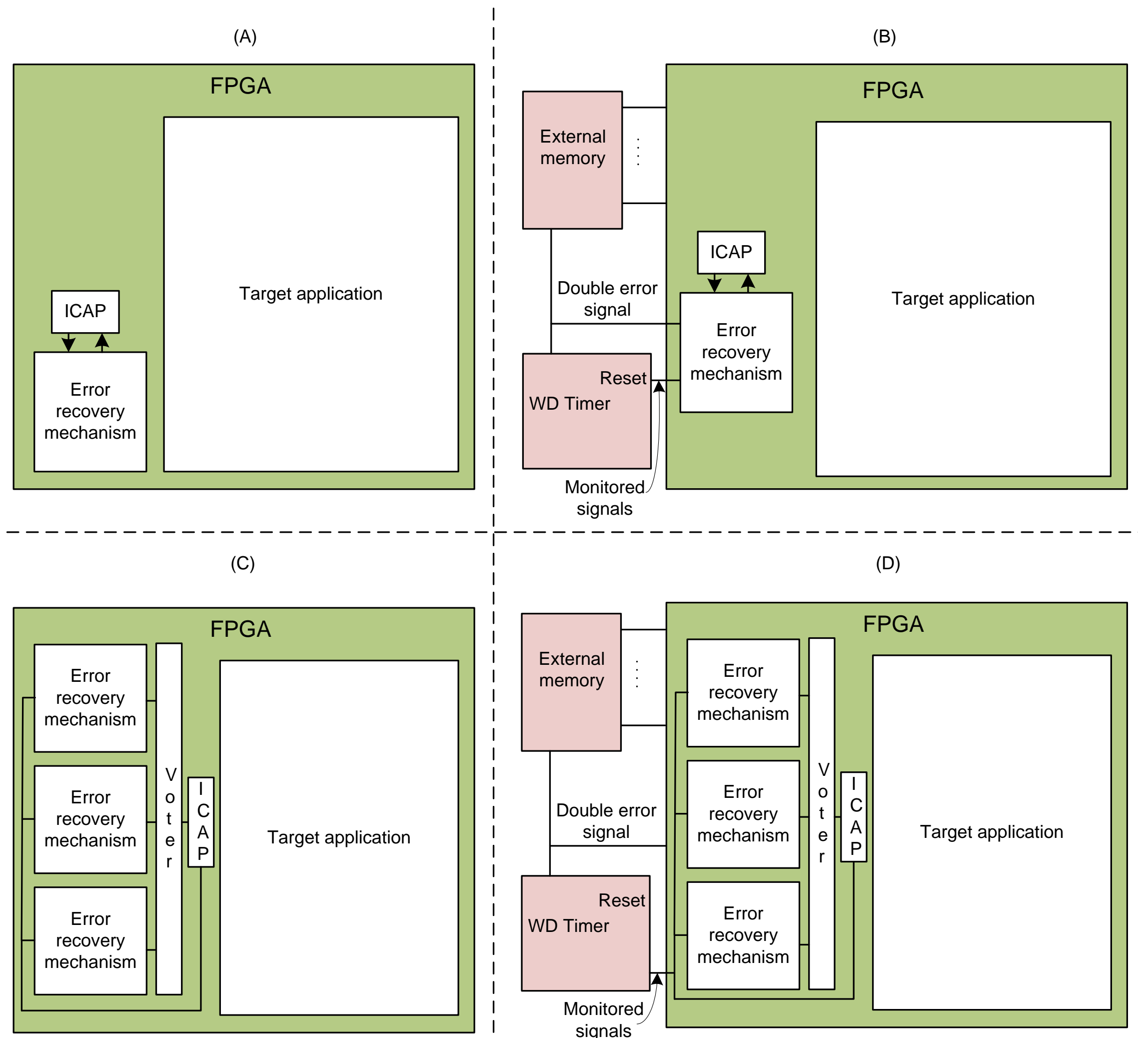


Figure 54: Self-recovery architectures

7.2.8 Fault-emulation experiment

In order to determine the reliability of the proposed recovery architectures we developed an environment for SEU emulation. The fault injection was performed by changing the logic state of a FPGA configuration bit using a runtime reconfiguration. Such a fault-emulation approach enabled us to inject faults at a precise location within the FPGA configuration memory.

The fault-emulation experiment was implemented using an internal fault-injection mechanism controlled by a computer. The hardware structure of the SEU emulation is shown in Figure 55. The fault-injection mechanism can inject a fault at a selected location within the configuration memory and is able to check if the injected fault has been corrected by the error-recovery mechanism. The computer serves as a fault-emulation controller, external memory, and reconfiguration device. It also acts as a watchdog timer for the error-recovery mechanism. During the fault-emulation experiment the error-recovery mechanism and the fault-injection mechanism are placed in the FPGA configuration separated from each other. The faults are injected only into the configuration frames of the error-recovery mechanism (regular or TMR version).

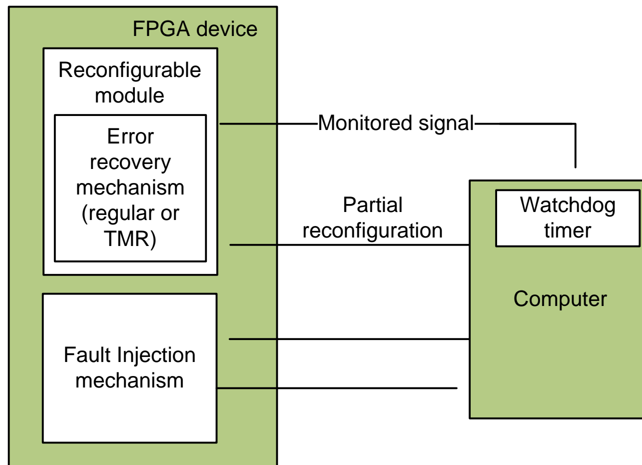


Figure 55: The structure of the fault-emulation experiment

The process of fault emulation begins by configuring the FPGA device with an error-free configuration. The error-recovery mechanism is stopped until the fault (i.e., the emulated SEU) is injected. For each selected fault the corresponding bit inside the error-recovery mechanism is corrupted. After the fault injection the error-recovery mechanism is started. It then runs until it scrubs the entire configuration memory once (one cycle). Depending on how the error-recovery mechanism was affected by the injected faults, the following situations may occur:

- The error-recovery mechanism operation is unaffected. In this case the error-recovery mechanism corrects the fault and completes the cycle, which resets the watchdog timer. The fault-injection mechanism confirms that the fault has been corrected. The computer logs the result and triggers the injection of the next fault.
- The erroneous operation of the error-recovery mechanism is detected by the watchdog timer. In this case the error-recovery mechanism has not completed the cycle and has not reset the watchdog timer. The computer logs the result, reconfigures the error-recovery mechanism from the stored partial configuration image, and triggers the injection of the next fault.
- The erroneous operation of the error-recovery mechanism is detected by the fault-injection mechanism. In this case the error-recovery mechanism completes the cycle and resets the watchdog timer, but does not correct the injected fault. The computer logs the result, reconfigures the error-recovery mechanism from the stored partial configuration image, and triggers the injection of the next fault.

Two separate fault-emulation experiments were performed to evaluate the reliability of the four proposed self-recovery architectures.

In the first experiment, 181056 faults were injected into the 138 configuration frames occupied by the normal implementation of the internal error-recovery mechanism to evaluate the architectures A and B. The results of the fault emulation are presented in Table 12. Of all the injected faults, 9177 of them affected the operation of the internal error-recovery mechanism. The faults that affected the internal error-recovery mechanism (denoted as critical faults) were further classified by their location. The majority of the faults (78%) occurred in the FPGA routing, 21% of the faults occurred inside CLBs, and 1% in the block RAM configuration.

Of these 9177 faults the watchdog timer detected 8458, while 719 (7.8%) faults remained undetected. We further examined the operation of the error-recovery mechanism under the influence of these undetected faults. The mechanism failed to

correct the injected fault, but did not further corrupt the configuration memory.

In the second experiment, 708480 faults were injected into the 540 configuration frames occupied by the TMR implementation of the internal error-recovery mechanism to evaluate the architectures C and D. The fault-emulation results are presented in Table 13. A wrong operation of the mechanism was detected in 554 cases. The majority of the critical faults (398) occurred in the FPGA routing. These faults were associated with the case where a single upset corrupted the signal routing of multiple modules inside the TMR or the routing inside the majority voter. The number of these faults could be reduced by applying a fault-tolerant routing algorithm, for example (Sterpone and Violante, 2008). All of the 69 critical faults that occurred in the LUT content corrupted the logic of the majority voter circuit. The 87 faults in the initial CLB configuration bits also caused the wrong behavior of the error-recovery mechanism. These faults alter the basic functionality of the resources in the CLB and can affect multiple modules of the TMR.

The watchdog timer detected the wrong behavior of the error-recovery mechanism in 90% cases.

Table 12: Fault-emulation results for the architectures A and B on Virtex 5 FPGA

All injected faults: 181056	Wrong operation of internal scrubber	Detected by watchdog timer	Undetected by watchdog timer
Critical faults	9177	8458	719
Routing	7180	6626	554
LUT content	1429	1307	122
CLB config	509	472	37
BRAM configuration	59	53	6

Table 13: Fault-emulation results for the architectures C and D on Virtex 5 FPGA

All injected faults: 708480	Wrong operation of TMR scrubber	Detected by watchdog timer	Undetected by watchdog timer
Critical faults	554	499	55
Routing	398	350	48
LUT content	69	62	7
CLB config	87	87	0
BRAM configuration	0	0	0

7.2.9 Reliability estimation

The reliability of Xilinx devices is being evaluated in an ongoing Rosetta experiment (Lesea, 2005). The current reliability estimation of the devices can be found in the Xilinx device reliability report (Xilinx, 2011). The SEU error rate is stated in terms of the Failures In Time (FIT) or the Mean Time Between Two Failures (MTBF). The FIT is the number of failures that can be expected in 109 hours of operation. Our experimental results are obtained from a Virtex 5 FPGA; therefore, we concentrated on the reliability of our self-recovery architectures implemented on this particular device. The FIT in the earth's atmosphere for Virtex 5 FPGA according to (Xilinx, 2011) is 151 FIT/Mb. The

reliability of a particular target design can be estimated based on the FIT of the FPGA device and the number of critical bits of the design.

Table 14 presents the reliability estimation for different design scenarios. The entire FPGA configuration has over 5000 FIT. However, a design uses only a certain number of configuration bits. We made an estimation of the reliability of an average design on the XC5VLX110T. According to (Chapman, 2010), an average design with a device utilization of 80% has about 11% of bits that are critical to its operation. The average design without error recovery has approximately 554 FIT.

The self-recovery architecture A has an internal error-recovery mechanism included in the target design. Only the bits that compromise the operation of the error-recovery mechanism are critical. The fault-emulation experiment indicated that 9177 bits in the configuration of the error-recovery mechanism are critical. The estimated reliability of our internal error-recovery mechanism is 1.49 FIT. In comparison, the Xilinx SEU controller reported in (Chapman, 2010) has an estimated 8.6 FIT and is less reliable. The self-recovery architecture B with the external watchdog timer also recovers from the majority of cases where the internal recovery fails. The number of critical bits is reduced to 719 and the reliability is increased to 0.12 FIT, which results in 1 million years of MTBF.

The self-reparable architecture C has an internal error-recovery mechanism implemented in the TMR. The failure of the internal TMR was detected in 554 injected faults, which results in an estimated 0.09 FIT. The external watchdog timer in the architecture D detected 90% of the failures and increased the reliability by ten times to 13 million years of MTBF.

Table 14: SEU reliability estimation

System on xc5vlx110t	Number of critical bits	FIT (SEU/10^9h)	MTBF (Years)
The whole FPGA device	31.1 Mb	5039.8	22.7
Average design without error recovery	3.42 Mb	554.4	206
Self-recovery architecture A	9177 b	1.5	$7.68 \cdot 10^4$
Self-recovery architecture B	719 b	0.12	$1.0 \cdot 10^6$
Self-recovery architecture C	554 b	0.09	$1.3 \cdot 10^6$
Self-recovery architecture D	55b	0,009	$13.0 \cdot 10^6$

8 Conclusions

This dissertation addresses the problem of increasing the reliability and dependability of applications implemented in a SRAM-based FPGA. The main reliability concerns of SRAM-based FPGAs are soft-errors produced by radiation. Ion, proton, or neutron particles with high energies that hit the circuit can produce enough charge to change the contents of the memory cell (bit-flip). This phenomenon is called a single-event-upset SEU. These bit-flips affect the FPGA circuits' configuration memory.

To cope with soft-errors different techniques have been proposed. The fault-tolerance techniques are based on hardware and or temporal redundancies. The techniques proposed in the thesis improve the conventional on-line testing and recovery techniques in terms of reduced hardware overhead, reduced timing overhead, and increased reliability.

Developed solutions were thoroughly tested by hardware fault-emulation experiments. These experiments require understanding of the FPGA configuration bit-stream and efficient tools, which the reports of other authors usually lack. For this purpose we developed an automated fault-emulation tool that is using partial reconfiguration. It handles a wide range of SEU-induced faults on FPGA resources including LUT, flip-flop, and Block RAM. The proposed method is suitable for any system that can be equivalently ported to the FPGA platform with partial reconfiguration capabilities.

In the following we refer to the main problems elaborated in the thesis and summarize the developed solutions.

We addressed the problem of designing a compact on-line data-correction mechanism and BIST for the advanced-encryption-standard (AES) algorithm suitable for small-size FPGAs. The developed on-line test technique is based on parity prediction and has the following features:

- The parity check is performed in all the AES processes: encryption, decryption and key schedule.
- No additional FPGA hardware resources are needed to implement the parity prediction of the Sub Bytes and Mix Columns operations, which results in a low hardware overhead.

The cryptographic algorithms exhibit a good statistical property for pseudorandom testing. We used this property to make an efficient BIST with a low hardware overhead. The idea is to run the encryption, decryption, and expand the key processes with parity error-detection in a number of iterations. The output of the previous iteration is fed into the input of the next iteration. In this way the pseudorandom input test patterns are generated. The parity check is used as a test of the AES core. Therefore, a separate output analyzer is not required. High fault coverage was achieved for an extensive set of faults, including functional faults occurring in the FPGA LUTs. Both the on-line error-detection mechanism and the BIST have low hardware overheads. The AES core with the on-line test and BIST can be implemented in a 50k FPGA device.

Next, an error-recovery technique for multiprocessor systems on the FPGA was developed. The method extended the single-processor version of the algorithm to multiprocessors. The proposed approach has a low hardware overhead since the FPGA configuration memory checking is performed by the same resources as those employed in

the target application. Different test scheduling techniques are possible, which provides high flexibility.

Furthermore we developed a small hardware mechanism for the recovery of soft-errors in FPGAs. The described error-recovery mechanism for SRAM-based FPGAs has a low hardware overhead and can be employed in different self-recoverable architectures. Our error-recovery mechanism supports the Xilinx Virtex 4 and Virtex 5 FPGA families and can be easily extended to Virtex 6 devices. A similar recovery principle can also be applied to the FPGA devices of other manufacturers.

The main contributions of the thesis can be summarized as follows.

- A compact 32-bit AES core including online error detection and an efficient built-in self-test suitable for small-size FPGAs was developed. The implemented AES is specially designed for FPGA-based embedded applications since it is tuned to specific FPGA logic resources. This is the smallest AES core on FPGA with on line error detection reported in the literature. In contrast to other solutions that focus on the individual AES processes (i.e., encryption, decryption, and key expansion) we perform an on-line test of the complete AES.
- An error-recovery method for embedded multi-processor systems on SRAM-based FPGAs was developed. The error-recovery algorithm performs an on-line test of the FPGA configuration memory and recovers errors using dynamic partial reconfiguration. The processor cores perform a distributed recovery procedure. The developed approach outperforms the existing single-processor error-recovery solutions.
- A small hardware mechanism for the recovery of soft-errors in FPGAs. The proposed solution has a smaller hardware overhead than other reported recovery mechanisms. The estimated reliability of our internal error-recovery mechanism is better than the existing Xilinx SEU controller, which is currently a widely accepted solution in practice.

9 Acknowledgements

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this thesis.

I would like to express my deepest gratitude to my advisor, Prof. Dr. Franc Novak, for his excellent guidance. He offered me his experience and ideas for my research, and advised me in proper witting of scientific papers.

Special thanks go to Assist. Prof. Dr. Anton Biasizzo. Without his expert help, support, and brilliant ideas this research would not have been possible.

I am grateful to my colleges at the Computer Systems Department for their help and inspiration during my research work at “Jožef Stefan” Institute.

Finally I would also like to thank my family and friends for their patience and support.

10 References

Aktouf, C.; Robach, C.; Kač, U., Novak, F. On-line testing of embedded architectures using idle computations and clock cycles. In: *IEEE International On-line Testing Workshop.* 5–7 (1999).

Al-Asaad, H.; Shringi, M. On-line built-in self-test for operational faults. In: *Proceedings of IEEE AUTOTESTCON.* 28–32 (2000).

Alderighi, M.; D'Angelo, S.; Mancini, M.; Sechi, G. R. A fault injection tool for SRAM-based FPGAs. In: *Proceedings of International On-line Testing Symposium- IOLTS.* 71–78 (2003).

Alexandrescu, D.; Anghel, L.; Nicolaidis, M. New methods for evaluating the impact of single event transients in VDSM ICs. In: *IEEE International Symposium On Defect And Fault Tolerance In Vlsi Systems Workshop, DFT.* 99–107 (Proceedings IEEE Computer Society, 2002).

Anderson, D. A. Design of self-checking digital networks using coding techniques coordinates. *Sciences Laboratory, Report R/527* (University of Illinois, 1971).

Antoni, L.; Leveugle, R.; Feher, B. Using run-time reconfiguration for fault injection in hardware prototypes. In: *Proceedings of Defect and Fault-tolerance of VLSI systems-DFT.* 245–253 (2002).

Asadi, G.; Miremadi, S. G.; Zarandi, H. R., Ejlali, A. Evaluation of fault-tolerant designs implemented on SRAM-based FPGAs. In: *Proceedings of Pacific Rim Dependability Conference-PRDC.* 327–332 (2004).

Asadi, H.; Tahoori, M. B. Soft error mitigation for SRAM-based FPGAs. In: 23rd *IEEE VLSI Test Symp.* 207–212 (2005).

Asadi, H.; Tahoori, M. B.; Mullins, B.; Kaeli, D.; Granlund, K. Soft Error Susceptibility Analysis of SRAM-Based FPGAs in High-Performance Information Systems. *IEEE Transactions on Nuclear Science* **54**, 2714 (2007).

Benso, A.; Prinetto, P. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation* (Kluwer Academic Publishers, 2003).

Berg M. et al. Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis. *IEEE Transactions on Nuclear Science* **55**, 2259 (2008).

Berger J. M. A note on error detection codes for asymmetric binary channels. *Information and control* **4**, 68 (1961).

Bertoni, G.; Breveglieri, L.; Koren, I.; Maistri, P.; Piuri, V. A Parity Code Based Fault

Detection for an Implementation of the Advanced Encryption Standard. In: *Proceedings of DFT*. 51–59 (2002).

Bertoni, G.; Breveglieri, L.; Koren, I.; Maistri, P.; Piuri, V. Error Analysis and Detection procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Transactions on Computers* **52**, 492 (2003).

Bosea, R. C.; Ray-Chaudhuri, D. K. On a class of error correcting binary group codes *Information and control* **3**, 68 (1960).

Carmichael, C. Triple Module Redundancy Design Techniques for Virtex® Series FPGA. *Xilinx Application manual XAPP 197* (2000).

Carmichael, C.; Wei Tseng, C. Correcting Single-Event Upsets in Virtex-4 FPGA Configuration Memory. *Xilinx Application manual XAPP1088* (2009).

Carter, W. C.; Schneider, P. R. Design of dynamically checked computers. In: *Proceedings of 4th IFIP Congress*. 878–883 (1968).

Chapman, K. SEU strategies for Virtex-5 Devices. *Xilinx Application manual XAPP864* (2010).

Crain, S.; Mazur, J. E.; Katz, R. B.; Koga, R.; Looper, M. D. Lorentzen K. R. Analog and digital single-event effects experiments in space. *IEEE Transactions on Nuclear Science*, **25**, 983140 (2001).

Di Natale, G.; Doulcier, M.; Flottes, M. L.; Rouzeyre, B. Self-Test Techniques for Crypto-Devices. *Integration, the VLSI Journal* **18**, 329 (2010).

Di Natale, G.; Doulcier, M.; Flottes, M. L.; Rouzeyre, B. A Reliable Architecture for Parallel Implementations of the Advanced Encryption Standard, *Journal of Electronic Testing* **25**, 269 (2009).

Di Natale, G.; Flottes, M. L.; Rouzeyre, B. A Novel Parity Bit Scheme for SBox in AES Circuits. In: *Proceedings of Design and Diagnostics of Electronic Circuits and Systems*. 11–13 (2007).

Drineas, P. and Makris, Y. SPaRe: selective partial replication for concurrent fault-detection in FSMs. *IEEE Transactions on Instrumentation and Measurement* **52**, 818733 (2003).

Eto, E. Difference-Based Partial Reconfiguration. *Xilinx application manual XAPP 290* (2007).

Feldhofer, M.; Dominikus, S.; Wolkerstorfer, J. Strong authentication for RFID systems using the AES algorithm. In: *Proceedings of Hardware and Embedded Systems*. 357–370 (Springer Verlag, 2004).

Freiman, C. V. Optimal error detection codes for asymmetric binary channels. *Information and control* **5**, 64 (1962).

Gaj, K.; Chodowiec, P. Very compact FPGA Implementation of the AES algorithm. In: *Proceedings of Hardware and Embedded Systems*. 319–333 (Springer Verlag, 2003).

Hamming, R. W. Error detecting and correcting codes. *Bell System Technical Journal* **29**, 147 (1950).

Heiner, J.; Collins, N.; Wirthlin, M. Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing. In: *IEEE Aerospace Conf.* 1–10 (2008).

Hodjat, A.; Verbauwhede, I. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA. In: *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines.* 308–309 (IEEE Computer Society, 2004).

Hulme, C. A.; Loomis, H. H.; Ross, A.; Rong Yuan, A. Configurable fault-tolerant processor (CFTP) for spacecraft onboard processing. In: *Proc. IEEE Aerospace Conf.* 2269–2276 (2004).

Jing, M. H.; Chen, Z. H.; Chen, J. H.; Chen, Y. H. Reconfigurable system for high-speed and diversified AES using FPGA. *Microprocessors and Microsystems* **31**, 94 (2007)

Joye, M.; Manet, P.; Rigaud, J. B. Strengthening Hardware AES Implementations Against Fault Attacks. *IET Information Security* **1**, 106 (2007).

Kafka, L.; Novak, O. FPGA-based fault simulator. In: *Proc. IEEE Design and Diagnostics of Electronic Circuits and systems*. 272–276 (2006).

Karri, R.; Wu, K., Mishra, P.; Kim, Y. Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* **21**, 1509 (2002).

Karri, R.; Wu, K.; Mishra, P.; Kim, Y. Concurrent Error Detection of Fault Based Side-Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers. In: *Proceedings of Design Automation Conference*. 579–584 (2001).

Katz, R. et al. An SEU-Hard flip-Flop for Antifuse FPGAs. In: *International Conference On Military And Aerospace Applications Of Programmable Logic Devices, MAPLD.* 1–8 (2001).

Leavy, J.; Hoffmann, L. F.; Shovan, R. W.; Johnson, M. T. Upset due to a single particle caused propagated transient in a bulk CMOS microprocessor. *IEEE Transactions on Nuclear Science* **38,** 1493 (1991).

Lesea, A.; Drimer, S.; Fabula, J. J.; Carmichael, C.; Alfke, P. The rosetta experiment: atmospheric soft error rate testing in differing technology FPGAs. *IEEE Transactions on Device and Materials Reliability* **5**, 317 (2005).

Leveugle, R.; Antoni, L. Dependability Analysis: a New Application for Run-Time Reconfiguration. In: *Proceedings of IPDPS.* 1530–2075 (2003).

Leveugle, R.; Antoni, L.; Feher, B. Using run-time reconfiguration for fault injection applications. In: *Proceedings IMTC.* 1468–1473 (2001).

Lim, D.; Peattie, M. Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations. *Xilinx application manual XAPP 290* (2002).

Lima Kastensmidt, F.; Carro, L.; Reis, R. *Fault-Tolerance Techniques for SRAM-Based*

FPGAs (Springer, Dordrecht, 2006).

Lima Kastensmidt, F.; Sterpone, L.; Sonza Reorda, M.; Carro, L. On the Optimal De-sign of Triple Modular Redundancy Logic for SRAM-Based FPGAs. In: *IEEE Proc. Design, Automation and Test in Europe Conference.* 1290–1295 (2005).

Maistri, P.; Leveugle, R. Double-Data-Rate Computation as a Countermeasure against Fault Analysis. *IEEE Transactions on Computers* **57**, 149 (2008).

Mathew, J.; Jabir, A. M.; Pradhan, D. K. Design Techniques for Bit-Parallel Galois Field Multipliers with On-Line Single Error Correction and Double Error Detection. In: *Proceedings of IEEE International On-Line Testing Synposium.* 16–21 (2008).

Maxfield, C. *FPGAs: Instant Access* (Elsevier, Oxford, 2008).

Messenger, G. C. Collection of charge on junction nodes from ion tracks. *IEEE Trans. On Nuclear Science* **29**, 2024 (1982).

Moratelli, C.; Ghellar, F.; Cota, E.; Lubaszewski, M. A Fault-Tolerant DFA-Resistant AES Core. In: *Proceedings of ISCAS*. 244–247 (2008).

Mozaffari-Kermani, M.; Reyhani-Masoleh, A. A Lightweight Concurrent Fault Detection Scheme for the AES S-boxes Using Normal Basis. In: *Proceedings of CHES*. 113–129 (2008).

Mozaffari-Kermani, M.; Reyhani-Masoleh, A. Fault Detection Structures of the S-boxes and the Inverse S-boxes for the Advanced Encryption Standard. *Journal of Electronic Testing* **99**, 1 (2009).

National Institute of Standards and Technology (NIST). Specification for the ADVANCED ENCRYPTION STANDARD (AES), *Federal Information Processing Standards Publication 197* (2001).

Nicolaidis, M.; Zorian, Y., On-Line Testing for VLSI-A Compendium of Approaches. *Journal of Electronic Testing* **12**, 7 (1998).

Ocheretnij, V.; Kouznetsov, G.; Karri, R.; Gössel, M. On-line Error Detection and BIST for the AES Encryption Algorithm with Different S-Box Implementations. In: *Proceedings of On-Line Testing Symposium*. 141–146 (2005).

Paschalis, A.; Gizopoulos, D. Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **24**, 88 (2005).

Peterson, W. *Error-correcting codes* (The Mit Press, Cambridge, 1980).

Peterson, W. W. On checking an adder. *IBM Journal of Research and Development* **2**, 166 (1958).

Pilotto, J.; Azambuja, R.; Lima Kastensmidt, F. Synchronizing Triple Modular Redundant Designs in Dynamic Partial Reconfiguration Applications. In: *Proceedings of the 21st annual symposium on Integrated circuits and system design*. 199–204 (2008).

Pratt, B. et al., Fine-Grain SEU Mitigation for FPGAs Using Partial TMR, *IEEE Transactions on Nuclear Science* **55**, 2274 (2008).

Rebaudengo, M.; Sonza Reorda, M.; Violante, M. A new functional fault model for FPGA Application-Orianted testing. In: *Proceedings of Defect and Fault Tolerance in VLSI Systems*. 372–380 (2002).

Reed, I. S.; Solomon G. Polynomial codes over certain finite fields *J. Soc. Indust. and Appl. Math.* **8**, 300 (1960).

Rouvroy, G.; Stadeart, F. X.; Quisquter, J. J.; Legat, J. D. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Aplications. In: *Proceedings of Information Technology: Coding and Computing*. 583–587 (2004).

Satoh, A.; Sugawara, T.; Homma, N.; Aoki, T. High-Performance Concurrent Error Detection Scheme for AES Hardware. In: *Proceedings of CHES*. 100–112 (2008).

Schrimpf, R. D.; Fleetwood, D. M. *Radiation Effects And Soft Errors In Integrated Circuits and Electronic Devices* (World Scientific, London, 2004).

Schubert, A.; Anheier, W. On Random Pattern Testability of Cryptographic VLSI Cores, *Journal of Electronic Testing* **16**, 803 (2000).

Smith, M. J. S. (10 ed.) *Application-Specific Integrated Circuits* (Addison-Wesley, Boston, 2001).

Sterpone, L.; Violante, M. A New Algorithm for the Analysis of the MCUs Sensitiveness of TMR Architectures in SRAM-Based FPGAs. *IEEE Transactions on Nuclear Science* **55**, 2019 (2008).

Sterpone, L.; Violante, M. A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs. *IEEE Transactions on Nuclear Science* **54**, 965 (2007).

Sterpone, L.; Violante, M.; Rezgui, S. An Analysis Based on Fault Injection of Hardening Techniques for SRAM-Based FPGAs. *IEEE Transactions on Nuclear Science* **53**, 2054 (2006).

Tanoue, S.; Ishida, T.; Ichinomiya, Y.; Amagasaki, M.; Kuga, M.; Sueyoshi, T. A novel states recovery technique for the TMR softcore processor. In: *Proc. Int. Conf. Field Programmable Logic and Applications FPL*. 543–546 (2009).

Vaidyanathan, R.; Trahan, J. L. *Dynamic Reconfiguration: Architectures and Algorithms* (Kluwer Academic/Plenum Publishers, New York, 2004).

Verdel, T.; Makris, Y. Duplication-based concurrent error detection in asynchronous circuits: shortcomings and remedies. In: *Proceedings of 17th IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems-DFT*. 345–353 (2002).

Violante, M.; Meinhardt, C.; Sonza Reorda, M.; Reis, R. A Low-Cost Solution for Deploying Processor Cores in Harsh Environments. *IEEE Transactions on Industrial Electronics* **58**, 2617 (2011).

Von Neumann, J. Probabilistic logics and synthesis of reliable organizms from unreliable components. In: *Automata Studies*. 43–98 (NJ: Princeton Univ. Press, New York, 1956).

Wegrzyn, M.; Novak, F.; Biasizzo A. Functional testing of processor cores in FPGA-based applications. *Computing and Informatics* **28**, 1001 (2009).

Wu, K.; Karri, R.; Kuznetsov, G.; Gössel, M. Low Cost Concurrent Error Detection for the Advanced Encryption Standard. In: *Proceedings of International Test Conference*. 1242–1248 (2004).

Wu, S.; Yen, H. On the S-box architectures with concurrent error detection for the advanced encryption standard. *IEICE transactions on fundamentals of electronics communications and computer science* **89**, 2583 (2006).

Xilinx, Device Reliability Report, Second Quarter 2011. *Xilinx User Guide UG116* (2011).

Xilinx, PlanAhead Software Tutorial. *Xilinx User Guide UG 743* (2010).

Xilinx, Virtex 4 Configutation User Guide. *Xilinx User Guide UG071* (2009).

Xilinx, Virtex 4 User Guide. *Xilinx User Guide* UG070 (2008).

Xilinx, Virtex 5 Configutation User Guide. *Xilinx User Guide UG191* (2010).

Xilinx, Virtex 5 User Guide. *Xilinx User Guide UG190* (2010).

Xilinx, Xilinx TMRTool. *Xilinx User Guide UG 156* (2007).

Yang, F.; Chakravarty, S.; Devta-Prasanna, N.; Reddy, S. M.; Pomeranz, S. M. R. An Enhanced Logic BIST Architecture for Online Testing. In: *Proceedings of 14th IEEE Int. On-Line Testing Symp-IOLTS*. 10–15 (2008).

Yen, C.; Wu, B. Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard. *IEEE Transactions on Computers* **55** 720 (2006).

Zhang, X.; Parhi, K. K. High-speed VLSI architectures for the AES algorithm, *IEEE Transactions on VLSI systems* **12,** 957 (2004).

11 Bibliography

Legat, U.; Biasizzo, A.; Novak, F. Partial Runtime Reconfiguration of FPGA, Applications and a Fault Emulation Case Study. *International Review on Computers and Software (IRECOS)* **4**, 606 (2009).

Legat, U.; Biasizzo, A.; Novak, F. Automated SEU fault emulation using partial FPGA reconfiguration. *Design and Diagnostics of Electronic Circuits and Systems (DDECS).* 24–27 (2010).

Legat, U.; Biasizzo, A.; Novak, F. A compact AES core with on-line error-detection for FPGA applications with modest hardware resources. *Microprocessors & Microsystems* **35**, 405 (2011).

Legat, U.; Biasizzo, A.; Novak, F. Self-reparable system on FPGA for single event upset recovery. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC).* 1–6 (2011).

Legat, U.; Biasizzo, A.; Novak, F. Soft Error Recovery Technique for Multiprocessor SOPC. *Asian Test Symposium* (ATS). 175–180 (2011).

Index of Figures

Figure 1: Basic Architecture of SRAM-based FPGA.. 8
Figure 2: Basic structure of a slice in the configurable logic block. 9
Figure 3: Structure of Virtex 4 CLB 10
Figure 4: Structure of Virtex 5 CLB 10
Figure 5: Connection signals of a block RAM in dual-port mode 11
Figure 6: Internal routing example 11
Figure 7: Configuration organization of configuration memory 12
Figure 8: Configuration interfaces of Xilinx FPGAs 13
Figure 9: FPGA design layout with three reconfigurable modules 15
Figure 10: Charged particle striking the silicon circuit 16
Figure 11: Transient current pulse induced by a particle hit 17
Figure 12: Soft errors in integrated circuits 17
Figure 13: SEU in a SRAM memory cell 18
Figure 14: SEU in an internal FPGA routing 19
Figure 15: SEU in Configurable Logic Block 19
Figure 16: Concurrent on-line testing principle 24
Figure 17: Time-redundancy technique to detect a SET in combinational circuits 24
Figure 18: Duplication and comparison technique to detect SET and SEU 25
Figure 19: Error-detection technique using parity check 25
Figure 20: Time-redundancy technique to mitigate a SET in combinational circuits 27
Figure 21: The majority voter scheme and truth table 27
Figure 22: The architecture of the basic TMR technique 27
Figure 23: The architecture of TMR for the states recovery 28
Figure 24: The architecture of Xilinx TMR 28
Figure 25: Hamming code error-correction implementation 29
Figure 26: Hamming code encoder and decoder implementation 30
Figure 27: Error-recovery techniques 31
Figure 28: Fault-injection procedure with artificial radiation 33
Figure 29: Fault-injection procedure with HDL simulation 34
Figure 30: Fault-emulation procedure 35
Figure 31: Fault-emulation design flow 35
Figure 32: Basic hardware architecture of the proposed approach 36
Figure 33: The data flow for the AES encryption algorithm 40
Figure 34: The first quarter of the round memory access 42
Figure 35: 32-bit AES architecture (Rouvroy et al., 2004) 44

Figure 36: AES architecture with parity check 45
Figure 37: Masking of a single fault in a dual-port ROM..47
Figure 38: Correct implementation of CT-boxes to prevent fault masking 48
Figure 39: Parts of CT-box ROM accessed during the encryption, decryption 50
Figure 40: BIST data flow...51
Figure 41: BIST architecture...51
Figure 42: Hardware-software architecture of the fault-emulation tool 52
Figure 43: Extracting information from 4 input LUT functional blocks 53
Figure 44: Simplified Virtex 4 slice structure (G side) 54
Figure 45: Stuck at 0 fault on the G2 input 55
Figure 46: Fault coverage of CT-boxes versus the number of BIST iterations 57
Figure 47: Fault coverage of the AES logic in the LUTs...58
Figure 48: Required hardware architecture 60
Figure 49: Block diagram of the dual-processor system...63
Figure 50: The structure of the fault-emulation system 63
Figure 51: Hardware architecture of the error-recovery mechanism 67
Figure 52: Simplified FSM of Virtex 5 recovery controller 68
Figure 53: Hardware-architecture block diagram of error-recovery mechanism...............70
Figure 54: Self-recovery architectures 72
Figure 55: The structure of the fault-emulation experiment 73

Index of Tables

Table 1: Failures In Time (FIT) by FPGA technology 20
Table 2: Comparison of the implementation results 49
Table 3: Frequency of the accessed CT-box ROM parts 50
Table 4: Implementation of the proposed BIST on the Xilinx Spartan 3 XC3S50-4........ 52
Table 5: Injected faults and fault coverage 56
Table 6: Syndrome value and the corresponding error status... 61
Table 7: Fault-emulation results 64
Table 8: SEU reliability estimation of our dual-processor system 65
Table 9: Error-recovery time comparison.. 69
Table 10: Hardware implementation comparison... 69
Table 11: Hardware implementation of the error-recovery mechanism in TMR 70
Table 12: Fault-emulation results for the architectures A and B on Virtex 5 FPGA......... 74
Table 13: Fault-emulation results for the architectures C and D on Virtex 5 FPGA......... 74
Table 14: SEU reliability estimation 75

Index of Algorithms

Algorithm 1: Error recovery algorithm pseudo code.. 62

Appendix

1. Journal papers

Legat, U.; Biasizzo, A.; Novak, F. Partial Runtime Reconfiguration of FPGA, Applications and a Fault Emulation Case Study. *International Review on Computers and Software (IRECOS)* **4**, 606 (2009).

Legat, U.; Biasizzo, A.; Novak, F. A compact AES core with on-line error-detection for FPGA applications with modest hardware resources. *Microprocessors & Microsystems* **35**, 405 (2011).

Legat, U.; Biasizzo, A.; Novak, F. On line self-recovery of embedded multi-processor SOC on FPGA using dynamic partial reconfiguration. *Information technology and control*, Accepted for publication in 2012.

Legat, U.; Biasizzo, A.; Novak, F. SEU recovery mechanism for SRAM-based FPGAs. *IEEE Transactions on nuclear science*, In final review stages 2012.

2. Conference papers

Legat, U.; Biasizzo, A.; Novak, F. Some approaches to partial reconfiguration of FPGA. In: *International Conference on Microelectronics, Devices and Materials and the Workshop on Advanced Plasma Technologies* (MIDEM). 125–128 (2008).

Legat, U.; Biasizzo, A.; Novak, F. On-line error detection of a compact 32-bit hardware AES implementation. In: *European Test Symposium* (ETS), 1–5 (2009).

Legat, U.; Biasizzo, A.; Novak, F. Fault simulation with partial reconfiguration of FPGA. In: *International Conference on Microelectronics, Devices and Materials and the Workshop on Advanced Plasma Technologies* (MIDEM). 157–160 (2009).

Legat, U.; Biasizzo, A.; Novak, F. Automated SEU fault emulation using partial FPGA reconfiguration. In: *Design and Diagnostics of Electronic Circuits and Systems (DDECS).* 24–27 (2010).

Legat, U.; Biasizzo, A.; Novak, F. Concurent self-test and repair of embedded multi-core SOC on FPGA using dynamic partial reconfiguration. In: *International Conference on Microelectronics, Devices and Materials and the Workshop on Advanced Plasma Technologies* (MIDEM). 161–164 (2010).

Legat, U.; Biasizzo, A.; Novak, F. FPGA soft error recovery mechanism with small hardware overhead. In: *European Test Symposium* (ETS), 207 (2011).

Legat, U.; Novak, F. Self-reparable systems on FPGA. In: *Jožef Stefan International Postgraduate School Students Conference* (IPSSC). 86–91 (2011).

Legat, U.; Biasizzo, A.; Novak, F. Self-reparable system on FPGA for single event upset recovery. In: *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC).* 1–6 (2011).

Legat, U.; Biasizzo, A.; Novak, F. Soft Error Recovery Technique for Multiprocessor SOPC. In: *Asian Test Symposium* (ATS). 175–180 (2011).