CONTRIBUTIONS IN THE FIELD OF INFORMATION SECURITY. ERROR-DETECTION ARCHITECTURES

PhD Thesis for obtaining the scientific
title of Doctor Engineer
at the
Politehnica University of Timişoara
in the field of COMPUTER ENGINEERING
for

Ing. Andreea BOZESAN

Scientific advisor:	Prof.Dr.Eng. Mircea VLADUTIU
Scientific reviewers:	Prof. Dr. Eng. Horia CIOCARLIE
	Conf.Dr.Eng. Mihai UDRESCU
	Conf.Dr.Eng. Lucian PRODAN

Date of public PhD presentation: 09.04.2015

Romania, 300159 Timişoara, Bd. Republicii 9,
tel. 0256 403823, fax. 0256 403221
e-mail: editura@edipol.upt.ro

ABSTRACT

This thesis, entitled “Novel Means of Achieving Information Security. Error-Detection Architectures” encompasses the research of 3 years as part of the PhD program, but also represents the continuation of the work started during the university and masters programs regarding the IDEA NXT crypto-algorithm.
The thesis starts by presenting the state-of-the art in cryptography, concepts and challenges as well as the current trends. It then presents the asymmetric and symmetric cryptographic algorithms as the main techniques used to secure important information, with a focus on the IDEA NXT algorithm, the newest trend in encryption, and the work carried on as part of this research to improve the speed of the algorithm.

A major part of the current research is from the testability domain. An introduction in this field is made, followed by the presentation of the various strategies used (both online and offline) and the different fault models which generally affect the well-functioning of a system. Also, the need for fault tolerant designs and reliable systems is highlated and details about how to achieve this goal are presented as well. Bringing together the need for data security and system availability and reliability, a couple of testing architectures were proposed and implemented for IDEA NXT and the obtained experimental results regarding the performance are presented in detail, in comparison with similar experiments conducted for other crypto-algorithms, especially AES. The efficiency of the error-detection mechanisms is demonstrated by injecting faults into the designs and verfying the testing architectures’ outputs. Finally, future work and directions are briefly discussed.

Timişoara, April 2015   Andreea Bozesan

ACKNOWLEDGEMENTS

A major research project could never be the work of a single person. The contributions of many different people, in their different ways, have made this possible. I would like to express my appreciation especially to the following.

I am forever grateful to my supervisor and mentor, Prof. Dr. Ing. Mircea Vladutiu, who has given me continuous support and advice, transferred some of his vast knowledge and understanding of test engineering and Cryptography and also encouraged me to continue my research every step of the way, even in the moments of indecision and dissapointment.

Dr. Ing. Flavius Opritoiu for the long weekdays and weekends of work on various papers and his efforts in proof reading the drafts and giving feedback. Without his support and guidance throughout the research project, I wouldn't have been able to put the topic together.

I am also grateful to my colleagues in ACSA, Dr. Lucian Prodan and Dr. Mihai Udrescu for their encourages and experience sharing, and also to my fellow PhD colleagues Alexandru Iovanovici, Alexandru Topirceanu and Cristian Cosariu for their knowledge sharing and help with information and organizational topics.

I would also like to acknowledge the academic and financial support of the Politehnica University of Timisoara, which supported my participation to various conferences and gave me access to laboratories and material necessary to conduct my research and experiments. I am also grateful to the AFCEA international organisation which supported my participation to the SIITME snd AQTR symposiums and whose proud member I am since 2013.

Of course, this project would not have been possible without the participation of the students, which participated in some of the experimental work. Working with them gained me perseverance and eye for detail.

Last but not least, I would like to thank my mother for her unconditional and continuous love and support. She sticked with me though my rough patches and encouraged me to never give up, but fight each day for my goals.

Bozesan, Andreea

NOVEL MEANS OF ACHIEVING INFORMATION SECURITY. ERROR-DETECTION MECHANISMS

PhD Thesis of UPT, Series 9, Nr. 52, Editura Politehnica, 2015, 106 pages, 79 figues, 90 tables.

ISSN: 1842-4937

ISBN: 978-973-625-834-3

Key words: cryptography, encryption algorithm, IDEA NXT, testing, parity-checking, DFT, offline tesing, BIST

Summary,

With the chosen research topics, the thesis adresses up-to-date concerns and problems people are dealing each day regarding the security of sensitive information and the possibility of their data and privacy being invaded. Encryption algorithms are presented as a solution for this.

Also, error-detection mechanism must be used for ensuring that the algorithms are fully functional at all times, and in case an error is manifesting, it is found as soon as possible as to minimize the damage done and reduce the costs of repair.

List of Figures

List of Tables

Contents

INTRODUCTION

This PhD thesis describes the research activity carried on at the “Politehnica” University of Timisoara in the past 4 years. The doctoral program makes a foray in the domain of Computer Science, and more specifically the fields of Computer Hardware Design and Cryptography, with an emphasis on data security mechanisms, encryption algorithms and Built-In Testing techniques for encryption algorithms.

The opportunity of the theme is justified by the fact that start of the new millennium stays under the sign of technological development, and the field that has grown the most and in the shortest period of time is the one of computers. There is no field in the industry that doesn't make use of computers and the new facilities that come with the expansion of computer networks and the Internet. The present and future in this phase of society evolution, characterised by globalisation, feasible due to the extraordinary progress made in science and technology in recent decades, belongs to rapid communications made with the help of computers, networks of interconnected computers, digital technology, the Internet and GSM mobile communications. These systems have the undeniably advantage of the huge mount of data that can be shared with a used practically insignificant material support and great speed of data transmission in comparison to the amount of data. This type of communications has a major drawback at the moment, namely the difficulty of ensuring confidentiality of information or avoiding the data interception by unauthorized third partys. In the absence of securing such data, extremely unpleasant events with unpredictable consequences can be produced. An un-authorised party can vitiate or change the content of an email, break bank accounts, transfer money to other accounts than intended, steal personal card information, steal information in the military, scientific, diplomatic fields, etc.

Security in these areas can be achieved by encrypting the stored data or the one being transmitted through the internet with cryptographic algorithms. Even some of the mobile phones can use software that encrypts the call in progress. The most widely-used crypto-algorithm at present date is the symmetric bloc cipher, AES. However, AES starts to show its weakenesses in recent years, as more and more articles describing successful attacks to breaking AES appeared in the past years.

Going through the literature, I discovered a new family of symmetric-algorithm, built on the structures of IDEA and AES crypto-algorithms, which is believed to offer an increased security than AES, since there is no proved attack against IDEA NXT at the moment. Other advantage of IDEA NXT is the fact that its decryption process is almost the same as the encryption process, which makes it perfect to use for systems where bi-directional communication is done.

Given all of the above, I believed IDEA NXT could successfully become the new generation in encryption in a few years, the rightful successor and replacement of AES. For this reason I considered appropriate to studyIDEA NXT in more detail and to create a hardware implementation of the algorithm (taking as starting point its mathematical structure) which can be run on a FPGA device, so that the algorithm could be easily integrated in a crypto-chip. This is the first implementation of this kind for NXT, the other two hardware implementations built for it up-to-this date are made for ASICs. Once an algorithm is used in a real system, faults inserted in different stages of the algorihm's design or implementation could cause malfunctions which could lead to faulty system components or even complete system failure. In order to avoid the propagation of such errors for IDEA NXT, I believed approapiate to design and implement in hardware a couple of off-line (while the system or algorithm is not functional) as well as on-line testing architectures (concurrent with the algorithm's operation) for detecting various types of errors that could compromise its well-functioning. The error-detection mechanisms are built specifically for the this algorithm and are meant to detect at early stages the faults introduces in the design and implementation phases on the algorithms, as well as the ones caused by external factors which can somehow alter the normal operation flow.

I imagined this research as a top-down approach, following the flow showed by Fig. 1.

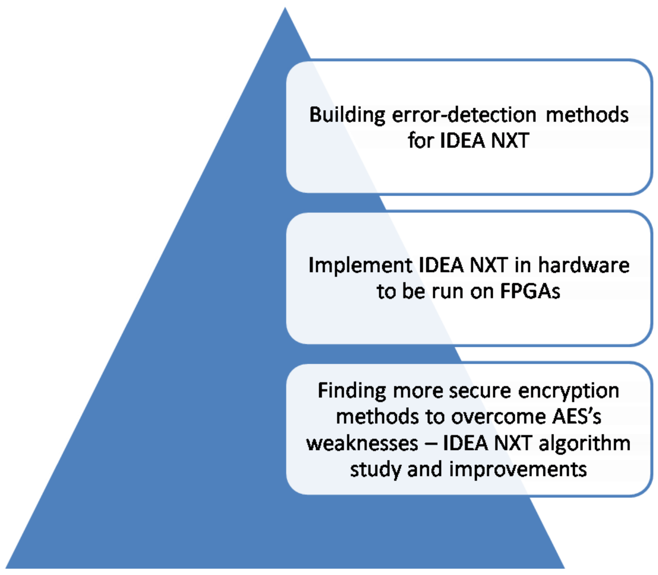

Building error-detection methods for IDEA NXT
Implement IDEA NXT in hardware to be run on FPGAs
Finding more secure encryption methods to overcome AES's weaknesses – IDEA NXT algorithm study and improvements

Fig 1 Thesis nucleum – top-down approach

The following paragraphs briefly present the content of each chapter of the thesis.

Chapter 1 makes an incursion in the field of cryptography and symmetric crypto-algorithms.

Chapter 2 focuses on the novel private-key cryptographic algorithm, IDEA NXT, highlighting its strong points in comparison to similar, well-known symmetric ciphers. A series of optimizations I brought to the fundamental structure of this crypto-algorithm, especially regarding its speed are also illustrated. The hardware implementation of the IDEA NXT algoritm is also described in a detail manner, as well as the one built for the modified version of NXT.

Notions of testing principles and methods from the literature and the main fault tolerance techniques are examined and discussed in Chapter 3.

Chapter 4 introduces the testing architectures which I designed for the IDEA NXT algorithm. This chapter also presents the fault-injection method I used for simulating the presence of faults in the IDEA NXT algorithm in order to test the fault detection rate of the proposed error-detection schemes.

Finally, Chapter 5 draws the conclusions of this thesis based on the experimental results, marking its original contributions and drawing future directions of research.

Chapter 1 FUNDAMENTALS OF CRYPTOGRAPHY

Communication systems with interconnected computers are the future in global communications. But with the expansion of electronic data processing and their transmission through computer networks, the importance given to security aspects has increased considerably. The need for security and authenticity occurs at all architectural levels of computer networks connected to the Internet to prevent unauthorized access to the network, which could lead to deterioration or destruction of data. A series of questions also arose:

- Which is the way to send a message secretly, without it being intercepted by a third party?
- How can the sender of the message be sure that his message will not arrive at its destination un-altered?
- Which is the way to assure that a message comes from the sender he expects and that the message comes un-altered?

A low-level, unprotected solution would be to hide the existence of the message itself by writing it, for example, with an invisible ink, or to send it through someone you trust. A high-level, scientific solution for all this is to be found in the field of Cryptography.

Cryptography is the science that deals with the transmission of secret data by converting data (plain text) into encrypted text (ciphertext) using a piece of information called "key", known only to authorized parties. Today, cryptographic techniques can be found both in the public and private sectors, wherever processing, transferring and / or storing information is necessary. Cryptography can be used to alert the changing content of a document to certify the identity of the person who issues a message, to maintain security of online communications or to protect important and private data.

It can be considered that this science began some 4000 years ago, when Egyptians wrote enciphered messages on pyramids. The earliest cryptographic system SCYTALE is attributed to Caesar who sent commands to his generals with its help nearly 2500 years ago. The system was a very simple one which worked by replacing one letter of the alphabet with another (A with D, B with E, etc). "In 1460, Leon Alberti created a cipher wheel. In 1585, Blaise de Vignere formally described the polyalphabetic substitution cipher. In 1790, Jefferson developed a cylinder comprised of 26 disks, each with a random alphabets." [1].

In 1817, Wadsworth invented a polyalphabetic cipher composed of two concentric circles (wheels) knows as the *Wheatstone disk*. Probably the best knows cipher of the last century is the *Enigma* which was used during World War II, composed of a number of rotor wheels with internal cross-connections which achieved word substitution by making use of a continuous alphabet [2].During the interbelic period, things started to advance quickly and solidly in this area, as the military field needed extremely secure communications to win their wars. In the late 1960s, in the IBM Watson Laboratory, *Horst Feistel* created the U.S *Data Encryption Standard (DES)* and in 1976, *Withfield Diffie* and *Martin Hellman* wrote a paper called "*New Directions on in Cryptography*" which changed fundamentally the approach to cryptography. They even invented cipher which would gain a great popularity – Diffie-Hellman public-key cryptosystem. Nowadays, the widespread use of computers, mobile devices, tablets, POS card transactions and the Internet have led to an even bigger interest in cryptography, because of the need to protect privacy.

1.1 Basic Notions and Taxonomies Regarding Dependability and Safety of Calculations

1.1.1 Definition of dependability and security. Attributes

The field of dependability comes from other related areas, such as fault-tolerance, trust and safety systems which were intensively studied in the '60s. Due to increasing interest in these areas in the 1970 and 1980 decades, the term "trust" (reliability) became overloaded and used with a different meaning than originally, as a measure of the failures of a system to accommodate various sizes which now will classified in safety, integrity, etc. Jean-Claude Laprie used the term "Dependability" to encompass all these relational disciplines, in the early 1980s.

As compared to early days, a keen interest is now manifested in this area. Research is undertaken by a number of prominent international conferences, of which can be mentioned International Conference on Dependable Systems and Networks and the International Symposium on Software Engineering trusted. The original definition of dependability encompasses the following non-functional requirements: availability, reliability and maintenance, which combines with the so-called depend *threats* and *failures*. This definition was later broadened to incorporate safety and security.

The primary definition of dependability is as follows: "dependability is a property that justifies the confidence in a system" [4]. The definition stresses the need to justify this confidence.

Alternative definition, that provides criteria for deciding whether a service is reliable, is: "the ability to avoid faults and the failure of system services which occur more frequently and in a much more severe than should be tolerated". Dependence between the two systems is the degree to which the first system dependability is affected by the other system.

The concept of dependability has the following attributes:

- *Safety* and *trust*: the continuity of services in the fairness
- *Safety*: absence of catastrophic consequences on the users and the environment
- *Integrity*: absence of any alterations to the system
- *Maintenance*: ability to withstand or repair modifications
- *Availability*: the system is prepared for correct functionality

As suggest the above definitions, only availability and reliability are quantified by direct measurements, others are subjective. Trust, for example, can be measured by the number of failures over time. When you bring the problem under discussion there is a security attribute that should be taken into account, namely confidentiality, which means the absence of unauthorized disclosure of information. Security is a composite of attributes that require the availability of co-existence (for actions authorized) of confidentiality and integrity (with the improper purpose of "unauthorized"). It is very important to grasp the difference between the definition of privacy (the duty to protect secret information), discretion (limiting the number of accesses to the system) and privacy (the ability and right to protect informed personal secret).

Attributes vary in importance according to the application that is intended to be achieved given computational system. Also, the extent to which a system possesses the attributes of dependability and security is a relative, because of the unavoidable presence of errors, systems are never fully available, reliable or safe.

In addition to the attributes already discussed a number of secondary attributes can be defined, which refine or specialize the primary ones. One such attribute is robustness, .ie. dependability vis-à-vis to the external errors, the system equation characterizing a specific class of faults [6]. The newly defined notion is particularly relevant for security, for which the following secondary attributes can be mentioned:

• Responsibility: availability and integrity of the identity of the person who made a particular surgery
• Authenticity: the origin and content integrity of a message
• Non-repudiability: availability and integrity of the identity of the sender, the addressee of a message (non-repudiation of origin, namely the reception)

A fundamental goal of cryptography is to achieve these goals both in theory and in practice [3].

Dependability and security classes are generally defined by analyzing the frequency and severity of errors for the relevant attributes of database applications. Variations in highlighting various attributes directly influence's balance techniques (prevention, tolerance, removal and provision of error) which will be applied to transform the resulted system in a safe and reliable one. Applying security measures to a system improves dependability by limiting the number of external errors.

1.1.2 Threats to the dependability and security of a system

Maintaining the dependability and security of a system unchanged is a challenge, because each component can make serious problems. To be able to fight against the threat, it must be first located and understood; this presumes grouping threats according to their severity, source of origin, the potential route of spread, etc. This is facilitated by the development of algorithms to protect the system from specific threats. In the development phase, errors can be introduced into the system by: [4]

Fig 2 Types of Maintenance

- The environment, the physical phenomena existing
- System Developers

- Software and hardware tools used by researchers in the development process
- Development and production facility

During its use, the system interacts with the environment, made-up of the following elements:

- Environment
- Network administrators
- Service Users
- Infrastructure: The communication links, power sources
- Opponents, which may cause altering the functionality or performance of the system: hackers, malicious software [5].

The attribute mentioned earlier, maintenance, involves repairs and modifications to the system during its lifetime. Various forms of maintenance are summarized in Fig. 2 [6].

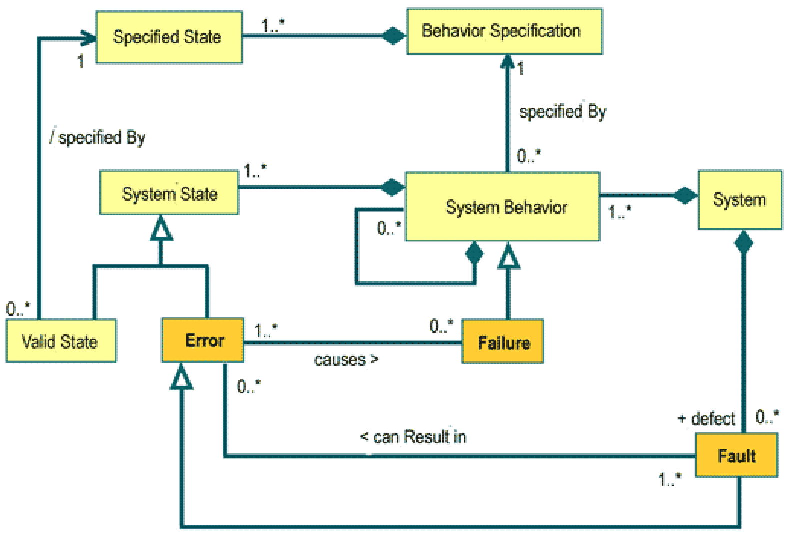


Fig 3 UML Diagram: relationship between faults, errors and failures [7]

It should be noted that repairing and tolerating errors are related concepts. The distinction between error tolerance and maintenance is that maintenance involves an external agent such as a repair or test equipment. Reparation can be seen as an activity tolerance of errors in a system incorporating the subject system repairs, and people carrying out repairs.

The next sub-chapter describes the main threats which jeopardise the security and dependability of systems, and examines ways in which they can be avoided - what can be done so that the negative effect on the system can be reduced. The three major threats are: faults, errors and failures.

1.1.3 The relationship between faults, error and failures

In the definition of fault tolerance there is the presumption that there is a specification which describes the correct system behavior.

A *failure* occurs when an active, functional system deviates from specified behavior. The cause of failure is called *error*. The error is an invalid state of the system, not allowed in the specification of behavioral state of the system. The error itself is the result of a *fault* in the system. In other words, a fault is the root cause of failure. This means that the error is only a symptom of a fault. A fault will not necessarily become an error, but the same fault can result in more errors. Similarly, a single error can lead to multiple failures.

These concepts are illustrated by a UML class diagram in Fig. 3 [7].

Faults, errors and failures operate according to a specific mechanism, known as the *"Fault-Error-Failure Chain"* [4]. As a general rule, a fault, when it is enabled, may result in an invalid state caused by an error which in turn can cause another error or a failure (which is a noticeable deviation from specified behavior at the system boundaries).

$$\begin{tikzcd}
\text{failure} \arrow[r] & \text{fault} \arrow[r] & \text{error} \arrow[r] & \text{failure} \arrow[r] & \text{fault} \\
\text{event} \arrow[r] & \text{cause} \arrow[r] & \text{state} \arrow[r] & \text{event} \arrow[r] & \text{cause}
\end{tikzcd}$$

Fig 4 Fault-Error-Failure Chain

Fig. 4 shows the temporal or causal relationship between faults, errors and failures. The arrows express that, by propagation, faults can cause errors, which in turs, can cause system or component failures. It should be noted that the chain propagation and thus instantiation can occur through interactions between components or systems, the composition of components in a system and creation or modification of a system.

An example of how errors can affect the functionality of a system is the ATM model. A person who want to make a transaction with his card presumes those money will get to the destination, without any attacker finding out the personal details. That ATM machine is using an encryption algorithm to secure the data which is being tranzactioned. If an error would manifest in the algorithm, the security of the whole operation would be compromised. Hence, it becomes obvious the need to periodically test that algorithm and detect an error as early as possible, so to avoid the complete failure of the system when it is too late. The nucleum of this idea was built around Heinz Bonnenberg's thesis [20] about securing VLSI equipments.

The main role of testing is detection of faults in general, just the presence of some specific fault or determination and correction of errors in design or even the testing procedure.

Optionally, the design can be made so that it incorporates testing facilitates. During early years, design and testing were separated. The final quality of tests was determined by keeping track of the number of defective parts shipped to the customer. Defective parts per million (PPM) shipped was a final test score.This approach worked well for small-scale integrate circuits [116].

During the 1980s, fault simulation was used, but this couldn't improve the circuit's fault coverage beyond 80%. Increased test cost and decreased test quality lead to Design for Testability (DFT) engineering. Various testing methods and ad-hoc testability measures were tried along the years to improve the testability of a design, but couldn’t go beyond 90% [63]. The most popular method up to date is Structured DFT and especially Scan Design, but more details about these notions and testing theory fundamentals and examples will be given in Chapter 3.

1.1.4 Ways of achieving dependability and security

A system cannot be fully trustable in any conditions and in relation to all aspects taken into account [7]. So, in order to be useful, the required dependability level and the properties the system is expected to have must be done explicitly. Mere knowledge of the threat is not sufficient to maintain a good system in operation. Various ways of combating them have been developed in recent years, perfectened because with the spread and diversification of threats has increased and the number of methods of control.

The four main methods that result from wrong-error-Failed Chain have been presented in Section 2.2.2. These are: prevention, removal, prediction and tolerance errors [6].

Preventing errors requires preventing errors located within the system. This is achieved through the development of sound methodologies and implementation techniques. There are some hardware methods that make use of active protection, such as "active locks (metal mesh covering the entire system) or detector voltage sources - to detect any abrupt changes in the system that may be a "glitch" (External wrong fault causing transitions of flip-flops). Passive protection includes cycles injecting random "dummy", encryption and memory buses, frequency generated internally unstable, however more difficult for an attacker to correctly understand the internal functioning of the system and thus develop effective redress schemes. Removing errors can be divided into two sub-categories:

- Removal during development - requires such a process so that errors can be detected and removed before the system is used. Once put into use, the system must register faults and removed during the maintenance cycle.
- Removal during use.

Prediction of errors deals with the prediction of the errors which are very likely to occur. Fault tolerance involves the development of mechanisms by which these systems can provide desired services even in the presence of faults, but without these services being somewhat affected.

Dependability methods aim to reduce the number of failures of a system presented to the user. Failures usually appear in time and it is useful to understand how that is measured and how often they occur, so that effectiveness of resources can be measured.

1.2 General Notions of Information Security

Securing information can be treated in two ways. One is the protection of processed data from intruders that can occur in various destructive purposes. The second relates to protection of information circulating on the means of communication [7].

Cryptography, the science considered secure communications systems the information conveyed includes the sub-domains:

• **Cryptography** - the branch of mathematics that deals with information security and authentication and Restriction of access to a computer system. In their implementation using both mathematical methods (taking advantage, for example, the difficulty of factorization large numbers) and quantum encryption methods. The term "cryptography" is composed of the Greek words *kryptós* (hidden) and *gráfein* (write).

- **Cryptanalysis**, and cryptographic analysis, dealing with broken processes without knowing the encryption procedures used in the encryption stage. It occurs on the public channel transmitting secure unruly cryptogram used to reconstruct the source processes. The attack on the encryption processes can be found at the level of recipients. They can be identified in various forms of intelligence infiltration in the state.
- **Steganography** in charge of securing private messages by using methods such as micropunctul, sympathetic inks, notices in newspapers advertising specialty predetermined understood. Such methods, however, belong to the past.

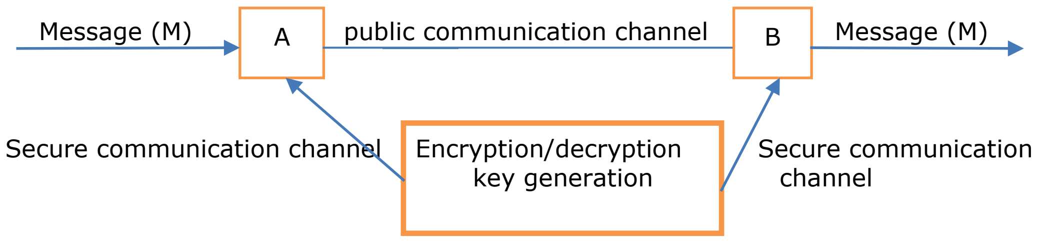

Message (M)
A
public communication channel
B
Message (M)
Secure communication channel
Encryption/decryption key generation
Secure communication channel

Fig 5 The principle of classical securing

The main classical cipher types are: [3]

- Substitution ciphers: the process by which each letter or group of letters in the message source is replaced by a new letter or group of letters.
- Transposition ciphers: encryption process, which relies on changing the order in which characters appear in the message source. All the rules used for this purpose is the cipher key or encryption (such as "Voyager" gets "ovayegr 'a trivial rearrangement scheme [8])

Conventional systems make use of an encryption key that is turned in one clear message visually impaired called Cryptogram, which is transmitted to the recipient on a public channel of communication. The public channel is a channel of communication understood unprotected especially towards outsiders anyway considering that they do not have the necessary decryption key. Only the recipient will have the key.

Fig. 5 presents the principle of safeguarding in a traditional, intuitive form, taken from [36]. For a variety of keys, only a couple of them can be used and from these only one at a time. As apparent from the figure these keys are generated in a particular place and the disadvantage of conventional processes is precisely that these keys must be transmitted to all users on the secure channel of communication [7].

The drawback of the classical cryptosystems is that they cannot mediate between the receiver and sender as to verify if a message was transmitter and what this message is. Even with the Key management issue, as it is hard to send the secret key in a secure manner, the user authentication and message integrity can be achieved but only the receiver can verify these.

1.3 Modern Criptography

1.3.1 Public-key cryptography

Having in mind the afore-mentioned downsides, new directions were investigated, emerging from Diffie and Hellman's research which was mentioned in the introduction. They proposed a novel type of cryptosystem which was later called public key crptosystem (PKC).

Before going further to detail on this, it must be said that the basic cryptographic building blocks, the atoms out of which all other cryptographic constructions are produced are the primitives. The basic symmetric key building blocks are considered in [117] to be block ciphers, hash functions and stream ciphers, as well as basic public key building blocks such as factoring, discrete logarithms and pairings. Modern cryptography then takes these building blocks/primitives and produces cryptographic chemes out of them. Besides there afore-mentioned building blocks, we must also metion also Message Authentication Codes (MAC), which are symmetric-key cryptosystems that aim to achieve message integrity. Most commonly used designs fall in one of two categories: block-cipher based schemes and hash function based schemes.

Asymmetric cryptography is a type of cryptography in which the user has a pair of keys, one public and one private. Using public key someone can encrypt a message that can be decrypted only with the paired private key. Mathematically, the two keys are related, but practically they are not derived from one another. A very appropriate analogy for the mailbox process presented in [8]. Anyone can put an envelope in someone's mailbox but only the mail owner of the envelope has access to it. Asymmetric cryptography is also called „public key cryptography”.

The two major branches of asymmetric cryptography are:

• *Public key encryption* - a message encrypted with a public key cannot be decoded only using the corresponding private key. It is used to ensure confidentiality.

• *Digital signatures* - a message signed with the private key can be verified broadcaster by anyone with access to this key, thus ensuring authenticity of the message [28].

Public key encryption schemes are seldom used to encrypt messages but more often to secure by encryption symmetric keys for future bulk encryption.

A major problem in using this type of encryption is the lack of confidence in the accuracy and authenticity of public key. -Normally the problem is solved using public key infrastructure (PKI) in which one or more persons provide authentic key pair. Another approach used by PGP (Pretty Good Privacy) is the concept web of trust.

1.3.2 Symmetric-key cryptography

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, in some cases, in which their keys are different, but related so that can easily be computed). This was the only kind of encryption publicly known until June 1976.

The modern study of symmetric-key ciphers relates mainly to the study of block ciphers and stream ciphers and to their applications. A block cipher is a modern version of Alberti's polyalphabetic cipher: block ciphers take as input a block of plaintext and a key, and output a block of ciphertext of the same size. Since messages are almost always longer than a single block, some method of combining together successive blocks is required. Several have been developed, some with better security in one aspect or another than others. They are the modes of operation and must be carefully considered when using a block cipher in a cryptosystem [27].

The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs which have been designated cryptography standards by the US government (though DES's was no longer used after the AES was adopted). Many other block ciphers have been designed and released, with considerable variation in quality. Many of them have been broken, such as FEAL.

Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state which changes as the cipher operates. That internal state is initially set up using the secret key material.

Cryptographic hash functions are a third type of cryptographic algorithms. They take a message of any length as input, and output a short, fixed length hash which can be used in (for example) a digital signature. For good hash functions, an attacker cannot find two messages that produce the same hash. MD4 is a long-used hash function which is now broken; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family is an imporved version of SHA-1, but it isn't yet widely deployed, and the U.S. standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST's overall hash algorithm toolkit" [24].

1.3.3 Symmetric-key algorithms

The main advantage of public-key encryption-systems is that it is ideal for transmitting information through insecure channels, but on the other side, they are much slower than symmetric systems [30]. The later can transmit larger volumes of data and can be used as a basis for constructing various encryption mechanisms, such as: pseudo-random number generator, function generator leakage scheme signature.

The main symmetric-key algorithms are: Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA) [26] - first algorithm proposed to replace DES's and Advanced Encryption Standard (AES) known as the Rijndael, which is the official successor and Triple DES algorithms DES [19]. Knowledge of these algorithms is needed to understand the cipher which makes the subject of this thesis. Because DES had become vulnerable because of a too small key length, NIST (National Institute of Standards and Technology) recommended the usage of 3DES, an algorithm that is essentially applying three times the DES. Although 3DES algorithm proved to be strong, it is relatively slow in software implementations, which is why NIST has submitted an application in 1997 with various proposals which could replace it.

The winning proposal was AES, a symmetric block cipher with 128-bit length text and key based on Galois Field GF (2^8), leading to a lower consumption in modern computers [9]. In June 2003 the US government decided that AES can be used for classified information also. Until the level SECRET, all three standardized key lengths – 128, 192 and 256 bit can be used. The TOP-SECRET information (highest classification level) can only be encrypted with 256-bit key lengths. Still there are numerous attacks demonstrated to it [29].

For example, for the extension of keys algorithm, in [10], the energy leaks which take place in the cryptosystem were demonstrated through a simple analysis of the energy (SPA – Simple Power Analysis). Observing the energy leaks leads to inducing the secret cryptographic keys. As SPA attacks are easy to be done and extremely efficient, in [11] there were proposed a series of protection measures in implementing the cryptosystems on smartcards and other similar devices.

DPA (Differential Power analysis) attacks take secret information from the statistical calculations of a set of traces of energy consumption.

Papers [5],[8],[10],[11], [12] demonstrated attacks against all the above algorithms, especially AES, considered the most secure algorithm in early 2000s, which was and still is used to protect everything from top-secret government documents to online banking transactions.

In 2001 there was the first single-key attack on the full AES cipher which is (very slightly) faster than brute force. It introduces a technique known as biclique cryptanalysis to remove about two bits from 128-, 192-, and 256-bit keys [7]. Given sufficient time, a brute force attack is capable of cracking any known algorithm.

There was a new direction of lightweight algorithms, with fast key-schedulers and substitution boxes (S-boxes) based on purely algebraic construction [13]. However the trend is not necessarily the safest, most of published attacks on block ciphers algorithms operate with just simple a key programmer, a good example being Muller's attack on the Khazad cipher [14]; algebraic substitution boxes were just an aid for algebraic attacks – as Courtois-Pieprzyk state in [15].

Based on the all this, it becomes clear the motivation to develop a serious alternative to the block ciphers.

1.3.4 New Trends and Recommendations in Cryptography

There are organisms that work to improve the resilience of Europe’s critical information infrastructure and networks. The best known are ECYPT and ECRYPT2. Also, the European Union Agency for Network and Information Security Agency released a document [117] which comprise a series of recommendations for algorithms, keysizes, and parameter recommendations, addressing the need for a minimum level of requirements for cryptography across European Union (EU) Member States in their effort to protect personal and sensitive data data of the citizens. As a general rule, they consider symmetric 80-bit security levels to be sufficient for legacy applications for the coming years, but consider 128-bit security levels to be the minimum requirement for new systems being deployed. Thus the key recommendation is that decision makers now make plans and preparations for the phasing out of what we term legacy mechanisms over a period of say 5-10 years. In selecting key sizes for future applications they consider 128-bit to be sufficient for all but the most sensitive applications.

Chapter 2 THE IDEA NXT CRYPTO-ALGORITHM

2.1 A New Generation in Encryption: IDEA NXT

In 2001, in Switzerland, at the request of Crypt Media AG, the company which has the publishing rights for IDEA, Pascal Junod and Serge Vaudenav initiated a new project called FOX which was aimed to be an improved version of both IDEA and AES algorithms by combining the speed of IDEA with the security of AES. In May 2005 MediaCrypt announced its final form under the name IDEA NXT.

IDEA NXT Family is a new generation of symmetric encryption algorithms, flexible and scalable from AG Mediacrypt to help secure governmental data and digital communications.

As the availability of rich media allows for easy and quick Internet downloads, recent reports show that over $3 billion is lost to content piracy each year [16]. Most of this piracy has been due to peer-to-peer audio networks, with video piracy becoming more popular as bandwidth grows. IDEA NXT can be integrated into Digital Rights Management systems and allows for unique dynamic recovery capability that protects system integrity.

Another trend is to move media content beyond its traditional livingroom location, into a home entertainment network and to on-the-go audio/video devices. IDEA NXT gives content owners and distributors additional options to secure the content for transportation and integration. It allows for new business models for content distribution. Its flexible and scalable features please both consumers, as they are given access to their content, and service providers, by securing that content.

IDEA NXT family consists of two block ciphers that have different sizes, key lengths and number of rounds:

• Standard NXT64 (block of 64 bits, 128 bit key, 16 rounds) that is compatible with IDEA and Triple DES

• Standard NXT128 (block of 128 bits, 256 bit key, 16 rounds) compatible with AES.

Variations of the cipher can also be constructed out of the Standard Version: 0-256 bit keys, amaximum of 255 rounds, and Standard tables can be replaced by individual tables (sbox matrix permutation) which are successfully used in case of attacks by loading new tables in the system, thus creating a dynamic method of system recovery. Variable number of keys provides the possibility for each user to choose the desired length of password, while the flexibility in the number of rounds translates into more secure applications and allows finer adjustment of the user's performance and security levels according to applications [16].

MediaCrypt published an article [16] which contains a comparison between IDEA NXT and AES, the first winning in a couple of extremely important chapters. It is more resistant to known attacks, has a stronger key planner and not least, has less underlying hardware, one aspect that is becoming increasingly more important.

The main reaon for which I considered studying this algorithm in more detail is the enhanced security it seems to offer, as, to the best of my knowledge, there is no proved attck to brake it, unlike AES, for which a number of successful attacks were presented in the literature. Also, the fact that for IDEA NXT the process of decryption is almost the same as the encryption one (since it is based on a Feistel scheme) makes it a perfect fit for situations in which bi-directional data needs to be transmitted in a secure manner.

2.2 Mathematical Structure of IDEA NXT

The IDEA NXT algorithm is based on the Lay-Masey scheme and the round functions are of type *Substitution-Permutation Networks* (*SPNs*) based on the Feistel scheme.

The 64-bit version of IDEA NXT represents the 'n-1' times iteration of a round function called lmor64, followed by applying a slightly modified function called lmid64. For decrypting, lmor64 is replaced by lmio64, function which uses the inverse of the OR ortomorphism. All these functions, based on the Lay-Massey scheme have an output on 64 bytes and two input parameters on 64 bits: the plain text and a sub-key, also known as *round key*.

In the 128-bit version of the cipher, *lmor64*, *lmid64* and *lmio64* are replaced by *elmor128*, *elmid128* and *elmio128*. The major difference is that the NXT128 functions use the extended Lay-Massey scheme combined with 2 ortomorphisms. The ortomorphism represents a Feistel scheme on a single round which has the identity function as a round function.

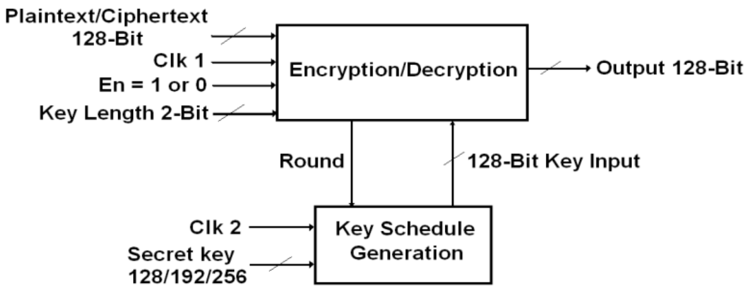


Fig 6 Top-level scheme of IDEA NXT\

The main function of NXT64 is f32, composed of 3 main parts: a substitution part, sigma4, a diffusion part mu4 and a round key addition part. The substitution part is composed of the *sigma8* and *mu8* functions which are made of 4, respectively 8 parallel calculations of non-linear bijective mapping. A substitution box (sbox) essentially is a look-up table filled with predefined values. The diffusive part (mu32/mu64) is a linear multipermutation defined of Galois Field GF(2^8) [13].

The key is processed by a Key Scheduler module which performs a four-layer encryption of its own before providing the obtained key value to the data encryption process itself.

The key scheduling algorithm is composed of four parts: one padding part, denoted P, which extends K(k) to *ek* bits, a mixing part M, one diversification part D whose kernel is a linear shift register noted LFSR (Linear Feedback Shift Register) and a non-linear part NLx. Because the non-linear part can operate on 64, 128 or multeiple of 64 bits, the Key scheduler is said to have three (corresponding) versions.

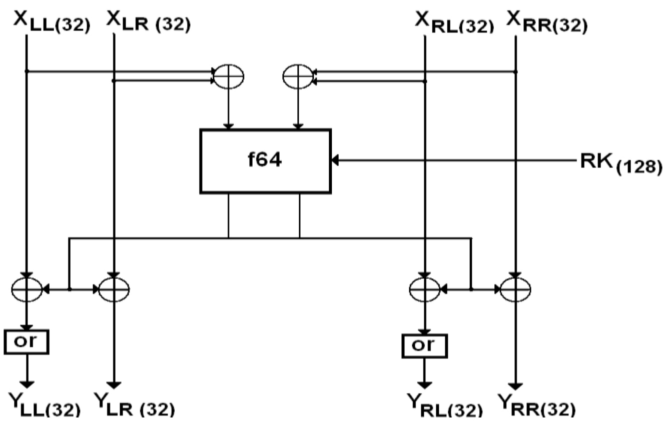


Fig 7lmor64 algorithm

Fig. 8 shows the main operations of the key scheduler algorithm. KS128 takes the following parameters as input: the key *k* which is *l* bits long, 0<=l<=256 and a round number *r* representing the number of rounds and return a number of *r* sub-keys of 128 bits length.

The padding is used to extend the key, if its length differs from the standard length of 'ek' bytes, by concatenation with the pad constant.

The main part of the mixing entity is a Fibonacci- type recursion whose first 2 elements have pre-defined values: *mkey(-2)=0x96A and mkey(-1)=0x76*.

The padding, mixing and diversification parts are efficiently implemented when the key length is fixed, for NXT128 on 256 bits. The sequences from the output of the LFSR can be pre-calculated and stored in the ROM for easy access to them [21].

There are three versions of the Key Scheduler algorithm, chosen depending on the key length and of the considered member of NXT family.

The non-linear step is itself made of multiple parts: substitution (which uses 4 parallel sigma4/sigma8 processes), diffusion (composed of four times mu4/mu8 functions plus mixing) and mixing. The result is obtained from various combinations of XOR operations between the 4 parts obtained by splitting the input vector.

The diversification part takes the key computed in the mixing part, denoted mkey, having *ek* bits length, the total number of rounds *r* and the current round number, *i,* 0<i<r, and modifies mkey with the help of a 24-bit LFSR. The mkey is seen as an array of [ek/24] 24-bit values mkeyj(24), with 0<=j<=[ek/24]-1 concatenated with one residue byte mkeyrb(8) (if ek=128) and is modified according to (1):

$dke_{yj}(24) = mke_{yj}(24)\ XOR\ LFSR((i-1)*floor(k/24)+j,\ r)$ (1)

for 0<=j<=(floor)(ek/24)-1;

dkeyrb(8) value is obtained XOR-ing the most 8/16 significant bits of LFSR((i-1)*(ceiling)(ek/24)+(floor)(ek/24),r) with $mkey_{rb}(8)$ /$mkey_{rb}(16)$ respectively. The remaining 16 (8) bits of the routine are discarded [22].

The kernel of the Key Scheduler is the linear shift register denoted LFSR (Linear Feedback Shift Register) which takes the total number of rounds 'r' and a number of preliminary clocks as inputs and uses an following irreducible polynomial over GF(2^8) for generating the encrypted keys. The remaining 16 (8) bits of the LFSR are ignored.

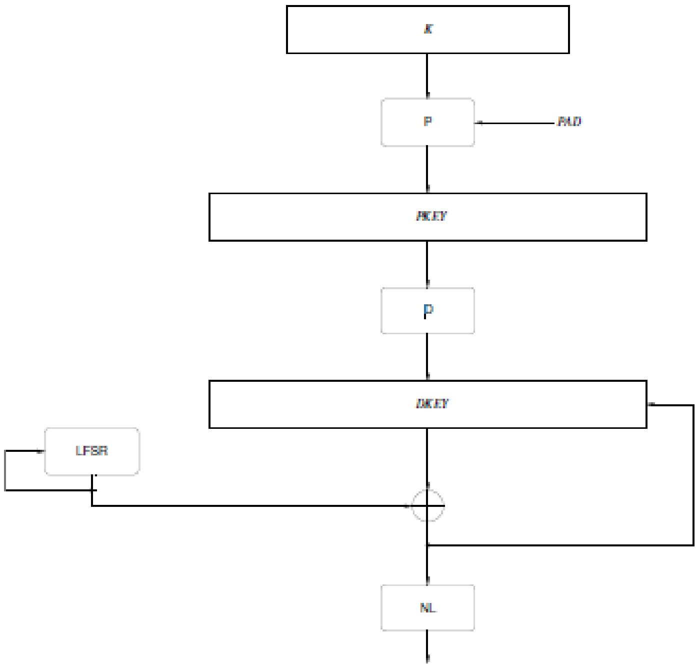


Fig 8 Key Scheduler of IDEA NXT

In essence DKEY value goes through the substitution layer consists of four parallel functions *sigma4* (*sigma8*), a diffusion layer consisting of four parallel functions *mu4* (*mu8*) and a mixing layer called mix64 (mix128 respectively). The result goes through a second layer of substitution, then is reduced to 64 (128) by two bits of the operation by or-exclusive and final value is encrypted by first lmor64 function (*elmor128*), where the value of the sub-key is given by the first half the DKEY and then lmid64 function (*elmid128*), the sub-key is in this case the second half of DKEY. The result is a round key 64 (or 128) number of bits corresponding to round i.

2.3 Speed Optimizations for IDEA NXT

2.3.1 Algorithm limitations

The IDEA NXT crypto-algorithm was thought to be extremely competitive in all the relevant security areas while still having no major speed penalties [2], and it can be said it succeeds when compared to its main competitors from the time it was created (DES, IDEA, AES). The comparison I made in a previous paper [17] shows both the 64-bit and 128-bit key lengths of IDEA NXT performed better in terms of throughput than the similar versions of the AES encryption algorithm. I implemented the 128-bit, 192-bit and 256-bit versions of the NXT algorithm in a pure hardware manner, using the Verilog modeling language, and compared the execution times with the execution times of similar implementations of the DES, IDEA and AES crypto-algorithms. The results presented in that paper showed IDEA NXT performed well on the large versions, but its small-length versions could without a doubt be improved, when compared to the widely-used 64 and 128-bit versions of AES.

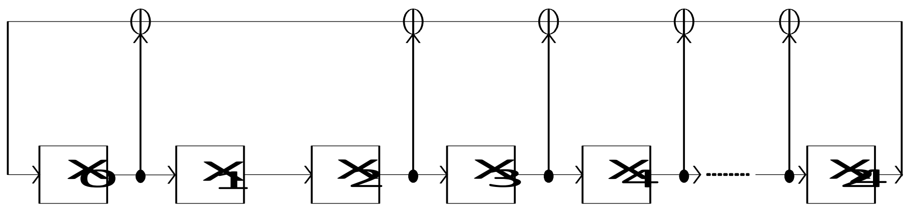


Fig 9 A 24-bit Linear Feedback Shift Register

The hardware implementation of all versions of IDEA NXT takes longer to execute compared to DES, IDEA and AES mostly because of the complex Key Scheduler and repeated rounds of encryption. So, in order for it to be competitive at this chapter also, a solution needed to be found to increase the speed either at the entire algorithm level, either at the key generator level, which in turn improves the overall speed.

Our goal was to find a way to speed-up the algorithm without essentially modifying its structure. One of the ideas was to improve the way keys are provided to the encryption layers, as IDEA NXT's Key Scheduler was proved to be a very complex process, translated into clear advantages in terms of security, but with a price to be paid in terms of small key agility. The time needed to compute the keys is about the time needed to encrypt six blocks of data in case of NXT64 or 12 blocks of data in case of NXT128 [13].

On a close analysis a time penalty in the diversification layer of the Key Scheduler was observed, where a round key (also called subkey) needs 6 clock cycles to be generated. The explanation for this is given in the following paragraphs.

If we consider the 64-bit version of IDEA NXT, with a key length of 128-bits, according to relation (1) from Chapter 2.2, 24 bits from the round key are processed at each iteration, except for the last iteration in which only 8 bits are processed. Since 128 equals 5 times 24 plus the remaining 8, it is concluded that 6 clock cycles are needed to generate a single subkey.

The 24-bit LFSR used in the Key Scheduler is represented in Fig. 10.

A linear feedback shift register is essentially a sequence (a_i), $i \in \mathrm{N}$ satisfying the recursion:

$$a_{i+n} = \Sigma(c_j * a_{i+j}) \qquad (2)$$

In other words it is as a shift register whose input bit is a linear function of its previous state. The most commonly used linear function of single bits is XOR. Thus, an LFSR is most often a shift register whose input bit is driven by the exclusive-or (XOR) of some bits of the overall shift register value [15].

For the case of IDEA NXT, as previously mentioned, the input vector is divided into 24-bit vectors, leaving 16-residual bits. The entity takes as input ports: the number of preliminary clocks (c), the vector and the total number of rounds (r), and generates an output port of 24 bits which will become part of the diversification key (dkey). The algorithm uses the following irreducible polynomial over GF(2^8) for generating the encrypted keys:

$$P(x) = x^{24} + x^{4} + x^{3} + x + 1 \qquad (3)$$

The algorithm is described formally below:

```
/ * Initialization */
reg = 0x6A || r(8) || NOT (r(8))
/* Pre-clocking */
for (p = 0; p < c; p = p-1)
   if (reg AND 0x800000) != 0x000000 then
           reg = (reg << 1) XOR 0x00001B
   else reg = (reg << 1)
end if
 end for
Output reg
```

The LFSR construction assures that the subkey generation process can be computed in the encryption as well as in the decryption direction with no speed loss.

2.3.2 Speedup Solution for IDEA NXT's Key Scheduler

Having all this in mind, the next step was to dissect the structure of the LFSR and analyze in detail each part of its mathematical structure, to see which part introduces the greatest overhead and what component is best to modify in order to improve the overall performance.

Let us denote the polynomial representation of IDEA NXT's LFSR as P(x), its equation is defined by relation (4):

$P(x) = a_{24} X^{24} + ...+a_4X^4 + a_3X^3 + a_1X + a_0$ (4)

Since its generator polynom is the one in relation (3), we can consider one of its radixes to be:

$X^{24} = X^4 + X^3 + X + 1$ (5)

The process of generating a round key by the Linear Feedback Shift Register would mean to shift the register with one position each iteration, so six shiftings would be needed for a complete generation of a subkey. This shifting translates into multiplying the P(x) polynom by X each clock cycle.

Our proposal is to not lose 6 clock cycles for the generation of a single round key, but to directly provide a round key in a single clock cycle. The solution for this is to directly multiply the P(x) polynom with X^6 each clock cycle, as follows:

(modulo(modulo(modulo(modulo(modulo(modulo(P(x)X))X)X)X)X)X) = $P(x)X^6 = a_{24}X^{30} + a_{23}X^{29} +a_{22}X^{28} + a_{21}X^{27} + a_{20}X^{26} + a_{19}X^{25} + a_{18}X^{24} + ...+ a_3X^9 + a_1X^7 + a_0X^6$

If we replace X^{24} by relation (5) we obtain, P(x) would be:

$P(x) = a_{24}X^{30} + a_{23}X^{29} +a_{22}X^{28} + a_{21}X^{27} + a_{20}X^{26} + a_{19}X^{25} + ...+ a_0X^6 + a_{18}X^4 + a_{18}X^3+a_{18}X+a_{18}$ (6)

Thus a_{18} is the new a_0.

Further replacing X^{24} with its radix from (5) in all places we can in P(x) we obtain a new definition for the Linear Feedback Shift Register's output:

```
new_reg = {reg[18], reg[18]+reg[19], reg[19]+reg[20],
              reg[18]+reg[20]+reg[21],reg[18]+reg[19]+reg[21]+reg[22],
      reg[19]+reg[20]+reg[22]+reg[23], reg[0]+reg[20]+reg[21]+reg[23],
      reg[1]+reg[21]+reg[22], reg[2]+reg[22]+reg[23],
      reg[3]+reg[23], reg[4], reg[5], reg[6], reg[7], reg[8], reg[9], reg[10],
      reg[11],    reg[12],    reg[13],    reg[14],    [15],    reg[16],    reg[17]};
(7)
```

The above assignation will be used in the round key generation process in Fig. 4, each new key being processed in the same way.

As an intemediar conclusion, this sub-chapter illustrated a method to improve the execution time of the IDEA NXT crypto-lagorithm, which works for all versions of the algorithm independent of the key and text length. The improvement consisted in modifying the LFSR equations used in the round key generation process, as to generate one key per clock cycle instead of one key in six clock cycles. The streams of pseudo-random numbers which make up the round keys are needed in both the encryption and decryption processes of the algorithm, to assure a superior level of security.

2.4 Hardware Implementation for IDEA NXT

The incresed security IDEA NXT offers and its other advantages discussed in the previous chapter are a good indication that this algorithm will start to be widely-used in the next couple of years, since its main competitor, AES, seems to be more vulnerable by day. When this happens, the algorithm will need to be integrated in a cripto-chip or other systems to be used in the field, and from here orginated my idea to implement it in a pure hardware manner, for FPGA devices. There are only two othyer hardware implementations of IDEA NXT up until now, and both are for ASICs, making my implementation the first of this kind – built for FPGAs.

Hardware–based encryption moves the encryption functions inside de hard-disk drive sub-system or crypto-chip, where the operating system doesn't reach them. In this way, these security-critical compo are protected from kits and malware, as stated in [123].

For an efficient design of the algorithm, the hierarchical design, which uses the concept known as "Divide et impera", was chosen. Hierarchical design implies, besides the sectioning the code into modules as simple as possible, design abstractions necessary to assure major, but also different functional targets (speed, performance, energy consumption).

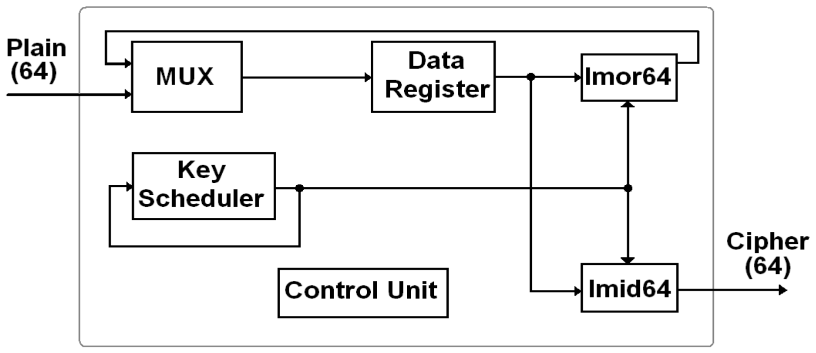


Fig 10 Encryption blocks of IDEA NXT [31]

Having these principles in mind, I have implemented a top-down design imagining a "black-box" system (the algorithm itself) which was systematically broken down into more specific modules.

The top-level entity is the "black-box" taking as inputs the round number and the plain (or encrypted) text, while giving the encrypted (or decrypted) text on the output port.

I adopted a structural approach to the hardware implementation using the Verilog modeling language. This means that all the components of the algorithm are implemented as blocks with input and output signals, which are interconnected according to the mathematical guidelines described in [20] as to achieve the encryption/decryption processes.

IDEA NXT represents the 'n-1' times iteration of a round function called *lmor64*/128, followed by applying a slightly modified function called *lmid64*/128. Fig. 10 shows all the blocks which were defined for the encryption process.

The main blocks are contained in a multiplexor which has two data entries and one command entry. The input signal of the encryption block and the output signal of the *lmor* block are inputs of the MUX.

The control signal will play the role of address entry. The output will be chosen depending on this signal, as follows: at the first iteration the input of the encryption block will be used, whereas for the rest of the iterations the output of the *lmor* block will be chosen. The rest of the blocks shown in Fig. 10 are as follows:

- Data Register – register which holds the result of the last executed iteration (this is also the next SIR which will be encrypted)
- Key Scheduler (KS) – block which produces the key that will further be used by the lmor and lmid in the text encryption
- elmor64/128: transforms a 64 or 128-bit input text and a key into an 64/128-bit output (these will be implemented as ports) with the help of an ortomorphism and the f64 function. It receives at its entry the latest iteration from the data register and the key from the KS. This block will execute the encryption operation r-1 times, where 'r' is the number of rounds. The result is sent back to the multiplexor in order for it to be passed on to the data Register.
- lmid64/elmid128 – encryption block which gets as input the last iteration from the Data Register and the key from the KS. Although it receives at each iteration the result of the last *lmor* operation, lmid won't encrypt what it receives at the entry until the control signal announces it that lmor executed its function r-1 times, and so the finalization operation – lmid- can take place. This block doesn't use an ortomorphism.
- Control Unit – used to calculate the number of iterations executed by lmor, so that at the 'r'-th iteration, lmid could be executed.

The decryption process is very similar to the encryption one, the only difference is that elmio64/128 is used instead of lmor64/elmor128 [17].

The experimental results showing the Time comparisons between small-bit versions of DES, IDEA, AES and the original and modified versions of the IDEA NXT crypto-algorithm will be detailed in Chapter 5.

Chapter 3 VLSI TEST PRINCIPLES AND ARCHITECTURES: DESIGN FOR TESTABILITY

When a crypto-algorithm is used in the field, or integrated in a crypto-chip, errors can occur and cause malfunctions. A good example of how errors can affect the functionality of a system is the ATM model. A person who want to make a transaction with his card presumes those money will arrive at the destination, without any attacker finding out the personal details of the card or the transaction. ATM machines use an encryption algorithm to secure the data which is being transactioned. If an error would manifest in the algorithm, the security of the whole operation would be compromised. Hence, it becomes a matter-of-course the need to periodically test that algorithm and detect potential errors as early as possible, so to avoid the complete failure of the system by only taking actions when it's too late. The nucleum of this idea was built around Heinz Bonnenberg's thesis [20] about securing VLSI equipments.

The main role of testing is detecting faults in general, confirming the presence of some specific fault or determinating and correcting errors in the design or even in the testing procedure. Optionally, the design can be made so that it incorporates testing facilitates.

During early years, design and testing were separated. The final quality of tests was determined by keeping track of the number of defective parts shipped to the customer. Defective parts per million (PPM) shipped was a final test score. This approach worked well for small-scale integrate circuits [2].

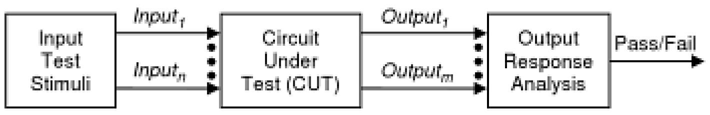


Fig 11 Basic testing approach [39]

During the 1980s, simulation of faults inside the system was a widely-used technique, but this couldn't improve the circuit's fault coverage beyond 80%. Increased test cost and decreased test quality lead to Design for Testability (DFT) engineering. Various testing methods and ad-hoc testability measures were tried along the years to improve the testability of a design, but couldn't go beyond 90%. The most popular method up to date is Structured DFT and especially Scan Design, but more details about these notions and testing theory fundamentals and examples will be given in later in the thesis.

As was denoted in the introduction of the thesis, any system must be tested for the various fault and errors which can occur in the system. It is well know that, if a faulty behavior is manifesting in a system, the further it propagates, the more damage it does and the harder it is to repair it. Not to mention the cost of repairs increases. Electronic testing includes IC testing, PCB testing and system testing at the various manufacturing stages and, in some cases, during system operation. Testing must point out errors, faults and can also be used to analyze their causes.

Testing usually implies applying a set of stimuli to the input ports of the *Design Under Test (DUT)* followed by analyzing the responses at the output ports, as can be seen in Fig. 11.

A VLSI design can be tested at various levels of abstraction, ranging from the physical (transistor) level, to the register-transfer level and the architecture level [110]. Testing can be done either online – concurrently with the system's operation, or offline – when the system or just a part of it is taking out of operation, or a combination of these two, like for example online testing discovers an error and offline mechanisms are used for diagnosis of the failed component.

The main metrics for testing are:

- **Realiability**: the probability that the system will operate normally until time t: $P(T_n>t) = e^{-\mathcal{E}t}$, where $\mathcal{E}$ is the **failure rate** [49]. Since a system is composed by various components, the overall failure rate is the sum of all individual failure rates of the system's components. The following key concepts must be known:
- **Mean time between failures**: MTBF = $\int e^{-\mu t} dt = 1/\mathcal{E}$
- **Repair time**: MTTR = 1/ μ, because $P(R > t) = e^{-\mu t}$
- **System availability** = MTTB / (MTBF + MTTR).

To test a circuit usually we apply a set of input patterns called **test vectors.** When applying all possible 2^n input vectors to a DUT with n inputs we do an **exhaustive testing,** which, if it passes, means that no functional fault affects its structure [52].

3.1 Design for Testability Architectures and Techniques

3.1.1 Design for Testability Principles

In [17] the technological challenges in the VLSI industry are presented. The authors highlight the importance of the VLSI testing, taking Moore' law as a starting point. The long-term challenges especially on the nanometer technology encompass a large spectrum of testing technology trends, including the development of new DFT and DFM (Design for Manufacturing) methods, Device Under Test (DUT), Automatic Test equipment (ATE) interface, automatic Test Pattern Generation (ATPG), speed testing with increased core frequencies, multi-GHz input / output protocols and other means to reduce manufacturing costs and increase reliability and yield.

The yield of a manufacturing process is defined as the number of acceptable parts divided by the number of parts fabricated. Methods to keep this metric at an acceptable rate are called design for yield (DFY). In order to properly measure these, the notions and metrics of system reliability and availability have to be properly understood and analyzed.

Since its beginnings following World War II, reliability theory has grown into an engineering science in its own right. The early development is discussed in Chapter 1 of Shooman's thesis [18]. Much of the initial theory, engineering, and management techniques centered about hardware; however, human and procedural elements of a system were often included. Since the late 1960s the term *software reliability* has become popular, and now reliability theory refers to both *software* and *hardware reliability* [19].

The reliability of a system is given by the reliability of its individual components, which is measured by the level of defects, noise level, failure rate [20].

So the conventional approach to reliability is to decompose the system into smaller subsystems and units. Then by the use of combinatorial reliability, the system probability of success is expressed in terms of the probabilities of success of the elements. Then by the use of failure rate models, the element probabilities of success are computed. These two concepts are combined to calculate the system reliability. When reliability or availability of repairable systems is the appropriate figure of merit, Markov models are generally used to compute the associated probabilities [21]. Often a proposed system does not meet its reliability specifications, and various techniques of reliability improvement are utilized to improve the predicted reliability of the design.

A very structured approach for designs that contain a large amount of sequential logic is required as part of a methodical design for testability (DFT) approach [26]. Initially, many ad hoc techniques were used for bettering testability. DFT ad hoc techniques consisted of making local modifications to a circuit to improve testability. Even if the improvements are visible, their effects are local and not systematic. Furthermore, these techniques are not methodical, in the sense that they have to be repeated differently on new designs, often with unpredictable results [27]. Also, because of their ad hoc nature, you cannot predict how long they would take to implement.

The need for a structured, methodical testing process which could easily be encompassed in the design flow became evident. Furthermore, structured DFT techniques are much easier to automate. The most important structured DFT technique is the **Scan design.** It makes DFT implementation easier to be managed and it achieves high fault coverage.

"Scan design, the most widely used structured DFT methodology, attempts to improve testability of a circuit by improving the controllability and operability of storage elements in a sequential design" [6]. Usually this is achieved by converting the sequential design into a scan design with 3 modes of operation: normal mode, shift mode, and capture mode. Circuit operations with associated clock cycles conducted in these 3 ways are referred to as normal operation, shift operation, and capture operation, respectively.

In the normal mode of operation no signal is on, so it has '1' logic value and the scan design runs in the functional mode. In the shift and capture modes, a test mode signal denoted TM is used to operate on all features which can simplify the diagnosis, testing and debugging processes or diagnosis tasks, better the fault coverage and improve the functioning of the components or tests blocks of the CUT [16]. The various modes of operation can be distinguished by integrating more test signals or clocks in the design.

3.1.2 ATPG

Test generation is defined in [6] as "the task of producing an effective set of vectors that will achieve high fault coverage for a specified fault model, which will uncover any defect in a chip". This task is a very challenging one, and mostly relies upon the Automatic Test Pattern Generation (ATPG).

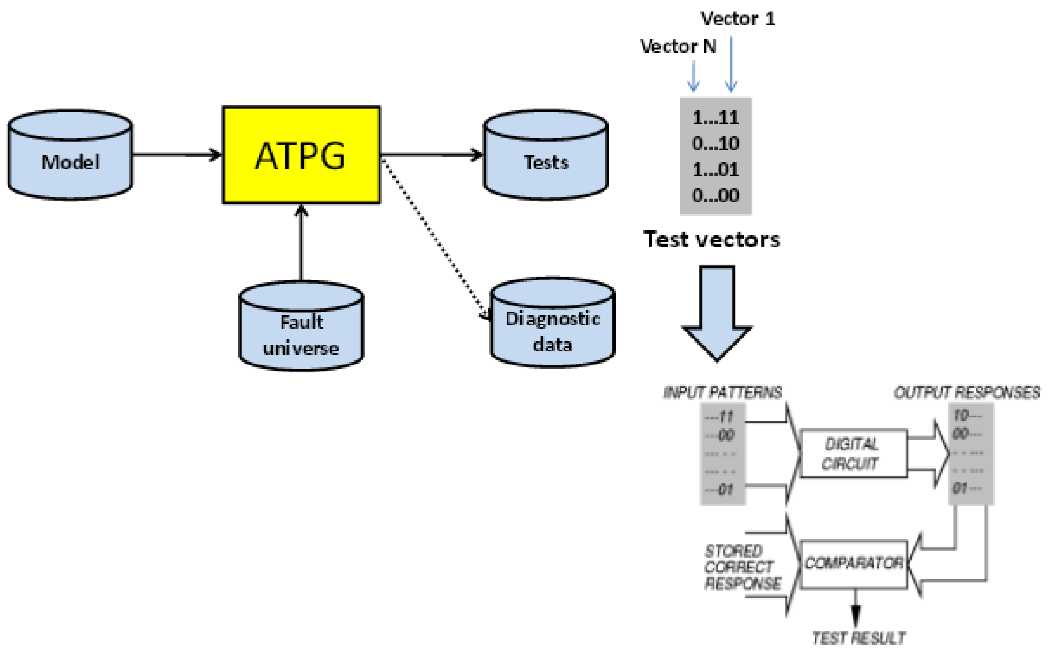


Fig 12 ATPG principles [31]

This is a serious alternative to the Design for Testability (DFT) techniques discussed in the previously and therefore much effort has been invested in improving this technique. The following paragraphs will present the theory behind the design of an ATPG and the learning mechanisms which can make ATPG more performant.

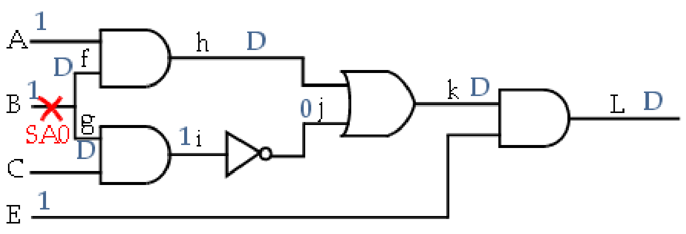


Fig 13 Sample circuit with stuck-at faults

ATPG is defined in [31] as an electronic design automation method / technology used to find an input (or test) sequence that, when applied to a digital circuit, enables automatic test equipment to distinguish between the correct circuit behavior and the faulty circuit behavior (caused by defects). The generated patterns are used to test semiconductor devices after manufacture, and in some cases to assist with determining the cause of failure.

The metrics for measuring the effectiveness of an ATPG algorithm is given by the amount of defects, or faults that are found and the number of test patterns which are generated. The test quality is considered to increase with the number of fault detections found and test application time is higher with more patterns applied. Another metric taken into account is efficiency, and this is different depending on the on the fault model under consideration, the type of circuit under test (full scan, synchronous sequential, or asynchronous sequential), the abstraction level of the CUT (gate, register-transfer, switch), and the test quality. Other advantages of ATPG are: it can find redundant circuit logic, it can prove an implementation matches another, achieves fault coverage grater then 98%. The testing principle for an ATPG is shown in Fig 12.

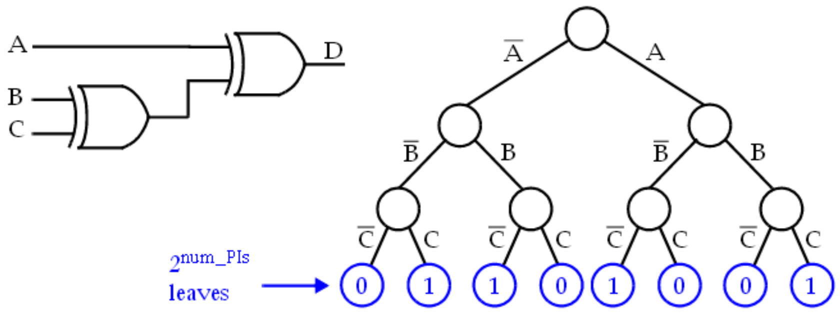


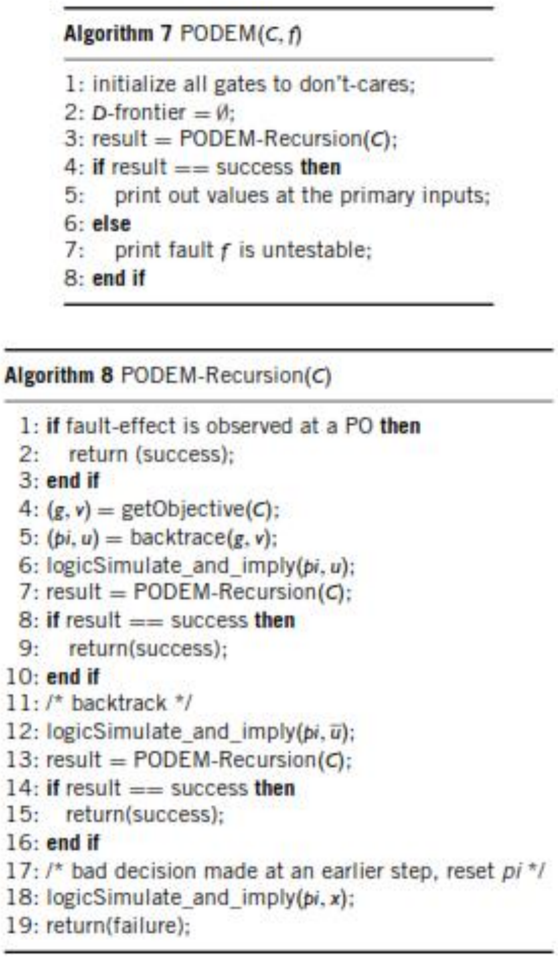
Fig 14 Binary Decision tree

I applied a custom ATPG method, called Path Sensitization Method to the registers used in both datapath and key scheduler blocks of the IDEA NXT hardware implementation scheme, to check for stuck-at faults. This method consists of 3 steps: fault activation (force tested node to negated fault value), fault propagation, line justification (justify internal signal assignments made to activate and sensitize faults).

The second and third step might be conflicting if, for example, different values are assigned to the same signal and require backtracking. In the sample circuit in Fig. 13, took from the above-mentioned block-scheme, if I target B stuck-at 0, fault activation requires *B=1, f=D and g=D*. For the fault propagation case, there are 3 possible scenarios: paths *f-h-k-L*, *g-i-j-k-L* and both. The first path requires *A*=1, *j*=0 and *E*=1. The decision here is to what path is best to synthesize. An approach would be to sensitize more paths at a time. I tried paths f-h-k-l and g-i-j-L simultaneously, but it lead to no result, as the D-frontier dissapears.

Another approach is to do use a backtracking algorithm in order to go to the last decision situation and make a different decision. I tried it for the g-i-j-L path, where instead of going from j to L I went through K instead. This way, I fould the best path to be g-i-j-k-L.

Decision trees are used to facilitate the implementation and visualisation of a backtracking algorithm, like the one in Fig. 14 and in the worst case scenario, they must go through the whole tree to prove a fault is *un-testable*. Such algorithms capable of searching an entire binary tree are called *complete*.



Algorithm 7 PODEM(C, f)

```
1: initialize all gates to don't-cares;
2: D-frontier = $\emptyset$;
3: result = PODEM-Recursion($C$);
4: if result == success then
5:     print out values at the primary inputs;
6: else
7:     print fault $f$ is untestable;
8: end if
```

Algorithm 8 PODEM-Recursion(C)

```
 1: if fault-effect is observed at a PO then
 2:     return (success);
 3: end if
 4: ($g, v$) = getObjective($C$);
 5: ($pi, u$) = backtrace($g, v$);
 6: logicSimulate_and_imply($pi, u$);
 7: result = PODEM-Recursion($C$);
 8: if result == success then
 9:     return(success);
10: end if
11: /* backtrack */
12: logicSimulate_and_imply($pi, \bar{u}$);
13: result = PODEM-Recursion($C$);
14: if result == success then
15:     return(success);
16: end if
17: /* bad decision made at an earlier step, reset $pi$ */
18: logicSimulate_and_imply($pi, x$);
19: return(failure);
```

Fig 15 Podem-algorithm [9]

A different approach for an ATPG algorithm is the one which that makes decisions only at primary inputs not at the internal nodes of the circuit. The PODEM algorithm described in [38] was designed and constructed based on this notion and makes decisions only at the primary inputs.

This algorithm checks at each step of the ATPG search process if the fault which is looking for is excited and if it is it goes on checking if there is an X-path from minimum one fault-effect in the D-frontier to a primary output, where an X-path is a path of "don't care" values from the fault-effect to a primary output [6]. If no X-path exists, it means that all the fault-effects in the D-frontier are blocked. In the opposite case, PODEM will choose the best X-path to propagate the fault-effect. In case the target fault wasn't yet excited, the first steps of PODEM will be to excite the fault. The normal flow of PODEM as well as its recursive variant are shown in Fig. 15.

For the recursive-PODEM, the search starts by picking an objective, and it backtracks from the objective to a primary input via the best path. Controllability measures can be applied in this case to choose the best path. At each step, new primary inputs will be assigned logic values. If the target fault becomes unexcited or the D-frontier gets emptied it means that somewhere a bad decision was taken, and it needs to be undone.

The backtracking mechanism assures that the most recent decision is reversed, if needed. If this reversal causes a conflict, the recursive algorithm will proceed in backtracking to earlier decisions, until the point where no more reversals are possible. At that time the fault is determined to be undetectable [6].

3.2 Fault Models

Because of the multiple types of VLSI errors, it is difficult to generate tests for real defects. Fault models are necessary for generating and evaluating a set of test vectors [6]. Many fault models have been proposed [7], but, unfortunately, there is no single fault model which can act like a template for all possible defects that might occur. In consequence, a combination of different fault models is often used in the generation and evaluation of test patterns.

Fault modeling can be viewed as generalizing the real physical conditions of defects across the abstraction levels for a system [48], [49]. Various fault models were proposed, which can be structured after the typical system development abstractions [50]:

- Faults at algorithm level
- Register transfer level
- Gate level
- Transistor level
- Layout level

Fault models must be handled at the appropriate level, as one cannot check for algorithmic errors at the tranistors level, for instance. So the fault simulation together with test vector generation must be done from the highest abstraction level, followed by an evaluation of the coverage in the lower levels and then deal only with the faults at those lower abstraction level which were not yet discovered, (top-down approach). Fault models at various abstraction levels are detailed some more in the following section. In order to evaluate the various types of faults and I used a simple circuit took from the hardware implementation I created for IDEA NXT, as it is detailed in the following sub-chapters. The sample circuit was used to both test the hardware implementation and to help decide which is better to use further in the testing experiments I needed to perform in this thesis.

3.2.1 Stuck-At Faults

The simplest and most-used fault model at gate level, one which was also used in the experiments conducted for this research, is the **stuck-at** fault model. This fault affects the state of logic signals on the inputs, outputs, sources, internal gate outputs and inputs of a circuit, making the correct value on an affected signal line to appear stuck at a constant - 0 or 1 logic – value denoted as **stuck-at-0** (SA0) and **stuck-at-1** (SA1). The example circuit took from the hardware IDEA NXT

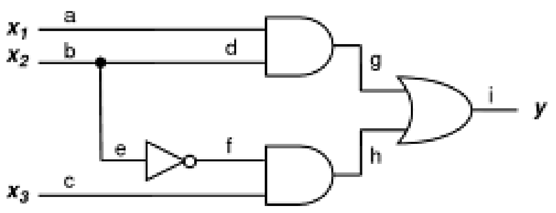


Fig 16 Stuck-at faults example

Table 1 Truth table for Fault-Tree and Faulty–circuits in Fig. 16

$x_1x_2x_3$	000	001	010	011	100	101	110	111
y	0	1	0	0	0	1	1	1
a SA0	0	1	0	0	0	1	0	0
a SA1	0	1	1	1	0	1	1	1
b SA0	0	1	0	1	0	1	0	1
b SA1	0	0	0	0	1	1	1	1
c SA0	0	0	0	0	0	0	1	1
c SA1	1	1	0	0	1	1	1	1
d SA0	0	1	0	0	0	1	0	0
d SA1	0	1	0	0	1	1	1	1
e SA0	0	1	0	1	0	1	1	1
e SA1	0	0	0	0	0	0	1	1
f SA0	0	0	0	0	0	0	1	1
f SA1	0	1	0	1	0	1	1	1
g SA0	0	1	0	0	0	1	0	0
g SA1	1	1	1	1	1	1	1	1
h SA0	0	0	0	0	0	0	1	1
h SA1	1	1	1	1	1	1	1	1
i SA0	0	0	0	0	0	0	0	0
i SA1	1	1	1	1	1	1	1	1

implementation is given in Fig. 16. The red dots in the circuit represent points on the lines where faults reside. As can be seen from the figure, a faulty line is permanently set to 0 or to 1. Also, a fault can be at an input or at an output of the gate.

Its truth table for the faulty circuits for all possible single stuck-at faults is given in Table I. The grey entries in the truth table are the points where the circuit's output response was different from the fault-free one, meaning that the corresponding inputs are valid test vectors which can detect stuck-at faults. Test vectors 011 and 100 can detect 10 faults and 001 and 110 can detect the rest 8 faults, obtaining a 100% stuck-at fault coverage.

This fault model also works for sequential circuits, but high coverage is more difficult to obtain than for combinational circuits, because in a sequential logic circuit it is necessary to generate sequences of test vectors, so DFT techniques are frequently used to ease circuit test generation, as stated in [9].

The defects which are not covered by stuck-at fault models can also be detected with the so-called *N-detect single stuck-at* fault test vectors, a notion defined in [10]. In the *N*-detect set of test vectors, each single stuck-at defect is detected by minimum *N* distinct test vectors. They don't also cover all possible defects so other types of fault models need to be used as well.

3.2.2 Transistor Faults

Like the name says, there are applied to transistors in the CMOS logic circuits, which can be **stuck-open** (**stuck-on**) or **stuck-short** (**stuck-off**) at the switch level. For explaining this concept I will reffer to the CMOS NOR gate with 2 inputs in Fig. 17, took from the Data Register in the IDEA NXT hardware model. If N2 would be a stuck-short fault, it would produce a conducting path between V_{DD} and V_{ss} for the test vector 00 [6].

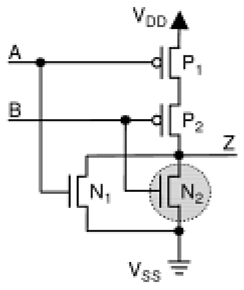


Fig 17 Two-input CMOS NOR GATE

This produces a voltage divider at the output Z which can be interpreted as wrong logic level by the gate inputs. But the short-on faults can be detected by monitoring the power supply current during steady state, I_{DD0}.

3.2.3 Open and Short Faults

Stuck-open and stuck-shorts faults cane also occur in the wires interconnecting the transistors of a logic circuit to form gates. A set of test vectors which can achieve an increased coverage of stuck-at and faults, will also be able to detect some of the wire Open Faults. There are also fault which are not as easy to detect, like a resistive open, which affects the signal path delay propagation.

A short between two transistors or connections between them is called a **bridging fault**. One bridging fault model is known as the **wired-AND/wired-OR** as the logic values were models as logical AND / OR on the shorted wires. In case of a wired-AND, the two shorted lines will be at 0 logic if one of the lines has 0 logic value, and the wired-OR means that the resulted signal will have the 1 logical value if one if the lines is 1.

In conclusion, the bridging fault can be modeled with an extra AND / OR gate, as can be seen in Fig. 18, which was adapted from [11].

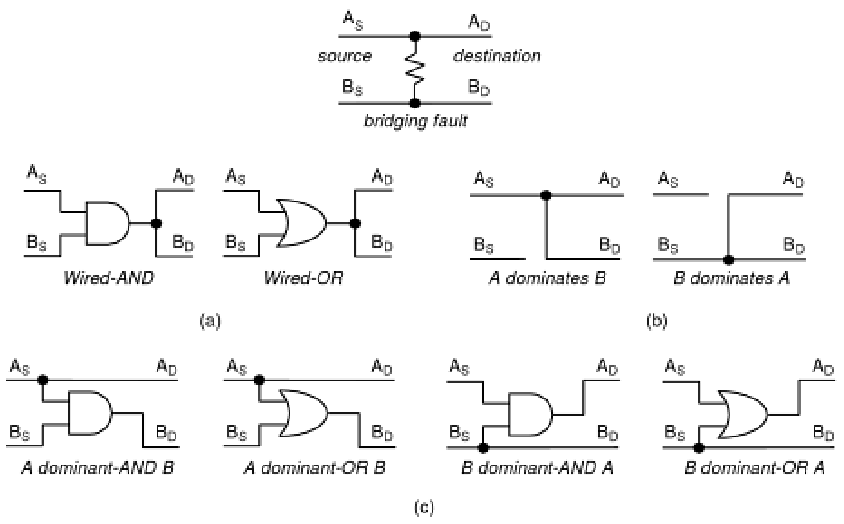


Fig 18 Bridging fault models

Table 2 Truth table for Fault-free and Faulty circuits in Fig. 18

A_SB_S	0	0	0	1	1	0	1	1
A_DB_D	0	0	0	1	1	0	1	1
Wired-AND	0	0	0	0	0	0	1	1
Wired-OR	0	0	1	1	1	1	1	1
A dominates B	0	0	0	0	1	1	1	1
B dominates A	0	0	1	1	0	0	1	1
A dominant-AND B	0	0	0	0	1	0	1	1
B dominant-AND A	0	0	0	1	0	0	1	1
A dominant-OR B	0	0	0	1	1	1	1	1
B dominant-OR A	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

This fault model however was initially created for the bipolar VLSI, so a new type of bridging needed to be designed for the CMOS VLSI. That type was called **dominant bridging fault,** in which one driver is supposed to dominate the logic value two shorts [13]. The dominant bridging fault is not as easy to detect as is predecessor because the erroneous behavior can be seen only on the dominant shortened net, instead of both nets (as can be for the wired-OR/wired-AND). If we analyze Table II, we can affirm that a set of test vectors which are able to detect bridging faults, can also detect all wired-AND and wired-OR bridging faults.

Another, derived fault model, tailored for particular cases of a resistive short was also designed and it is called **dominant-AND/dominant-OR** bridging fault, in which one driver dominates the logic values of the shortened nets, but only for some logic values. A set of test vectors which are able to detect all dominant-AND/dominant-OR bridging faults will also be able to detect all wired-AND / wired-OR bridging faults on that circuit [14].

3.2.4 Delay Faults and Crosstalk

A circuit is said to be error-free when not only a fault did not occur, but also if the correct logic signals are being propagated along the lines. A delay fault causes excessive delay along a path such as the total propagation delay does beyond the specified limit [6]. The 3 main delay fault models are:

- Gate-delay fault
- Transition fault – both appear when a time interval taken from a transition from the gate input to output goes beyond a certain range
- Path-delay fault – which is the sum of all gate delay along a signal path. The problem which can appear here is that there can be a really large number of paths.

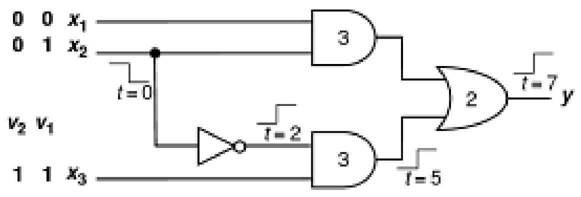


Fig 19 Path delay fault example

The NAND-OR circuit took from the IDEA NXT Data Register illustrated in Fig. 19 can be used to explain the concepts introduced above. The integer values on each gate represent the fault-free delays and *v1, v2* are test vectors used to test the path delay from input *x2* to through the NOR and AND gates until output *y*. If we consider the transition between the test vectors to happen at *t=0*, then with all the delays on the gates the transition should be propagated at the output at *t=7*. If a fault would have occurred sometime along this process, the transition time at the output would have been > 7.

However, with the size of components decreasing day by day, even reaching nanotechnologies, the cross-coupling capacitance and inductance between interconnections increases, creating crosstalk side-effects which may ultimately cause the chip to function erroneously. There are 2 main types of side-effects [15]: **crosstalk glitches and crosstalk delays**. Glitches are in fact pulses caused by coupling effect in interconnected lines, when for example a transition is applied to a line with strong driver and other lines have weaker signals. Crosstalk delay is cause by the same effect but can appear even if the line signals have close values, but have big loads. If a circuit with gate and interconnection delays also has crosstalk delays, the total circuit's delay increases considerably. Traditional delay fault models are not enough for these delay fault types, so various techniques, like physical design and analysis tools need to be used in conjunctions with conventional ones.

Because of the multiple types of VLSI errors, it is difficult to generate tests for real defects. Fault models are necessary for generating and evaluating a set of test vectors [6]. Many fault models have been proposed in [7], but, unfortunately, there is no single fault model which can act like a template for all possible defects that might occur. In consequence, a combination of different fault models is often used in the generation and evaluation of test patterns.

3.3 Fault-Tolerance in Computer Systems

Once a fault is present in a system, it must either be dealth with, by checking what caused it and fixing or replacing the faulty components, or ignored and the system is let to operate in its presence, with the non-faulty components doing the job. Fault-tolerant computing is defined in [43] as the art and science of building computing systems that continue to operate satisfactorily in the presence of faults or as the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases, the decrease is proportional to the severity of the failure, as compared to a npn-tolerant designed system in which even a small failure can cause total breakdown. Fault tolerance is particularly desired in high-availability or life-critical systems.

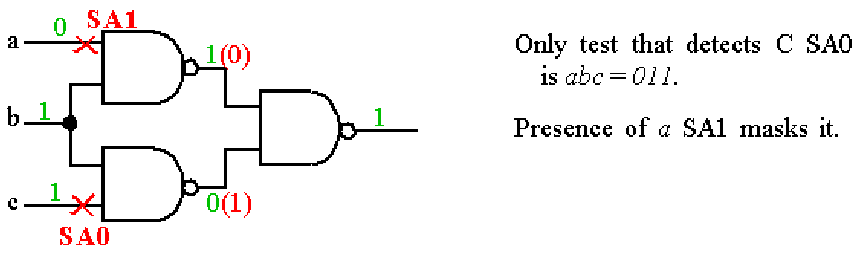

a
0
SA1
1(0)
b
1
1
c
1
SA0
0(1)
Only test that detects C SA0 is $abc = 011$.
Presence of a SA1 masks it.

Fig 20 Fault masking example [113]

A fault-tolerant design helps a running system to operate as expected, or potentially at a lower level, instead of failing completely, when some components of the system become erroneous and fail. This concept usually describes computer systems which are implemented in such a manner as to continue to work as functional as possible, maybe slower or with a reduce rate or longer response times in the presence of faults or failed parts. So, even if components or whole parts of the system become non-operational, the system itself will not fail completely, regardless if the problems appear in hardware or are caused by something in the software. A parallel can be done in the automobilistic field another when a motor vehicle can be designed so it would continue run even if one of its tires is punctured [58].

A lot of work has been done in the field of fault tolerance for the last 20-30 years, which concluded with building of a number of fault-tolerant machines. Fault-tolerant computing is a key factor in transportation, communication, e-commerce, aircrafts, internet transactions and more. As novel industries and technologies emerged, such as reconfigurable computing, mobile computing, parallel scalable computing, updated fault-tolerance schemes need to be developed in order to be up-to-date.

A fault-tolerant system must support one or multiple fault-types such as:

- transient, intermittent or permanent hardware faults
- software and hardware design errors
- operator or function software errors

physical damage.

Fault-tolerance is usually achieved by applying predefined sets of design and analysis methodologies to build systems with increased dependability and availability.

Most fault-tolerant designs are designed for recovery from random faults which appear in hardware components. The general approach is to partition the system into modules which behave like fault-containment regions. For every module a protective redundancy component is added so that in case the module becomes faulty or stops operating, others can take its role. Special mechanisms are added to detect errors and implement recovery. There are two main techniques used for **hardware fault recovery**: fault masking and dynamic recovery [115].

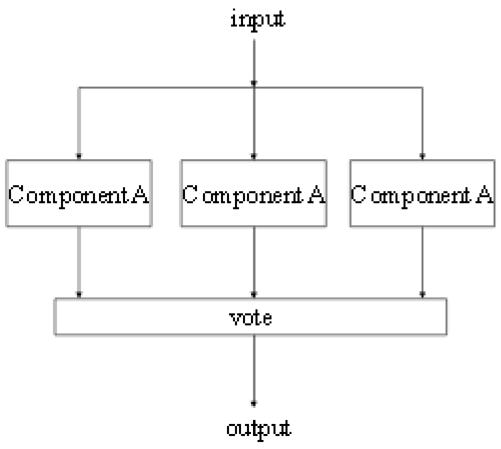


Fig 21 Triple Modular Redundancy [51]

Fault masking is a structural redundancy technique that completely masks faults within a set of redundant modules [43]. A couple of modules which have the same structure execute the same set of functions whose results are used in removing errors in a faulty module by submitting the output to a voting process. The most well-known fault-masking technique is Triple modular redundancy (TMR) in which the voting circuit is triplicated. A TMR system is considered to have failed when errors emerge from two modules belonging to a redundant as in this case the vote becomes invalid.

Dynamic recovery is used in the corner case when only one form of computation is running at a time and it might require automated self-repair. This technique also partitions the system into modules each having it own redundancy spare to protect it from failures [57]. Besides this other specific mechanisms are used for module fault detection, switching out an erroneus module, switch in a spare, and performing the necessary restoration software functions like rollback, initialization, restart or retry. This is done either by special hardware or separate processors. An example is given in Fig. 22.

In the software area the same principles of static and dynamic redundancy are applied and their specific adaptation is similar to the one used for hardware faults. One such approach, **N-version programming**, uses a form of static redundancy in distinct programs (versions) perform the same functions and their outputs participate in the voting process.

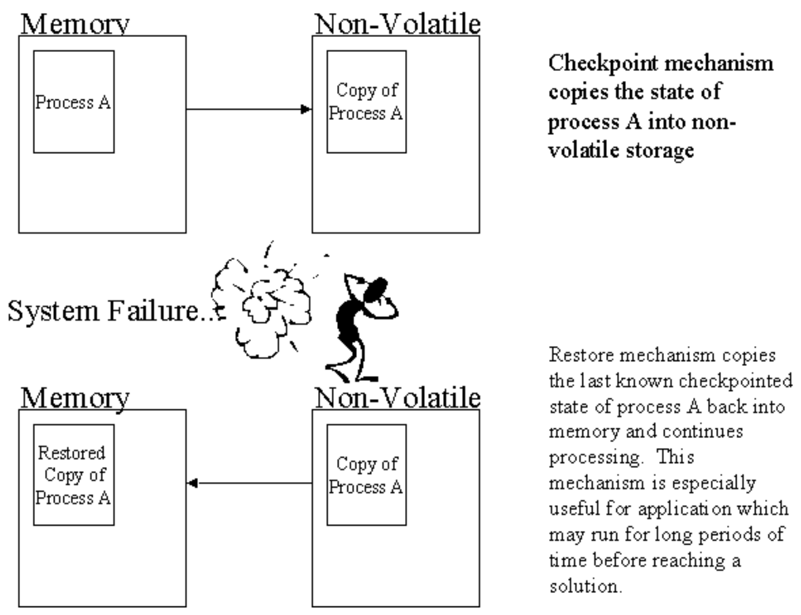

Memory
Process A
Non-Volatile
Copy of Process A
Checkpoint mechanism copies the state of process A into non-volatile storage
System Failure...
Memory
Restored Copy of Process A
Non-Volatile
Copy of Process A
Restore mechanism copies the last known checkpointed state of process A back into memory and continues processing. This mechanism is especially useful for application which may run for long periods of time before reaching a solution.

Fig 22 Dynamic recovery example [54]

However, the data being voted could have discrepancies so the various versions are analyzed by an initially-agreed and if they prove faulty, they are rejected so that in the end, only the good versions remain. Another dynamic technique makes use of **recovery blocks** in that software programs are partitioned into various code blocks and a series of acceptance tests are executed after every block; in case an acceptance test fails, a spare "recovery" block gets executed.

Another method for verifying the dependability of a SUT (system under test) is **Fault injection**, which implies inserting faults into the system and verifying its behavior when a fault occurs. The definition given in [55] is "the validation technique of the dependability of fault tolerant systems which consists in the accomplishment of controlled experiments where the observation of the system's behavior in presence of faults is induced explicitly by the writing introduction (injection) of faults in the system".

Fault injection techniques can be grouped into five main categories [56]:

Hardware-based fault injection: This is performed at physical level by injecting voltage turbulences (power supply disturbances), laser fault injection, changing the circuit's pin values or inserting environment parameters (heavy ion radiation, electromagnetic interferences)

Software-based fault injection: This technique aims to reproduce at software level the faults that could be produced in hardware.

Simulation-based fault injection: This is performed by injecting faults in high level models (using distinct description but mostly VHDL models). This way, the system is evaluated at early stages, when just one of the system's models is available and at various abstraction levels. A good and robust system must be designed so that the different abstraction layers are interoperable and testing / validation processes are seemingly and effortlessly integrated.

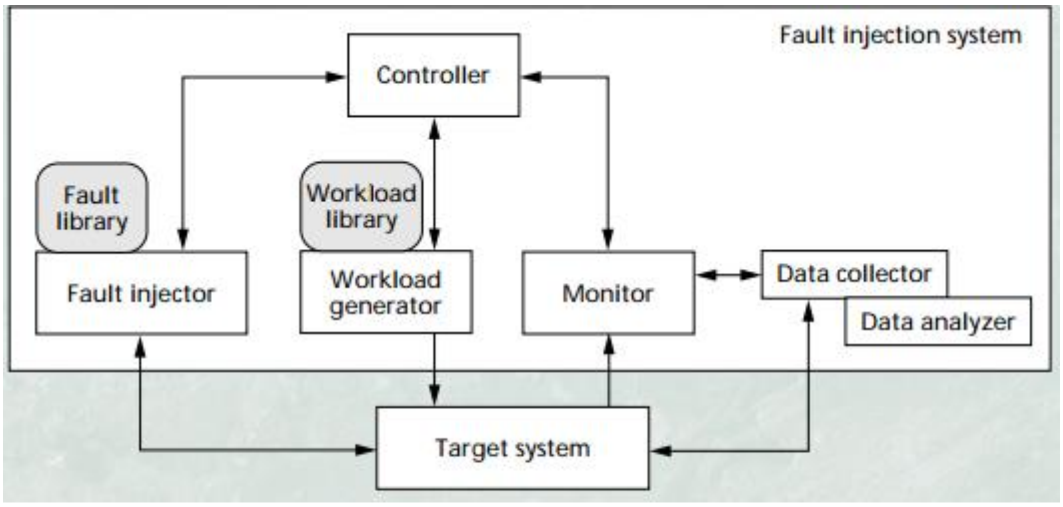


Fig 23 Components for a Fault Injection Environment [114]

Emulation-based fault injection: Represents a viable alternative for the simulation-based fault injection, as it reduces the time spent on the simulation. For circuit emulation and speeding-up the simulation process it makes use of Field Programmable Gate Arrays (FPGAs). A circuit is studied in its real environment and then emulated on a board, keeping in mind real-time interactions.

Hybrid fault injection: This fault injection method is a possible way to assess the consequences of hidden bugs, which makes use of the advantages from both the software and hardware techniques, to discover problems from both areas.

An example of a basic Faults Injection environment is showed in Fig. 23.

The Fault injection's role is, in essence, to check if the response of a system is identical to what was specified, when various errors and faults are present in the system. Fault injection techniques have two main functions:

- removing faults (by correcting potential fault tolerance deficiencies in the system); here the aim is to have the highest possible fault coverage, in both quantity and fault types
- forecasting faults (by evaluating the coverage distribution of the system, namely the factor and latency)

3.3.1 Hardware-based Fault Injection

Hardware-based Fault Injection can be made either with contact, and is called pin-level injection (probes, socket-injection) or without contact, like heavy ion radiation and electromagnetic interference.

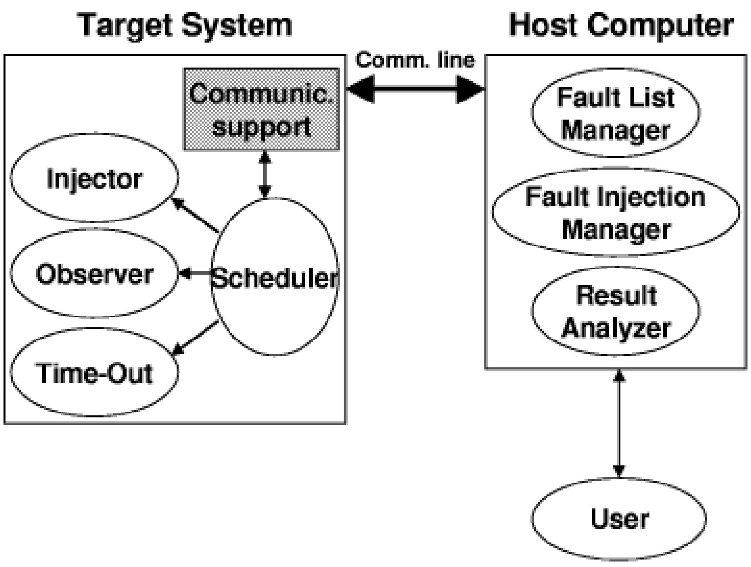


Fig 24 Hardware Fault-injection

Some advantages of this technique are presented in [43]:

- Hardware fault injection is able to access memory locations which otherwise could not be reached. For example, the Heavy-ion radiation method can inject fault into VLSI circuits at locations which are impossible to reach by other methods.
- It works best in systems which need high time-resolution for hardware monitoring and triggering.
- Injecting directly faults into hardware circuitry is, in many situations, the only viable options for estimating with accuracy coverage and latency.
- It mainly injects faults with low perturbation.
- This technique is most appropriate for low-level fault models.

- Not intrusive: No modification of the target system is required to inject faults.
- Experiments are fast and can be run in near real-time, allowing for the possibility of running a large number of fault injection experiments.
- The main disadvantages are their high costs and the fact that they are harder to control [121].

The main tools used for achieving hardware fault tolerance are: RIFLE, FOCUS, MESSALINE, MARS. A general example of a hardware fault tolerant system is shown in Fig. 24.

3.3.2 Software-Based Fault Injection

If hardware malfunctions are the most disastrous, software faults are the most frequent. Fault injection techniques are possible solution in finding common or hard to find bugs. It mainly consists in modifying the software running on the system in such a way that the system's state can be changed as well to fit the programmer's understanding of the system. There is a wide range of faults that can be injected, from erroneous error conditions and flags, register and memory faults to dropped or replicated network packets. These faults can also be injected into simulations of complicated systems, where the implementation details are hard to understand, but the interactions and communications are easier to be followed, or directly into running systems, to study the effects.

Faults can be corrupted memory or registers, missing messages, responses, erroneous disk reads, bad timings, etc. A simulation means running the system with a certain fault and examining its behavior in the presence of that fault. These types of simulations are time consuming because they comprise all operations and details of the system. After a simulation has taken place and results were analyzed, the faults which the system has successfully taken care of are cataloged and the performance of the design in measured depending on the number and magnitude of these faults.

These simulations can either be non-intrusive, or intrusive in case time is a key factor, in which case fault injection mechanisms will interrupt the functioning of the system, and cause produce outputs that are not very reliable as they would not happen if the injection scheme was not present. This happens because the injection mechanism functions on the same system as the software under test.

Advantages of software-based fault injection mechanism:

- This technique can be applied to both applications and operating systems, something that is difficult to achieve with hardware fault injection.
- Fault-injecting experiments can be performed in near real-time, which means that a large number of experiments can be run in a relatively small amount of time.
- The best results are obtained when running the fault injection experiments on the real hardware that is executing the real software so that any design error that may be hidden in the hardware or software is rapidly found.
- Does not require any special-purpose hardware; it has a low complexity, low development and low implementation cost.
- No model development or validation required.
- Can be adapted for novel classes of faults.

Of course, this mechanism also has a series of limitations:

- Reduced set of injection instants (only at assembly instruction level).
- It’s not able to inject faults in locations un-accessible to the software.
- The source code must be changed in order to accommodate the fault injection mechanism, which means that the code that is running while the fault scheme is taken place is not the exact which will be implemented in the field.

Fault-tolerant design involves more than just reliable hardware and software. Many issues can be avoided if planning is made in advanced, having in mind all the “what if..?” scenarios.

More details about fault injection techniques and how they were applied in this research will be given in the next chapter.

Chapter 4 TESTING ARCHITECTURES FOR THE IDEA NXT ENCRYPTION ALGORITHM

The previous chapter introduced various types of errors which can occur in VLSI circuits and the fault models which are necessary for generating and evaluating a set of test vectors for the circuit under test, to asses its correct functioning or to discover errors at early stages.

Many fault models have been proposed [7], but, unfortunately, there is no single fault model which can act like a template for all possible defects that might occur. In consequence, a combination of different fault models is often used in the generation and evaluation of test patterns.

Searching for defects at gate and transistor levels made me realize this is just the tip of the iceberg. In a complex algorithm such as IDEA NXT, defects can propagate at all levels, and the approach must cover all levels, starting with the most extensive one – the algorithm abstraction level itself.

A hardware implementation of the algorithm offers increased speed of execution and superior key strength, but its integration into a circuit or a chip must also assure integrity, confidentiality and non-repudiation alongside the sine-qua-non request of security [25]. Brian Gladman affirmed that „it is relatively easy to build an encryption system that is secure if it is working as intended and is used correctly but it is still very hard to build a system that does not compromise its security in situations in which it is either misused or one or more of its sub-components fails (or is 'encouraged' to misbehave)" [123].

The costs of repairing a faulty cryptographic system is quite high and would also represent a problem for its developer. In addition to the probability of faults affecting a crypto-chip, they are prone to malicious attacks. Crypto-systems usually protect important and sensitive information, such as keys stored in Automatic Teller Machines or Payment services. As has already been proved in various papers - [4], [17], [51], [52] - there are many different types of attacks that can compromise the encryption process even in the case of a hardware implementation of a cryptographic algorithm, as stated in [52], [53] and [54]. Attackers can inject faults into crypto-chips and cryptographic cores, which can lead to permanent faults by modifying the underlying semiconductor layer [55], [25]. I can mention particularly linear and differential cryptanalysis and fault attacks [56], [57], [58]. A successful attack on such a system translates into serious financial implications for the user. By injecting faults into a cryptographic unit, enough information can be obtained to reduce greatly the efforts to reaching the secured data, as shown in [60]. Novel techniques such as side-channels, timing of module's operation or fault injection represent serious security threats for system's security.

Side-channel attacks [61] require just the observation of power, thermal and electromagnetic radiation, whereas faults can be injected into designs by multiple mechanisms, as described in [18]:

- Physical injection of an error through a focused laser beam
- Inject a glitch by rapidly varying the clock frequency
- Insert a spike into the module's power supply by carrying the supply's voltage
- Overheat or freeze the unit so that the existing defect will manifest by itself

Injected faults are *transient* faults, meaning that they leave no trace regarding the failure actions provoked to the algorithm; at the next round of encryption of the algorithm, the fault will not manifest itself, by its effects already reside in the modified state.

Earlier this year, at the request of a financial institution, Kaspersky Lab's Global Research and Analysis Team performed a forensics investigation into a cyber-criminal attack targeting multiple ATMs in Eastern Europe. During the course of this investigation, it was discovered a piece of malware that allowed attackers to empty the ATM cash cassettes via direct manipulation [118].

At the time of the investigation, the malware was active on more than 50 ATMs at banking institutions in Eastern Europe. Based on submissions to VirusTotal, it was believed that the malware has spread to several other countries, including the U.S., India and China.

Besides these malicious attcks, crypto-algorithms are also prone to intrinsec errors, to defects which can occur at design or implementation level, which can periclitate the well-functioning of the algorithm once it is used in the field. It is also important to be able to differentiate between a malicious attack and ones provoked by functional problems, because these two distinct types inccur different means of dealing with them.

My main interest in the current research was to assure the correct functioning of the IDEA NXT crypto-algorithm in its useful time, and this can be achieved by detecting a faulty behavior as soon as it occurs. I conclusioned that in order to take the necessary measures to backfire a faulty behavior of the algorithm and assure the correct functioning of the system in which it is integrated as soon as possible, every encryption module should include mechanisms for error detection. Due to all this I came up with the idea of imagining a couple of error-detection architectures specifically designed for the IDEA NXT crypto-algorithm, ahead of time, forseeing its future integration in cripto-chips or other devices, for the reasons I exposed in the previous chapters. Once IDEA NXT will have an extended usage, it will become imperative to assure its correct functioning at any moment of time, by detecting at early stages any errors which could have been manifesting in the algorithm.

The protection of encryption modules can be performed either using external test modules, such as the *Automatic Test Equipment* (ATE) or internally, on-chip, with the help of build-in techniques. The first solution achieves higher fault coverage, but it is more costly, so generally on-chip, Design For Testability (DFT) solutions like *Built-In Self Test* and *Scan Chain* are preferred.

As mentioned in the previous chapter, there are two main types of testing strategies: online or concurrent, which detects errors while the system is running, and offline testing, which doesn't need the crypto-chip to operate when it does verifications. In my research I investigated both online approaches, based on computing the output parity of the algorithm, as well as different kinds of offline error-detection strategies, because I intended to cover an area as large as possible, for as many types of defects as I could. As stated in [45] the best protection is achieved using both online and off-line techniques, as the combination of the two covers a greater area of defects.

The test strategies I investigated will be presented separately in the following sub-chapters, describing both the theoretical aspects as well as the original variants constructed for IDEA NXT.

This following paragraphs will introduce a series of non-concurrent test architectures I designed for the novel family of crypto-algorithms, IDEA NXT. The proposed error-detection schemes are capable of verifying the integrity of a crypto-chip in an autonomous, non-concurrent manner. One of the testing solutions consists of stimulating the algorithm with test vectors and verifying the correctness of the output after a number of encryption or decryption iterations were run and the other one evaluates the response by compressing the results obtained after a predefined number of encryptions into a signature. The proposed test schemes offer a good trade-off between the length of the test process and the storage requirements for the correct responses. These originals off-line testing architectures were first introduced in a paper [57] which I presented at the ICSTCC IEEE conference, for which I was nomited for Best Paper Award of the Young scientists.

Foremost, the thesis also introduces a hardware architecture for online self-test in the context of the IDEA NXT crypto-algorithm. From the many techniques and solutions presented in the literature for increasing Built In Self-Test (BIST) capabilities, after a careful analysis of these approaches, I decided to focus my attention on solutions based on parity-based error detection. In this sense I designed and implemented a complete parity-based test architecture for IDEA NXT.

The solution I propose doesn't interfere in any way with the algorithm's structure, as there is a complete separation between the functional and testing channels. The proposed solution is the first of this kind for the IDEA NXT crypto-algorithm. I evaluated the performance of the proposed test strategy with different redundancy levels and, formulated recommendations for the concurrent detection strategy based on the obtained experimental results.

4.1 Offline Error-Detection Architectures for the IDEA NXT crypto-algorithm

The off-line test technique assumes that the entire digital system or just a part of it is "taken out of service" [10] so that the test process can take place. Because the hardware overhead is quite small (the system doesn't have to be kept in its normal operating mode), this technique allows for an extremely high coverage of the device's area.

Off-line test can be used in conjunction with online test for fault localization [24]. If a fault was detected with online testing, the off-line testing can be used to detect the exact location of the defect or the components to which it spread. Because of the way in which they are applied, off-line techniques are generally used for detecting faults at a larger set of locations. For this technique to function, the system's state needs to be controllable, so off-line techniques are used in idle cycles or after a system repair.

Off-line schemes can be implemented using either Built-In Self Test (BIST), its customized variant Built-In Logic Block Observer (BILBO), Scan Design or Multiple-Input Scan Register mechanisms. I constructed different architectures using each of these possibilities, in order to compare them and find the most efficient and appropriate solution. Each of them will be presented in detail in the following sub-chapters.

Before designing the fault-tolerant architecture, I started studying a couple of strategies for minimizing the aliasing probability, which can appear in signature compaction strategies. The solution I provide is to increase the number of output vectors that will be checked for errors.

4.1.1 IDEA NXT Built-In Self-Test Architecture

The starting point of the investigations for constructing an offline architecture was the typical built-in-self-test scheme in Fig. 26. This structure seemed fit for IDEA NXT because in its hardware – block construction, just a single register is used for the Datapath, as well as one register for the Key Scheduler, as in the typical scheme in Fig. 26.

Logic built-in self-test (BIST) is defined in [33] as a design for testability technique in which a portion of a circuit on a chip, board, or system is used to test the digital logic circuit itself.

Logic BIST has become the main testing technique for many applications in the aerospace / defense, automotive, banking, computer, healthcare, networking, and telecommunications industries where there is a need for a self-testing of on-chip, on-board systems to improve the dependability of the whole system, as well as the ability to perform remote diagnosis [33]. There are two main types of BIST: *online* – performed when the circuit is operating, and can be done *concurrently* or *non-concurrently* [34] - and *offline* – performed when the circuit is not running, so it doesn't detect any real-time errors.

Offline BIST can also be of two types itself: functional and structural. *Functional* tests are based on the functional specifications of the CUT and usually target functional or high-level fault models. *Structural* offline BIST tests are designed after the structure of the functional circuit. Structural offline BIST techniques can be divided into two categories [23]:

- *external BIST*, in which test pattern generation and output response analysis is achieved by a separate architecture than the one under test
- *internal BIST*, in which the functional storage elements are converted into test pattern generators and output response analyzers.

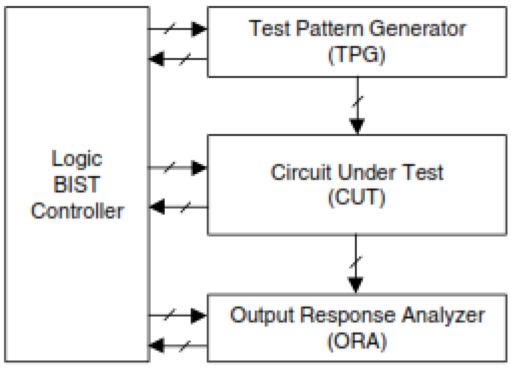


Fig 25 A typical logic BIST system [6]

Structural offline BIST has a couple of advantages compared to conventional scan [35]:

- BIST can run tests on the board or system at any time without needing a human tester.

- Because BIST implements the majority of testing directly on-chip, the error root can be easily found on the chip; most faults are detected on the fly, with no need to model them in software
- At-speed testing, another feature of BIST, can be used to detect many delay faults.
- Test costs are significantly lower because of the reduced test time, tester memory requirements and investment costs.

As seen in Fig. 25, the BIST system is composed of a Controller, which decides which operation is to be done when and by whom, a Test Pattern Generator (TPG) for feeding test patterns to the Circuit Under Test (CUT) and an Output Response Analyzer (ORA) which eventually analyses the outputs of the CUT and decides if an error was introduced in the system or not.

The next paragraphs will present the basic concepts and design rules of logic BIST, Test Pattern Generation and Output Response Analysis techniques including ones count testing, transition count testing, and signature analysis, as they are used in the testing architectures I developed which will be presented later on in the thesis.

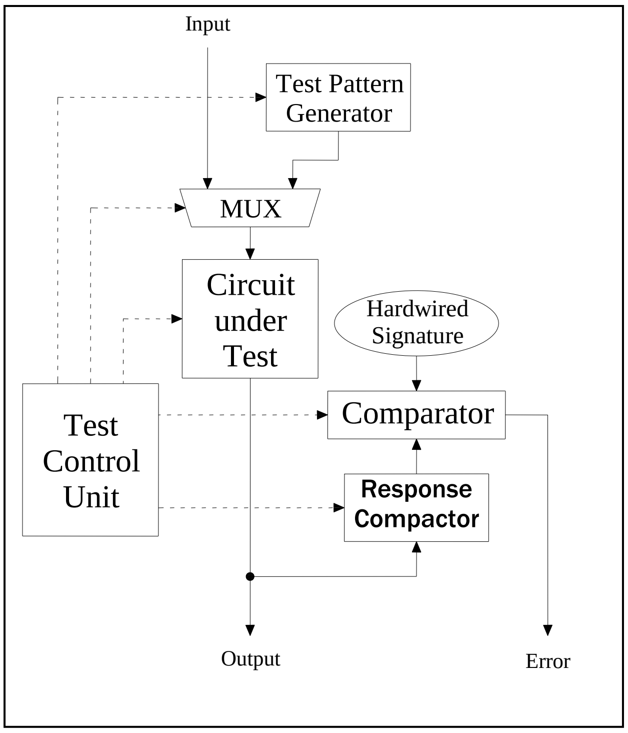


Fig 26 Built-In Self-Test architecture [59]

Our proposed offline BIST error-detection architecture assumes to use a Multiple In Fig. 26 a typical structured offline logic BIST system is illustrated. The **test pattern generator** (TPG) automatically generates test patterns to be driven to the inputs of the circuit under test (CUT). The **output response analyzer** (ORA) then compacts the output responses of the CUT into a **signature**. Specific BIST timing control signals, like clocks or scan enable signals, are generated by the logic BIST controller for coordinating the BIST functioning along the CUT, TPG and ORA. Once the BIST functioning completes, a pass/fail signal shows the final result of testing. It also compares the resulted signature with an embedded golden signature, and usually makes use of diagnostic logic for faults diagnosis. The storage elements in the TPG, CUT, and ORA must be initialized to known states before the self-testing took place so that only known values are passed on from the CUT to the ORA, as unknown values can corrupt the compressed signature and cause the malfunctioning of the BIST design. It can be concluded that the tested circuit must be designed with additional BIST-specific rules.

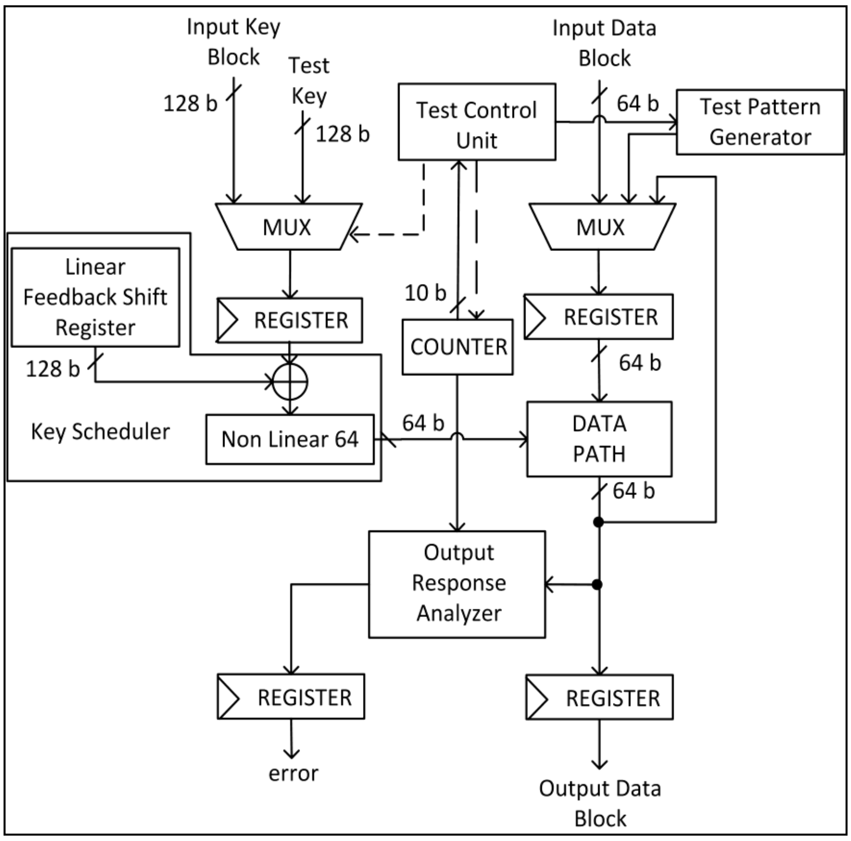


Fig 27 IDEA NXT BIST Architecture based on signature compression

Input Signature Register (MISR) compaction mechanism for evaluating the output of the encryption by comparing it to a so called “gold signature”, which is a hardwired value obtained by simulations of the IDEA NXT algorithm, in conjunction with a counter for keeping track of the number of encryption rounds that were executed.

The output resulted after a run of the encryption algorithm round is compacted, encompassing all the previous encryption outputs.

As can be seen in Fig. 27, the Output Response Evaluator takes as inputs the Datapath's output as well as the value of a Counter holding the number of encryption rounds that are to be performed. The number of rounds to be executed is determined by means of simulations. When the counter reaches this value, the ORA unit cheks if the result is the expected, correct one, and if so gives the error signal a value of '0', otherwise it considers there is a fault in the system and sets the error signal.

Test compression is a technique that compresses the deterministic test data set (stimulus and response) stored on automatic test equipment (ATE) with the help of new hardware added on the chip, before the scan chains. The *automatic test equipment (ATE)* is a machine that is designed to perform tests on different devices referred to as devices under test (DUT). An ATE uses control systems and automated information technology to rapidly perform tests that measure and evaluate a DUT. It has limited memory and input/output channels and a small bandwidth which causes the test speed to be low as well, hence the need to speedup testing. An architecture for general test compression is illustrated in Fig. 28.

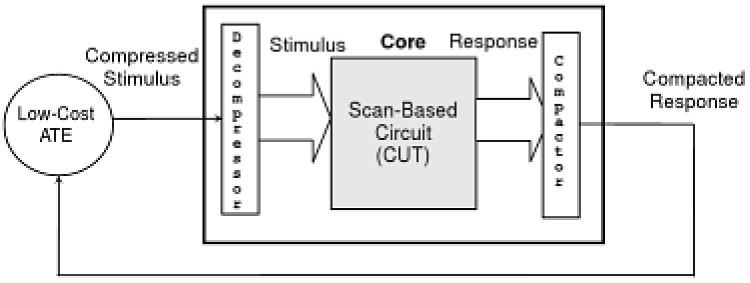


Fig 28 Architecture for test compression [6]

The test set can be generated for instance with an ATPG. As opposed to BIST, the test vectors generated with ATPG are applied completely non-changed to the circuit. This technique reduces the data size stored on the test equipment 10 or even 100 times, which leads to reduces memory and reduced test time. Bits which haven't got a value assigned to them are usually "randomly filled" with 1/0 by the ATPG, but in the case of stimulus compression, those values are left as "don't' care". This makes the test vectors easier to compress and the resulting test patterns will remain unchanged after decompression. Vectors in which not all bits are assigned values to are called *test cubes*.

Response Compaction is a technique used to minimize the test data volume replied back to the tester is test. This differs from test response compression in that it can be lossy, while the other must be lossless. This compaction can be made in time or space dimensions and can be linear or nonlinear. Example of time compaction techniques are X-compact [48], X-blocking, X-masking, as well as OPMISR [49] and convolutional compactor [50] as combined time-space techniques. Both Test Compaction and Test Compression are beginning to be successfully integrated in design flows, especially embedded hard cores, and they rival with BIST due to their good results and small costs. The Test Control Unit (TCU) in Fig. 28 has the role of supervising the test process with respect to both the Datapath and the Key Scheduler, also deciding when the evaluation of the obtained signature must take place. The encryption process takes as inputs a special test key on 128 bits.

In my implementation, because of the LFSRs contained within the Key Scheduler, I provide the key generation unit with the all-zero vector during self-test operation, allowing the 6 24-bit LFSRs to effectively stimulate the round key generation logic. As for the 64 bits of plaintext to be processed they are generated by a dedicated Test Pattern Generation unit.

I implemented three versions for the TPG: using a counter, a LFSR and a Cellular Automata, all on 8 bits each, in order to evaluate the area overhead and latency degradation incurred by each of the three approaches. As for the previous design, the outputs of the Datapath are grouped into a feedback interconnection which permits the output of the current iteration to serve as input to the next iteration and also as an encryption output stored in the output register.

4.1.1.1 Implementation of Test Pattern Generator

The most commonly used structure for generating pseudo-random number for test patterns in TPGs is the Linear Feedback Shift Register (LFSR). There are two main types of LFSR:

- *standard or external-XOR LFSR*, where all XOR gates are fed sequentially into one another and end up as the input to the least (or most, either is correct) significant bit of the LFSR.
- *modular* or *internal-XOR LFSR* where XOR gates feed into different registers within the LFSR, and are not sequential (the XORs are inside the shift reg.

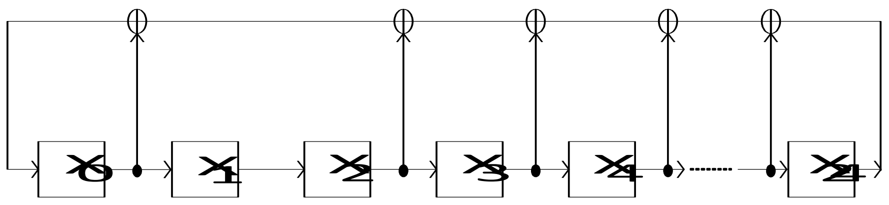


Fig 29 Standard LFSR for IDEA NXT

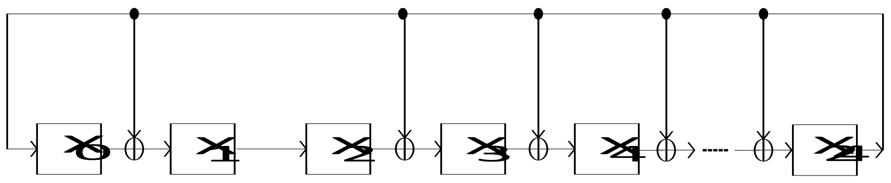


Fig 30 Modular LFSR for IDEA NXT

Fig. 29 shows a standard LFSR scheme, constructed for the LFSR used in the IDEA NXT algorithm, which has the *characteristic polynomial* of degree *24, f(x)=* $1+x+x^3+x^4+\ldots+x^{24}$. Fig. 30 shows a modular LFSR constructed for the same *characteristic polynomial.*

If I would consider a simple 3-bit LFSR, the only primitive polynomials for degree 3 are $1 + x^2 + x^3$ and $1 + x + x^3$ (they are reciprocals of each other, 1011 and 1101. So there are 2 possible representations for the primitive polynomials $1 + x^2 + x^3$ and $1 + x + x^3$ (they are reciprocals of each other, 1011 and 1101), but the second proved to generate better speed.

Since there are two primitive polynomials and two different implementation strategies, we therefore have four unique ways of implementing the LFSR. In fact, each of these implementations can differ according to which register is the most significant bit (either way will have 2^n-1 states, but with different sequences).

A modified, so-called complete LFSR can be used for exhaustive testing (applying 2^n exhaustive patterns to an n-input CUT). Exhaustive testing guarantees that all detectable, combinational faults (those that don't transform a combinational circuit into a sequential circuit) will be detected. This technique is applied only to circuits where the number of inputs is small, otherwise becomes ineffective as it will take too long to execute.

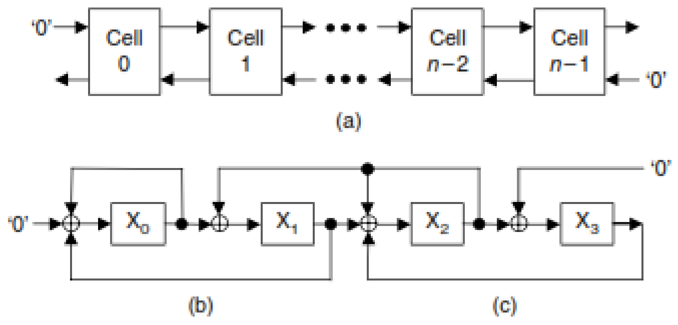


Fig 31 Cellular automata (a) general structure; (b) four-stage CA; (c) test sequence generated by (b) [38]

The techniques presented below are derived from a standard LFSR structure and are targeted at reducing the number of necessary test patterns:

- **Pseudo-random pattern generator (PRPG)** reduces test length but also the overall fault coverage. This technique was integrated in the second error-redection test scheme, which will be presented in the next sub-chapter.

- **Weighted LFSR**, first introduced in [37] modifies the maximum-length LFSR to produce an equally weighted distribution of 0's and 1's at the input of the CUT. It skews the LFSR probability distribution of 0.5 to either 0.25 or 0.75 to increase the chance of detecting faults that are difficult to detect using just a 0.5 distribution.
- **Cellular automata (CA)**, first described in [38] is a collection of cells with forward and backward connections. As can be seen in Fig. 31, each cell of the CA can only connect to its adjacent neighbors, the connections being described as rules. The CA-based PRPG can be programmed as universal CA for generating different orders of test sequences [39].

As stated before, for the first BIST testing architecture for the IDEA NXT, the TPG was constructed using a counter, a plain LFSR and a Cellular Automata, to see which is the best to use or if there are major differences in electing one solution or the other.

4.1.1.2 Implementation of Output Response Analyzer

The Output Response Analyzer can either be implemented by means of one's counting, parity checking or signature analysis – which can be implemented by several Single Input Signature Registers (SISRs) or a Multiple Input Signature Register (MISR).

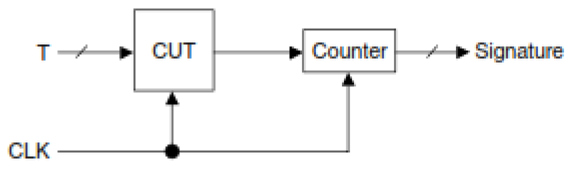


Fig 32 Ones counter as ORA [38]

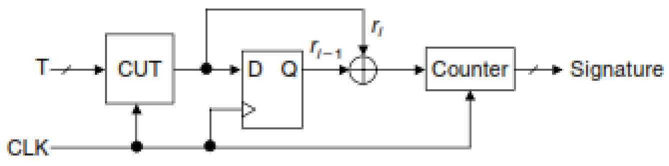


Fig 33 Transition counter as ORA

An Output Response analyzer comes in handy when using BIST techniques, because one cannot store all output responses on-chip, on-board, or in-system. An ORA compacts all responses into a signature and compares the result with a golden signature for the fault-free circuit which can be stored outside the chip or even on-chip. Output response analysis techniques are called Output response Compaction or Compression [40].

There are three main types of such techniques: transition **count testing, ones count testing** and **signature analysis**, the latter two being used in the test architecture I designed for IDEA NXT.

Ones count testing technique counts the number of 1's in the circuit's output bit stream and verifies it against the signature's one's count. Assuming all faulty sequences are equally likely to occur as the response of the CUT, the aliasing probability or masking probability (probability that the signature of a bad circuit will be the same as a good circuit) in using ones count testing is pretty significant.

Transition Count Testing uses a D flip-flop and an XOR gate connected to a One's Counter to count the number of transitions in the output data stream [16]. It is similar to One's Count Testing with the difference that its signature is defined as 0-to-1 and 1-to-0 transitions.

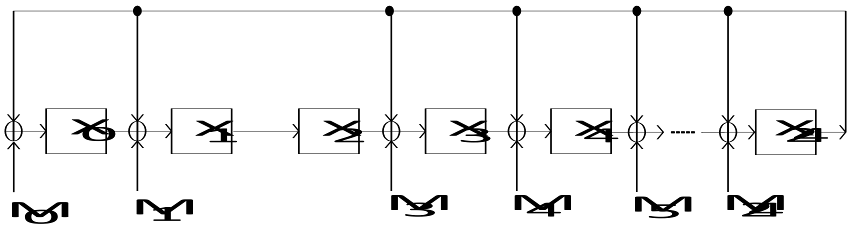


Fig 34 A 24-stage Multiple-Input Signature-Register (MISR)

Signature analysis is the most used response compression technique and is based on cyclic redundancy checking (CRC) [41]. Signature analysis was introduced in [42] and was first developed by Hewlett-Packard to test equipment in the field in the late 1970s.

There are two main types of signature analysis schemes:

- serial signature analysis for compacting responses from a CUT having a single output, also called **signal-input signature register (SISR)** which is in essence a CRC code generator a cyclical code checker
- parallel signature analysis for compacting responses from a CUT having multiple outputs, also called a **multiple-input signature-register (MISR)** – it is generally used to reduce the amount of hardware required for compressing a multiple bit stream; its functionality regarding the aliasing probability remains unchained for this implementation [43]. An example of MISR, adapted for IDEA NXT's characteristic polynom ($f(x) = 1+x+x^3+x^4+...+x^{24}$) is illustrated in Fig. 34.

In the end, after balancing the advantaged and disadvantages of all these possibilities, I chose to implement the ORA with **Multiple Input Signature Register,** as it will take two inputs: the value of the counter saying when the comparison must take place, and the output of the datapath as value to compare with the signature.

The testing strategy, as already stated, can be applied to a single encryption round or to the entire algorithm, requiring, in the latter case a complete run of all the 16 rounds. The advantage of the first solution is that the designer can select as many rounds of the algorithm as needed for execution during the test process thus offering flexibility to the test strategy.

4.1.2 IDEA NXT Feedback Loop Offline Test Architecture

The second testing architecture I propose is a Built-in Self-Test Design for Testability (DFT) solution in which input vectors are generated at each round of the IDEA NXT algorithm and the output responses are validated through a feedback interconnection by the system in which the algorithm is running. The repeated execution of the algorithm is managed in conjunction with a counter for verifying the final signature. This test approach was also applied to the AES [59] crypto-algorithm in [60] based on the concepts from [61].

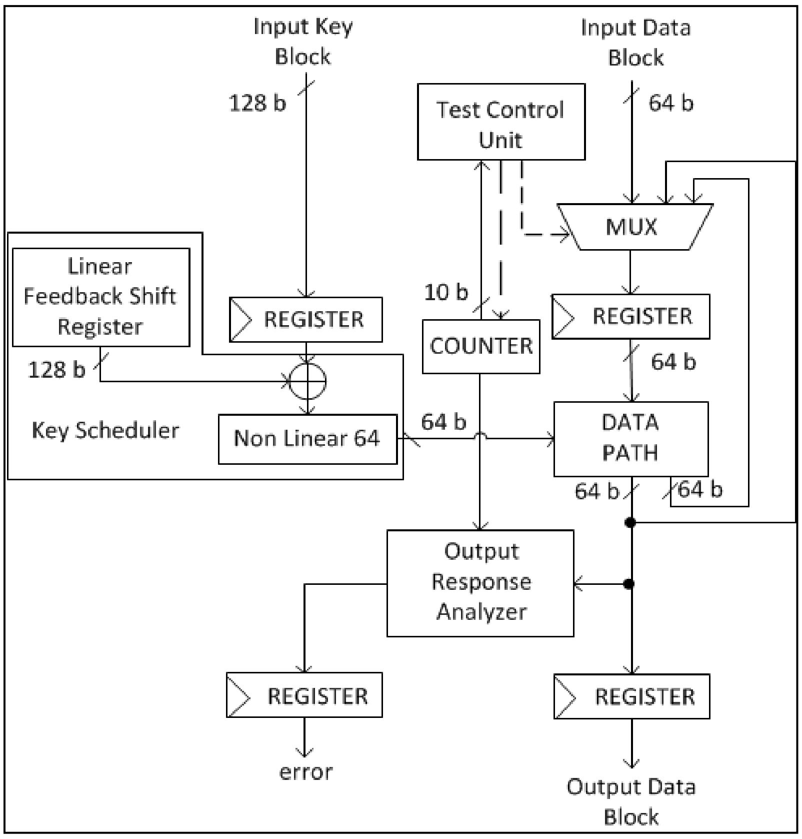


Fig 35 IDEA NXT Feedback Loop Offline Test Scheme

This offline BIST scheme I constructed can be depicted in Fig. 35. It is composed of a Test Pattern Generator (TPG), which generates stimuli vectors for the IDEA NXT Datapath. These stimuli are applied to the design for rendering evident different types of faults that can possibly affect the crypto-core and the result is verified with the help of a dedicated Output Response Analyzer built around a MISR.

The test architecture includes, as before, a Test Control Unit, which supervises the test process driving the ORA into accepting the responses generated at the circuit's output, one multiplexor (MUX), which selects between the 64-bit data block input, the round output (generated at each iteration by function lmor64) and the Datapath's final result (which will be lmid64's output). By using a multiplexing layer, we can run the tests either at algorithm level or just at the round level.

The result from the MUX is stored into a simple register delivering the partial processed block to the Datapath unit. The results from the Non-Linear step (NL64) of the Key Scheduler will serve as input for the Datapath, and the result obtained after the Datapath algorithm is run, on 64 bits, will be fed to the ORA for verification. The result of the IDEA NXT core also serves as input for the next encryption round, so if it is erroneous the fault will be propagated and amplified by the system for the next rounds, regardless if it will manifest in the next round or not.

If a fault is manifesting in the system, it will be detected by evaluating the signature associated with all the output responses generated for the input stimuli that were fed into the algorithm. The test architecture is responsible for evaluating the output vectors by compressing the results into a fixed-size vector called signature. The output is stored into a register, as can be seen in Fig. 35.

The proposed offline test architectures have the benefit of reducing the test flow length while introducing little hardware overhead as the experimental results reveal. The proposed test architectures can also be applied to the decryption process with almost no modifications, as just the round keys have to be generated in reverse order.

4.1.3 BILBO Offline Testing Architectures for IDEA NXT

McCluskey and Abramovici in [44] and [45] did an extensive study of architectures which incorporate logic BIST techniques into a circuit's design. They categorize them as follows:

- Architectures for which no special structures added to the circuit under test
- Architectures which incorporate scan chains into the design
- Architectures which configure the scan chains for test pattern generation and output response analysis
- Architectures which use the concurrent checking (or implicit test) circuitry of the design [51].

The first categories of BIST architecture make use of a pseudo-random pattern generator as well as a single-input signature register (SISR) or multiple-input signature register (MISR) in the validation of sequential and combinational CUTs with simple structures. Hewlett-Packard used this architecture in the 1970's for board-level fault diagnosis, and I did as well for the first two error-detection schemes we constructed.

An architecture that includes scan-chains in the designs is a LSSD on-chip self-test scheme in which the test scan chain is comprised of LSSD shift register latches and is connected to the scan output of the internal scan chain. Pseudo-random numbers are shifted into the combined scan-chain and the final result – a signature – is compared in the SISR with a pre-computed fault-free signature to generate a pass/fail error signal [6].

The most widely used BIST architecture that uses register reconfiguration is the Built-In Logic Block Observer (BILBO). This can be incorporated only in CUTs that can be divided into blocks or modules with their own input and output registers or storage which are redesigned to act like PRPGs for test generation or MISRs for signature analysis [47]. This type of register is called built-in logic block observer (BILBO). The BILBO can operate as one of the 4 below (at a time):

- PRPG
- MISR
- Parrallel load
- Shift register

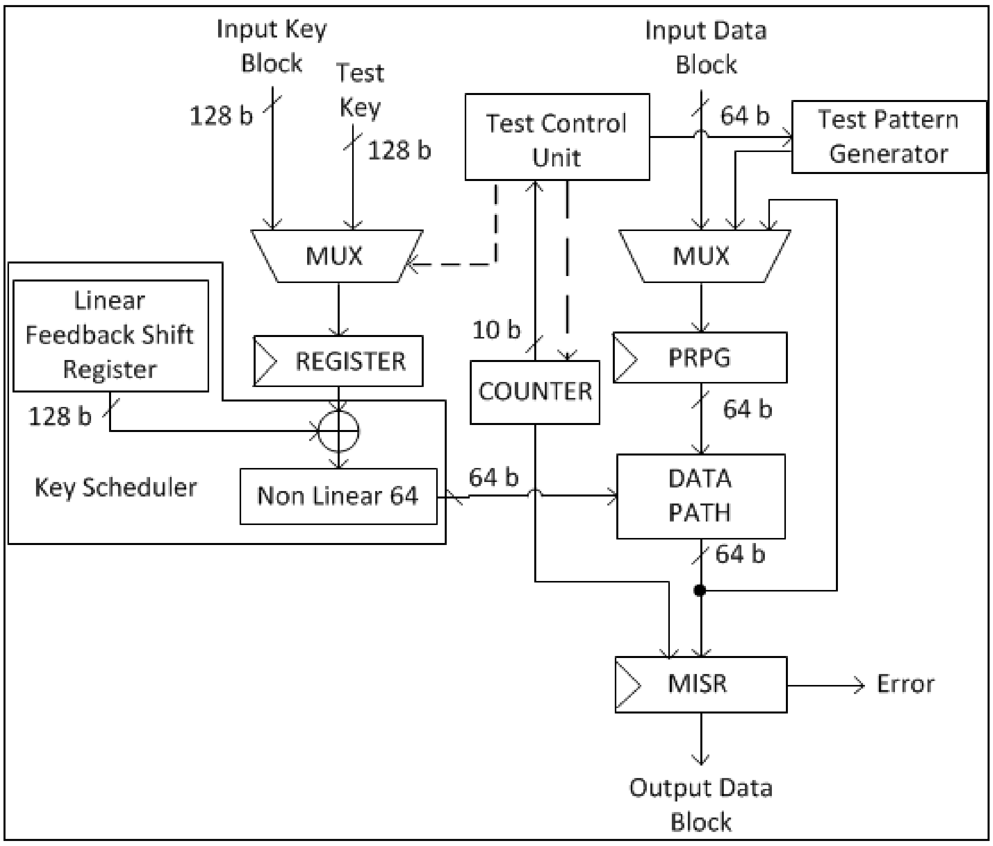


Fig 36 Error-detection architecture for IDEA NXT built with BILBO

BILBO is a test-per-clock BIST approach since a new test pattern is applied to the CUT and a new output response is compacted during each clock cycle of the BIST sequence.

BILBOs are usually used for testing RAM memories, ROM memories and other circuits where I/O registers can be reconfigured like mentioned above. In the case of BIST Architectures using concurrent checking, the circuit is used to verify the output response during offline testing.

After implementing the first two error-detection schemes presented above, I considered it appropriate to investigate the replace the regular BIST with a BILBO inside the offline test architectures, to see which technique gives the best results. In this sense, I designed an architecture extremely similar to the one in Fig. 35. The Key Scheduler part remains the same; as for the datapath design, the Test Control Unit is still there, as well as the counter counting the number of iterations that have passed until the comparison between the compacted signature must be analysed. The main difference in this scheme is the usage of a PRPG instead of the register where the result of lmor64 is stored at the end of each encryption round and a MISR instead of the second register, where the result of the lmid64 is stored (at the last iteration step). The PRPG will act as test pattern generator as it will provide sequences of pseudo-random numbers which will act as stimulus vectors directed at the input of the algorithm.

For the BILBO error-detection scheme, the Output Response Analyzer is no longer needed, as the MISR will be used to compact the algorithm's output - the signature - and to analyze this signature to conclude if an error infiltrated in the algorithm or not, in the same way as above. The TCU has the role of supervising the testing process, verifying the testing sequence and checking that the final signature has reached the MISR in order for the comparison with the known-correct value to take place.

The offline architecture build with BILBO is illustrated in Fig. 36. As will be evident in the chapter providing the simulation results after running all the error-detection algorithm I propose, the design with BILBO introduces a significant overhead in terms of area, so the plain BIST solution seems to be a better approach when constructing a testing architecture for the IDEA NXT crypto-algorithm.

4.2 A Parity-based Concurrent Error-Detection Architecture for IDEA NXT

4.2.1 Introduction to Concurrent Testing Architectures

Concurrent checking schemes are designed to detect a high percentage of all the possible errors that can occur during DUT's normal operation, which can be of various types: single errors, double errors, unidirectional errors, transient defects.

Typically, faults are modeled at the logic level by means of stuck-at defects since the failure (effect of fault activation and propagation) can be easily detected in terms of logic levels, unlike path delay defects that affect the propagation latency of the signals through the DUT. Moreover, a single fault can cause different types of errors to occur [58] and therefore, the designer is expected to design the test architecture as general as possible.

To the best of my knowledge no verification mechanisms have been implemented for the IDEA NXT crypto-algorithms family, nor offline or concurrent.

My goal was to increase the reliability of crypto-systems in which this algorithm is used, by creating a class of concurrent, self-testing architectures.

The fault detection principle I used is the non intrusive concurrent error detection mechanism from [64] based on the output's parity prediction. A parity detection mechanism is constructed around a DUT's module for which the output parity is checked against a predicted parity bit for that respective unit. This mechanism, similar to the one used of the AES algorithm [62] in [63], was first described in a paper [47] which I presented at the SOFA conference in 2014. In my implementations, as revealed in the experimental results chapter, I evaluated different levels of redundancy with respect to the number of parity bits.

The two extreme cases for the number of parity bits are single-bit parity and duplication. In single-bit parity, all output bits of the circuit are protected collectively by a single parity line. Duplication leaves the original circuit intact, incurring the additional cost of circuit's duplication for verifying the correctness of the original copy. In this context, the single-bit parity case is relatively inexpensive, as no redundancy is introduced. I used a reduced number of parity bits for my parity-checking architecture, analyzed in 3 distinct scenarios: 1 bit of parity associated with 4 bytes of the data processed by all units of the algorithm, 1 bit of parity protecting 2 bytes from the unit and 1 bit of parity associated with each byte.

The output parity prediction for a particular module consists of a mechanism for anticipating the parity of the output based solely on the module's input. By verifying the equality between the predicted parity and the actual parity of the output the architecture will detect any odd number of errors affecting the result of the protected module, while remaining completely independent from the DUT. This type of error detection fits well with the notion of integrated circuits that are designed to be totally self-checking with respect to a set of faults, as we can verify each stage and component of a cryptographic algorithm in the proposed manner.

As mentioned before, I constructed two concurrent architectures - for the Datapath and the Key Scheduler of the IDEA NXT algorithm in order for the whole algorithm to be checked for possible errors. The error detection mechanism will be described in detail in the following sub-chapters.

4.2.2 Error-detection mechanism for IDEA NXT's Datapath

As already described in the second chapter of the paper, IDEA NXT's Datapath consists of running (r - 1) iterations of the round function denoted as *lmor64*, followed by the application of a slightly modified version of it called *lmid64.* The concurrent architecture for the Datapath's *lmid64* is shown in Fig. 37. Both the IDEA NXT Datapath and the Key Schedule unit are designed to incorporate parity prediction modules. Ideally, the parity prediction channel would be completely decoupled from the modules it protects. In this manner, based only on input parity, the predictor is capable of anticipating the output parity. However, the completely decoupled solution is not always achievable as a reasonable tradeoff between error detection and complexity.

The concurrent testing scheme was constructed by adding two parity bits to the data block, denoted x_{lp} and x_{rp} each associated to the two 32-bit halves of the data block (denoted x_l and x_r in the diagram). Similarly, two parity bits were added to the 64-bit round keys used in the encryption process, denoted by r_{k0p} and $r_{k1p.}$

The difficulty of predicting the output parity appears in the case of the more complex operations, like *sigma4*, which consists of substitution boxes, and *mu4*, respectively, which uses complex operations in the GF(2^8). Regarding the parity prediction for the *sbox* instances, this is a complex problem due to operation’s non-linearity. The same observation is valid also for the orthomorphism (*ortho*) used in the *lmor64* function of the encryption as well as the inverse orthomorphism used by the *lmio64* function of the decryption process.For these complex operations we created a series of parity prediction modules which re-compute the value of the parity bits after operation's execution. A custom solution, tailored to the operation's specific implementation needed to be built for each of the prediction modules and will be presented in the following paragraphs.

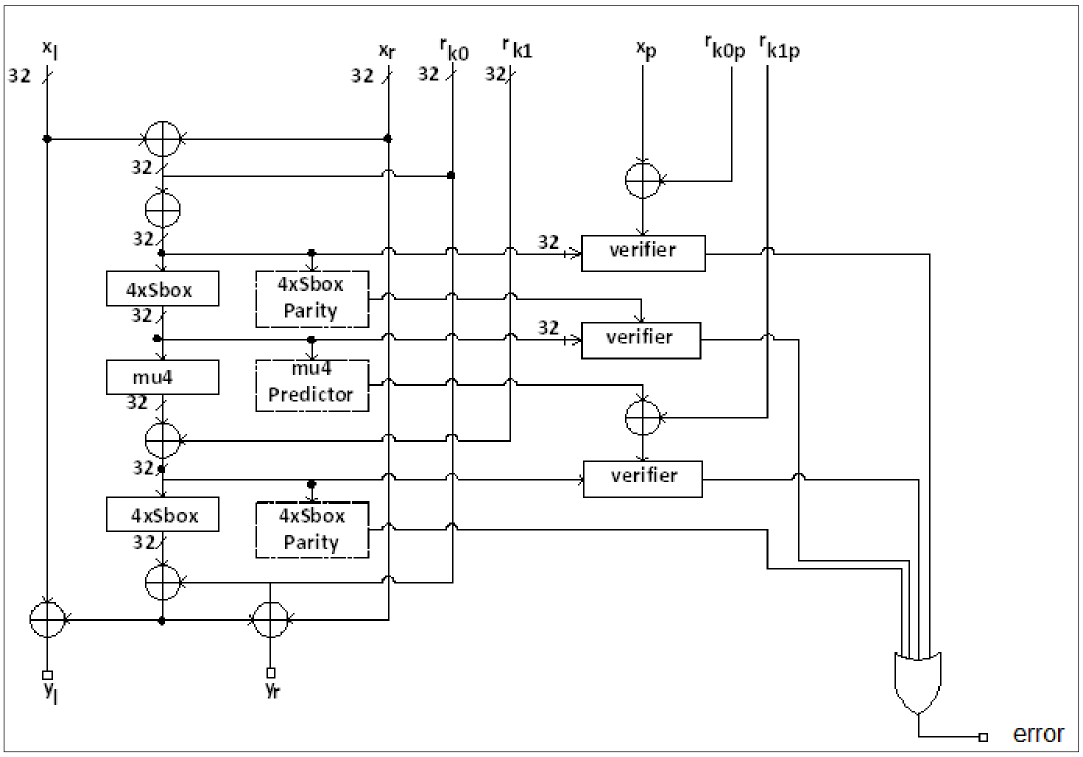


Fig 37- Parity-based test architecture for IDEA NXT's lmid64, part of Datapath and Key Scheduler

The prediction units are represented with dashed lines in Fig. 37 and Fig. 38 and their purpose pertains to generating the parity bits after execution of an operation for which the output parity cannot be predicted from the input parity. In such a case, it is investigated the input of the respective module for predicting the output parity. However, it must be assureed that no error occurred along the Datapath affecting the input to the respective module and, thus the module’s input correctness using the parallel parity channel must be verified.

For this reason verifier modules are added , whose error indicator are combined together for allowing to signal any discrepancy between the data lines and their associated parity bits.

As can be seen in Fig. 37, we created a testing scheme which operates in parallel with the algorithm's structure. The parity channel follows the exact same operations of the protected architecture for as long as the output parity can be predicted from the input parity. It is the case for the XOR modules.

The parity prediction of the *sigma4* module and of the *sbox* unit introduces irregularity into the parity-based verification architecture. *Sigma4* is composed of four *sbox*-es, taking as inputs 8 bits of data, and in consequence, for the case of using 1 parity bit associated to 4 bytes of data, calculating the parity of the operation is reduced to calculating the parity of each substitution box. Due to the transformation’s non-linearity, the output parity bit cannot be expressed in terms of sbox input bits and thus is realized by embedding an additional look-up table inside the module.

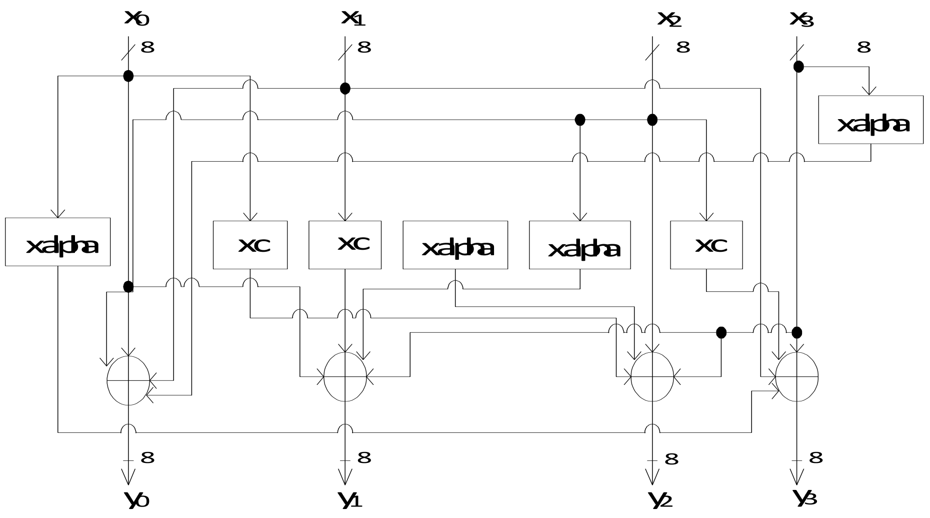


Fig 38 Parity check scheme for the mu4

The *sigma*4 output parity bit is obtained by summing up all the individual parity outputs. Since the parity bit is generated by predictors, as already explained, the *sigma*4 inputs are to be checked against errors by means of a verifier unit. Inside the verifier, *sigma*4’s input parity is computed by operating on all 32 bits with an XOR tree, obtaining a single parity bit which is then checked against the predicted parity bit run through the parity channel. After *sigma*4 execution, the parity bits must be recomputed from the current state in order for them to be reinserted into the parity channel.

The prediction for *mu4* requires a dedicated output parity prediction unit as it is an irregular operation. However, since the transformation can be described in terms of the linear XOR operators, unlike the case for *sbox*, the output parity of the *mu4* module can be express mathematically in terms of the input bits, based on function's internal transformations. The calculation of its parity bit was obtained by XOR-ing the bits from the four 8-bit-length outputs operation takes four 8-bit inputs

The scheme of the parity predictor we constructed is shown in Fig. 38. The *xalpha* unit multiplies the degree-8 polynomial associated with its input *i*, by the monomial x, operation performed modulo P(x), where P(x) is:

$$P(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1 \qquad (1)$$

The *xc* unit is similar to *xalpha*, the only difference is that it is using a different polynomial for multiplication, multiplication performed modulo P(x) from equation (1):

$$c(x) = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1 \qquad (2)$$

Another parity prediction module was implemented for the orthomorphism. The irregularity of this operation is visible in its defining equation from [2]:

$y_{(64)}$ = lmor64($x_{l(32)}||x_{r(64)}$) = OR ($x_{l(32)}$ XOR f32($x_{l(32)}$ XOR $x_{r(32)}$, $rk_{(64)}$) || ($x_{r(32)}$ XOR f32($x_{l(32)}$ XOR $x_{r(32)}$, $rk_{(64)}$) (3)

When employing 1 parity bit or more for each half of the *ortho* input, the output parity can be predicted right from the input parity due to operation's simple linear expression. Nevertheless, if using a single parity bit for the units input, the parity of the output cannot be predicted from input parity, requiring a dedicated predictor that generates the parity of the output using the input bits. If there were no XOR operation or if an XOR were applied symmetrically to the two halves, the parity of the outputs would be straightforward, but because of the XOR operation, a module for parity calculation was needed.

Fig. 40 illustrates the parity prediction scheme which was created for the orthomorphism. If we denote a_r and a_l the two halves of the input to the orthomorphismm and b_l and b_r the respective output halves, then the parity bit of this operation, denoted a_p, can be calculated like this:

$$a_p = \text{Parity}(\textit{ORTHO}(a_l, a_r)) = \text{Parity}(a_r \text{ XOR } (a_r \text{ XOR } a_l)))$$
$$= \text{Parity }(a_r) \text{ XOR Parity}(a_l) \text{ XOR Parity}(a_r) = \text{Parity}(a_l) \qquad (4)$$

This is the simplest parity prediction module in the entire scheme. As the orthomorphism is included in both Datapath and Key Scheduler, this parity predictor module will also be used in the error-detection scheme for the KS.

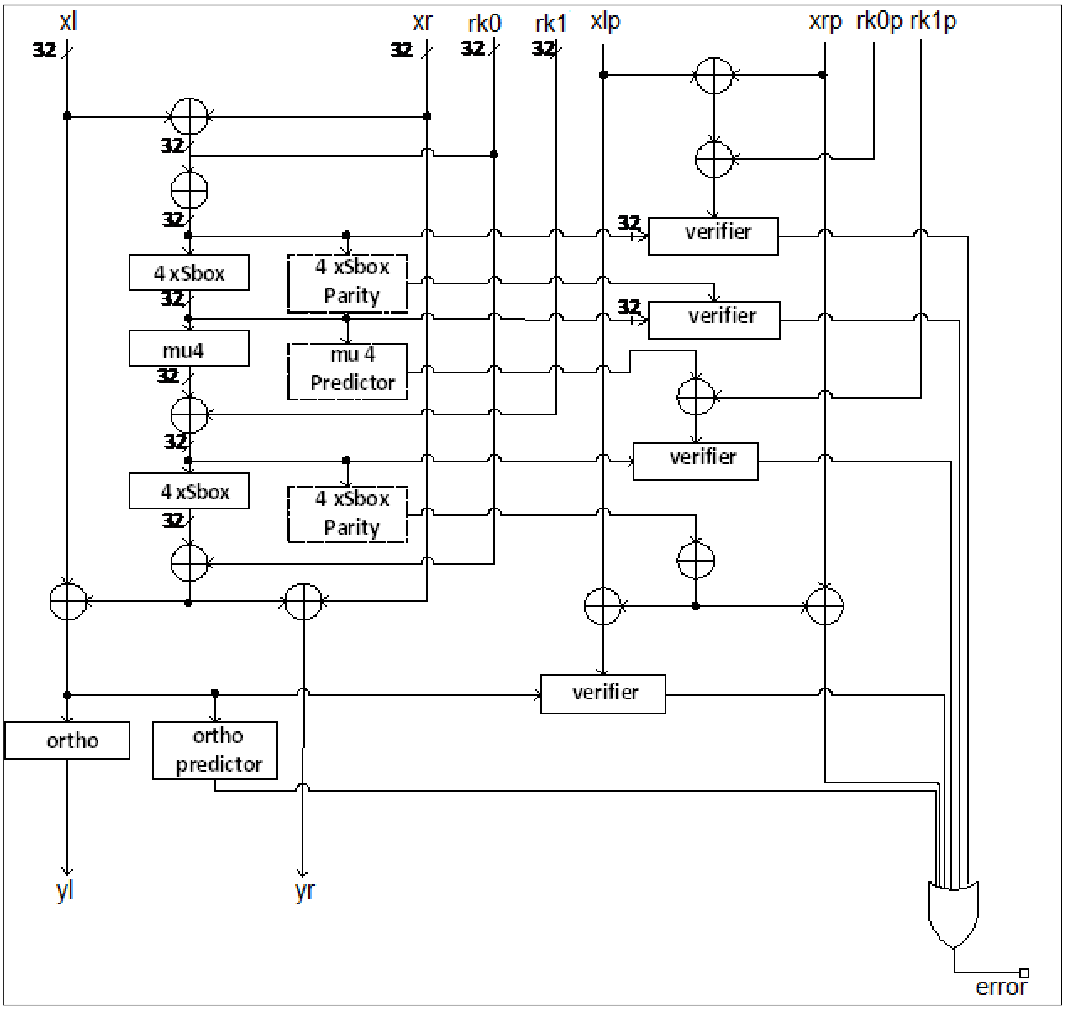


Fig 39 Parity-based test architecture for IDEA NXT's lmor64, part of Datapath and Key Scheduler

4.2.3 Error-detection mechanism for the Key Scheduler

IDEA NXT's Key Scheduler is mostly composed of the same operations as the Datapath, this is why the general testing architecture is very similar to the one we constructed for the Datapath, as can be seen by analyzing comparatively Fig. 39 and Fig. 37. When using 1 parity bit for each 4 bytes we will associate two parity bits for the round key, rk_{0p} and rk_{1p}, and only one parity bit for the data, denoted $x_{p.}$, where $x_p = \Sigma x_i$ (x_l, x_r denoting the two input halves for the Key Scheduler). The first parity bit of the error-detection scheme is calculated by doing an XOR between the two halves of the input data bytes and summing up the results of this for each round:

$$x_p = \Sigma a = \Sigma (x_l \text{ XOR } x_r) \qquad (5)$$

The output is furthered XOR-ed with rk0 and so the new intermediate parity bit will be the sum of these results:

$$\Sigma b = \Sigma a \text{ XOR } \Sigma rk_{0p} = x_p \text{ XOR } r_{k0p} \qquad (6)$$

For the *sigma4* and *mu4* operations we can use the *sbox* and *mu4* parity predictors which I already developed for the Dapatah error-detection scheme. After a parity bit is calculated for each of them, we must check for correctness with a *verifier* module to see if the parity data is correct. Three verifiers are used in the scheme. For the complementation operation, which takes place during key generation process, the parity bit doesn't change its value.

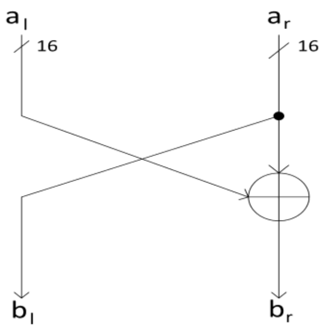


Fig 40 Parity prediction scheme for Orthomorphism

An element which appears in the Key Scheduler but not in the Datapath is the LFSR. Calculating the parity for the series of pseudo-random number generators is not a trivial task, and in consequence a Parity Predictor needed to be constructed for it also. A parity bit is generated for each group of 8 bits generated by each of the 6 24-bit LFSRs and they are combined correspondingly to assure the necessary parity data bits. Mention should be made that only 8 bits of the final LFSR are used. For the case of 1 parity bit associated to 32 data bits, out of all 128 bits generated through LFSRs, 4 parity bits are generated whereas for a redundancy level of 1 parity bit associated to 1 data byte, 16 parity bits are generated for the LFSR output.

All these parity predictors work together to achieve the common goal of calculating the parity at the algorithm’s output so as to detect any odd number of errors affecting the result of the protected module (for a detected odd parity).

4.3 Fault Injection for IDEA NXT

The main purpose of testing architectures is finding faults and other means by which the system under test's integrity could be compromised. This being said, the effective

eness of such an architecture can be measured by the number and/or variety of faults and defects it finds, and the moment in which those defects are found (the sooner they are discovered, the less time and money will take for the system's repair). Once such an architecture is designed, it is useful to test its effectiveness in an early design stage, before using it in a real system. One technique which validates the dependability of an error-detection scheme is **fault injection**.

4.3.1 VHDL-based Fault-Injection techniques

Fault injection was defined in [119] as the validation technique of the Dependability of Fault Tolerant Systems, which consists in the accomplishment of controlled experiments where the observation of the system's behaviour in presence of faults is induced explicitly by the written introduction (injection) of faults in the system. As already stated in Chapter 3.3 of this thesis, there are three ways of injecting faults in a hardware system: hardware implemented fault injection, software implemented fault injection and simulated fault injection.

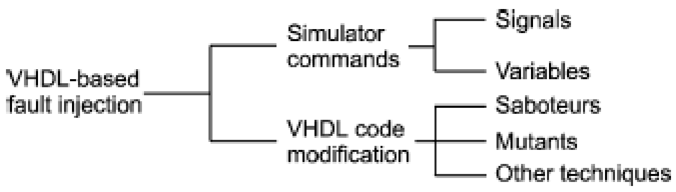


Fig 41 VHDL-based fault injection techniques [120]

The goal was to inject stuck-at-0 and stuck-at-1 defects into the testing schemes I built for the IDEA NXT. Since the error-detection architectures I build for the IDEA NXT crypto-algorithm were modeled in a hardware description language, I considered appropriate to investigate in more detail the *simulated fault injection* techniques. Simulation-based fault injection is a widely-used experimental technique for evaluating the dependability of a system during the design phase. An early diagnosis allows saving costs in the design process, avoiding redesigning in case of error, and thus reducing the time-to-market, as stated in [119]. Another strong point of these techniques compared to others is that those based on simulation offer both high observability and controllability of all the modelled components [121].

There are two main types of simulation fault injection techniques, as can be seen in Fig. 41:

- using *simulator commands* to modify the value of the model signals and variables without altering the VHDL code
- changing the VHDl (or Verilog) code using *mutants* of the system components / adding *saboteurs* between different components to alter the values of one or more input signals [120]. The technique of the sabouteurs

and mutants is also used for evaluating the fault tolerance algorithms and methodologies designed for quantum systems.

Using the first technique mentioned above, faults can be injected on either signals or variables; on signals both transient and permanent faults can be injected, but on variables one can only inject permanent faults, as can be depicted from the two pieces of pseudocode below [120]:

```
Step 1. Simulate_Until [injection instant]
Step 2. Modify_Signal [name] [faulty value]
Step 3. Simulate_For [fault duration]
Step 4. Restore_Signal [signal name]
Step 5. Simulate_For [observation time]
```

Fig. 42 a) Transient Fault Injection on Signals

```
Step 1. Simulate_Until [injection instant]
Step 2. Assign_Variable [variable name] [fault value]
Step 3. Simulate_For [observation time]
```

Fig. 42 b) Permanent Fault Injection on Variables

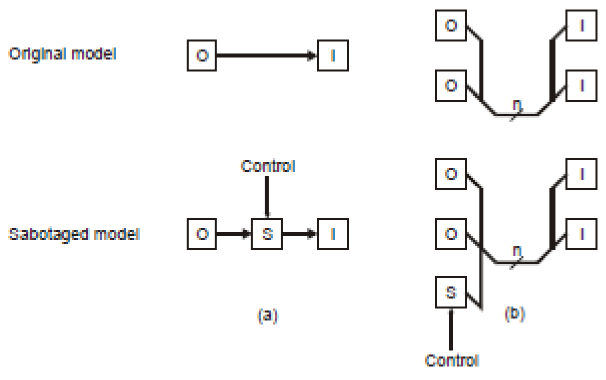


Fig 42 Different saboteur types: serial (a) and parallel (b)

Saboteurs are VHDL components inserted into a design to simulate fault injection by modifying signals's values; they remain inactive otherwise. There a various types of saboteurs:

- *serial simple* - interrupts the connection between an output (driver) and its corresponding receptor (input), modifying the reception value
- *serial complex* - interrupts the connection between two outputs and their corresponding receptors, modifying the reception values
- *parallel* - added as an additional source of a given signal [121]; implementing them is more complex and also they allow for fewer types of faults to be injected, so they are not as widely used as the serial ones

- *bi-directional serial simple/complex or parallel* – these were introduced in [120].

  The internal architecture of the saboteurs can be behavioural (a process whose sensitivity list contains the control and input/output signals) or structural (based on the use of multiplexers) [120].

Mutants are the names given to components which replace the original components of a design when wanting to simulate the behavior of that component in the presence of faults. If a mutant component is not activated, then the component behaves normally. This technique can be used in three distinct ways:

- adding saboteurs to structural model descriptions
- modifying structural descriptions by replacing subcomponents (i.e. replacing a NAND gate by a NOR gate, *if*-clause by a *case*-clause)
- modifying syntactical structures of behavioural descriptions

A common way of injecting permanent faults is to use VHDL's configurations, where mutant architectures can be binded to components for simulating fault injections, respectively binding non-altered versions of the atchitectures to components. The implementation of transient faults by means of mutants technique requires to carry out dynamic instantiation. A possible solution to achieve this is to use guarded blocks, as discussed in [122]. A guarded assignment is an assignment expression on a signal, conditioned by a boolean expression called guard which, if it is true, the assignment is executed and if not, a null assignment is generated.

4.3.2 Verilog-based Fault Injection Techniques for the Experimental Validation of a Fault-Tolerant System

The VHDL concepts above needed to be adapted for the Verilog language in which the error-detection schemes were implemented. This was mostly a four-step process, as described below:

1. Transform the design so that all components would become low-level elements such as logic gates and buffers
2. Calculate the number of primary gates and the number of stuck-at faults which could be injected in each testing scheme in order to see what number of injected faults could be considered a suitable sample which would confirm the correctness and efficiency of the design
3. Set the above calculated number of input and / or output gate ports to value '0' or '1' to simulate stuck-at-0 and stuck-at-1 faults; the gates would be chosen randomly
4. Check the output of the mutant architectures to verify their effectiveness in discovering faults

Chapter 5 Experimental results

The previous chapters of this thesis focused on presenting the theoretical aspects of the this research, as well as the state-of-the-art in the domains of interest (cryptography and testing mechanisms). The following paragraphs aim to analyze in an experimental manner the theoretical topics presented beforehand, namely the speed improvement brought to IDEA NXT, the error-detection architectures built specifically for this crypto-algorithm and the fault injection techniques used to demonstrate the efficiency of these architectures.

5.1 FPGA Devices

In order to evaluate the performance of the proposed speed improvement resulted from the changed I applied to IDEA NXT's Key Scheduler, which was presented in Chapter 2, I implemented the IDEA NXT algorithm in Verilog and synthesized for the Altera Cyclone II EP2C35 FPGA, using the Quartus II tool.

FPGAs are dedicated integrated circuits defined in [18] as an aglomeration of programmable logical blocks and programmable interconnections in a regular structure which goes by the name "FPGA fabric".

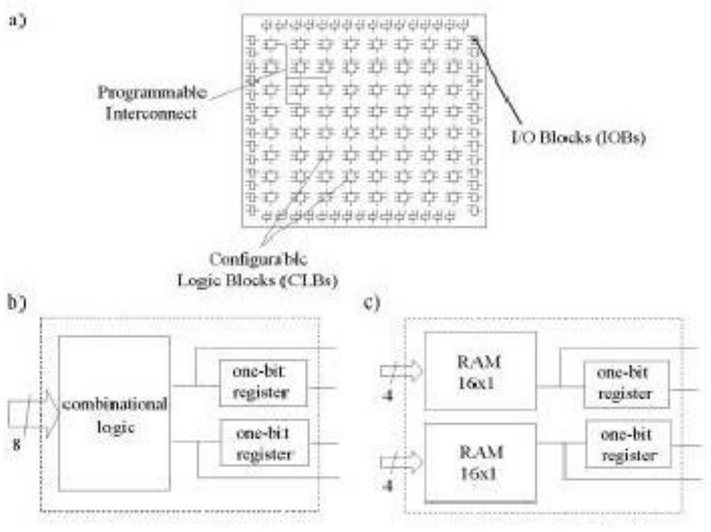


Fig 43 FPGA a) General structure. b) Internal structure of a CLB configured in the 'logical' way. c) Internal structure of a CLB configured in the 'memory' way

The device's architecture is composed of the so-called logic elements or configurable logic blocks (LE or CLB) which encompass both combinational logic (i.e. gates) as well as sequential logic (flip-flops and latches) connected through programmable interconnections, like Fig. 10 a) shows. CLBs are surrounded by input-output blocks (IOBs). These ports are in general programmable in order to offer input, output or bidirectional signals, but also facilities like low-power or hyperspeed.

All FPGAs need to be programmed. The configuration information is stored with the help of three technologies which determine the type of FPGA: SRAM, antifuse and flash, the disadvantage of the first being the fact that the device must be configured each time it is turned on [94].

Such a device must be designed at various levels of abstraction in a so-called a "Hierachical design" [18]. Starting from the requirements, refining the idea to many levels of detail, the designer must reach an architecture which can then be extended in logical implementations. Fig. 44 illustrates the block diagram of the DE2 platform.

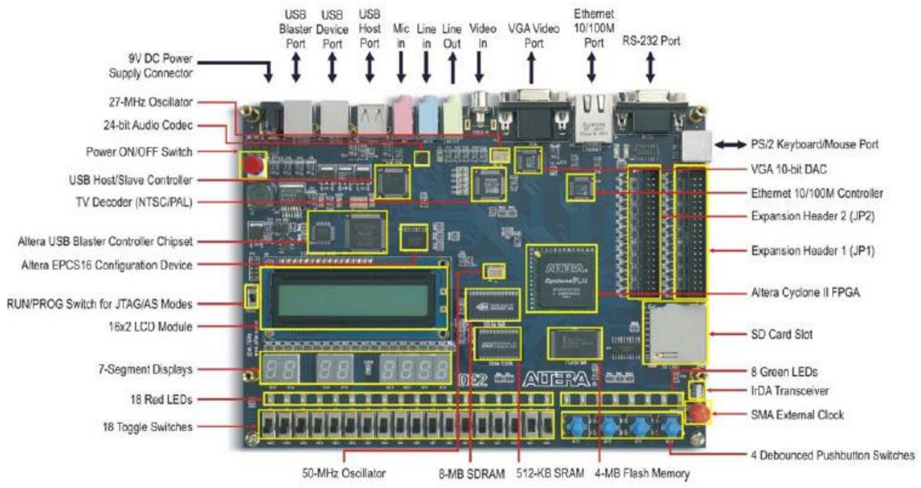


Fig 44 Altera DE2 Platform

For maximum flexibility, the connections are made through the FPGA Cyclone II device, so the user can configure the FPGA to implement any system design. The platform comes with a facility of the control panel which allows the user to access numerous components on the board through a USB connection from a host computer. The platform contains a serial EEPROM chip which stores the configuration data for Cyclone II, the configuration being loaded automatically in the FPGA each time I apply voltage to the board. By using Quartus II it's possible to reprogram the FPGA in any moment and to change the non-volatile data stored in the EEPROM.

The 18 switches are used as data inputs for the circuit. Each switch is connected directly to a pin of the FPGA. When the switch is in the "DOWN" position (the nearest to the board) it supplies a Low voltage level (0 volts) to the FPGA, and when it is in the "UP" position it has the logical level "1 (3.3 volts). There are also 27 controllable LEDs, located above the 18 switches and 8 leds situated above the 4 "toggle switch" buttons. Each LED is acted upon directly by a pin as follows: a high logic level lights the led while a low level quenches it. The pin assignation can be found in Altera DE2's datasheet. The board also includes 2 oscilloscopes which produce clock signals of 27MHz and 50MHxz.

The LCD module is built in a LSI controller, which has two 8-bit registers, one for instructions (IR) and one for data (DR). IR stores instructions like deletion of the display or the cursor movement, as well as addressing information for DDRAM and CGRAM. The DR register temporarily stores data which will be displayed, represented by codes of 8-bit characters. The LCD has 2 lines of 16 characters each, the maximum number of characters which can be displayed being 128. In the tables below it ius shown the relation between the DDRAM addresses and the positions on the liquid crystal display.

5.2 Time comparison between different versions of AES, the original and the speed-up versions of the IDEA NXT crypto-algorithms

A general version of the IDEA NXT crypto-algorithm with the LFSR modified as described in Chapter 2 was implemented in Verilog and synthesized for the Altera Cyclone II FPGA, using the Quartus II tool, as stated earlier. The synthesis results are presented below. For the compilation and simulation of the implementation altera's Modelsim tool was used.

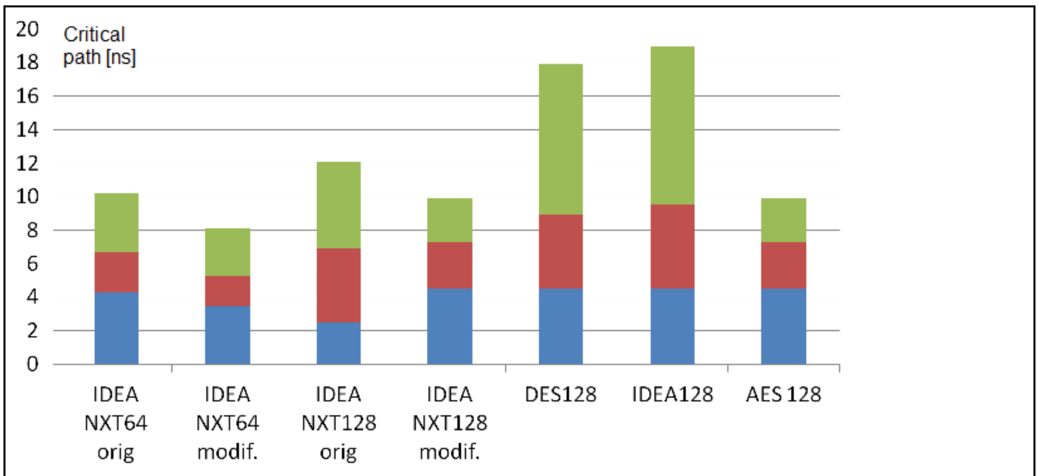


Fig 45 Time comparison between small-bit versions of DES, IDEA, AES and the original and modified IDEA NXT crypto-algorithms

Modelsim is a widely-used logic simulation tool for verification and debugging of digital circuits. Altera provides a version of ModelSim software, which includes libraries for Altera's FPGA devices [20]. The simulation permits the visualization of the signal's binary values as waveforms which are low if the value is '0' logic and are high if the signal's value is '1' logic.

In order to evaluate the performance of the proposed speedup solution, I compared this implementation with the hardware implementations of the original

64, 128 and 256 -bit versions of IDEA NXT I had proposed in a previous paper [17] and also with the similar implementations of the AES, DES and IDEA crypto-algorithms [17].

As Fig. 45 shows, there is a significant speed improvement in the algorithms's critical path when considering smaller text and key lengths (64-bit, 128-bit respectively). The modified version of NXT64 is 12% faster than the original version of NXT64, whereas the 128-bit modified version of the algorithm shows a speedup of 8% when compared to its unmodified version, and reaches the execution time of the 128-it version of AES. The 128-bit versions of DES and IDEA crypto-algorithms are clearly left behind in terms of execution times, being 8%, respectively 9% slower than the modified IDEA NXT 128.

When large key and text sizes are considered, the speed-up is not as significant as in the smaller length versions, but nevertheless I obtained a 9% improvement compared to the original NXT256 version, as it can be depicted from Fig. 46.

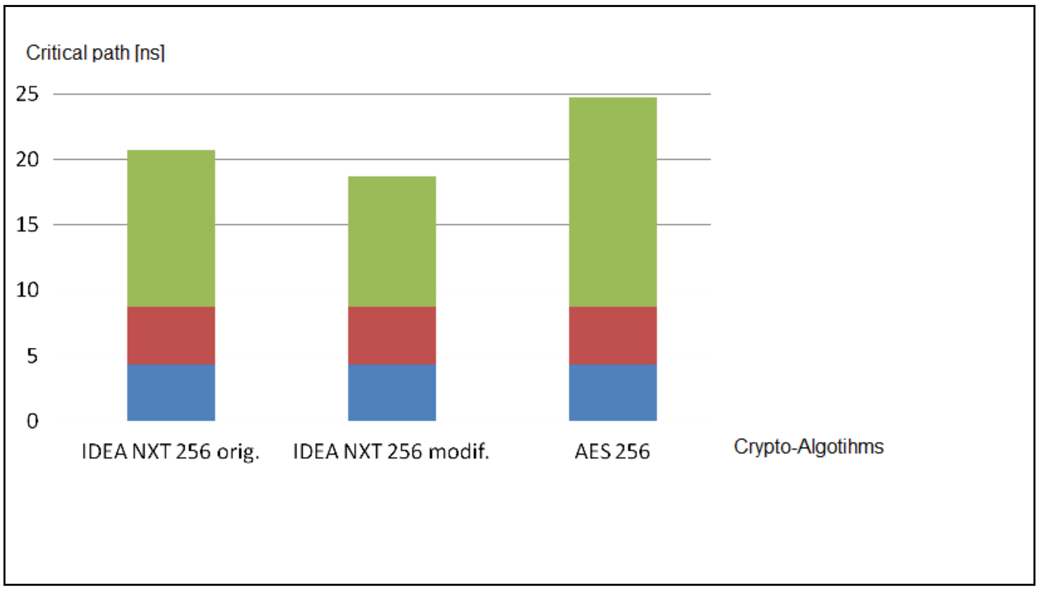


Fig 46 Simulation comparison between large-bit versions of AES and the original and modified IDEA NXT crypto-algorithms in terms of critical path

This sub-chapter illustrated a method to improve the execution time of the new family of crypto-algorithms, IDEA NXT, which works for all versions of the algorithm independent of the key and text length. The improvement consisted in modifying the LFSR equations used in the round key generation process, as to generate one key per clock cycle instead of one key in six clock cycles. The streams of pseudo-random numbers which make up the round keys are needed in both the encryption and decryption processes of the algorithm, to assure a superior level of security.

5.3 Experimental Results of IDEA NXT Off-line Testing Architectures

This section covers the implementation aspects for the testing strategies that were presented beforehand, as they were applied to the 64-bit IDEA NXT encryption/decryption process. The encryption, decryption and the dual architecture performing both the encryption and decryption were modeled using the Verilog hardware description language. The IDEA-NXT encryption architecture was extended with the offline test architectures presented in this thesis and the designs were synthesized and implemented using Xilinx ISE 14.7, for the Xilinx Virtex 4, XC4VSX35 Field Programmable Gate Array (FPGA) [63]. The synthesis and implementation steps were performed with maintaining the designs' hierarchy. The metrics of interest were considered to be: the area requirements (total number of Slices used), the critical path length (and consequently the maximum clock frequency) as well as the throughput, which is calculated according to the formula below:

*Throughput [Mbps] = data block size in bits * maximum frequency / (number of rounds)* (2)

Table 3 IDEA NXT VS AES BASE IMPLEMENTATIONS PERFORMANCE

Crypto-algorithm	**Area [slices]**	**Critical path [ns]**	**Throughput [Mbps]**
IDEA NXT64 Virtex original encryption architecture	4705	39.33	101.68
IDEA NXT64 Virtex enc-dec architecture	4749	41.04	91.71
AES encryption	2079	7.65	1520.89
IDEA NXT64 Virtex modified encryption architecture in [120]	4690	30	132
IDEA NXT64 ASIC encryption architecture in [120]	9562	231	924

The results obtained for the hardware implementation of the IDEA NXT algorithm (the one where the speed-up was performed) will be presented first, then the same metrics will be analyzed in respect to the offline error-detection architectures, then to the parity-based testing architectures, in order to see how much overhead was induced with the testing modules which I added and conclude if the proposed architectures' designs are well constructed or need to be further optimised.With respect to the basic IDEA-NXT architectures (encryption, decryption and dual designs) the implementation results for the architecture presented in [64] of the widely-used AES crypto-algorithm were offered as reference. AES is also a symmetric cipher with 128 bits for the input data block and keys on 128, 192 and 256 bits.

As can be observed in Table IV, when a complete encryption-decryption process is run the area requirements grow, as well as the length of the critical path and in consequence the throughput is diminished. However, the value with which it grows is not that significant.

The obtained results were also compared with similar hardware designs of IDEA NXT which were presented in [120]. Their designs were synthesized using Spartan-IIE and Virtex-II Pro FPGAs. The hardware utilization in their case is much less in the Spartan design because it is only designed for one round and would require 16 clock cycles to fully encrypt/decrypt the data once data and round keys have been loaded. The one-shot Virtex design can easily be compared to our implementation of IDEA NXT encryption and also to the encryption/decryption design because it also has sixteen combinational rounds and needs only one clock cycle to complete the process [120]. In table IV it can be observed that their Virtex implementation has a larger throughput, because of the larger critical path, but also a larger design space than ours, so what they gain in speed they lose in area requirements.

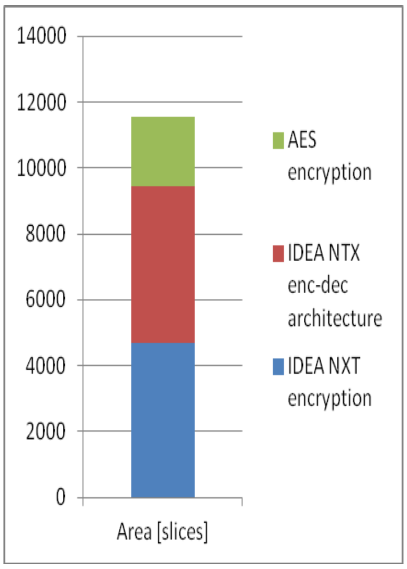


Fig 47IDEA NXT vs AES base implementation Performance in terms of Hardware Area

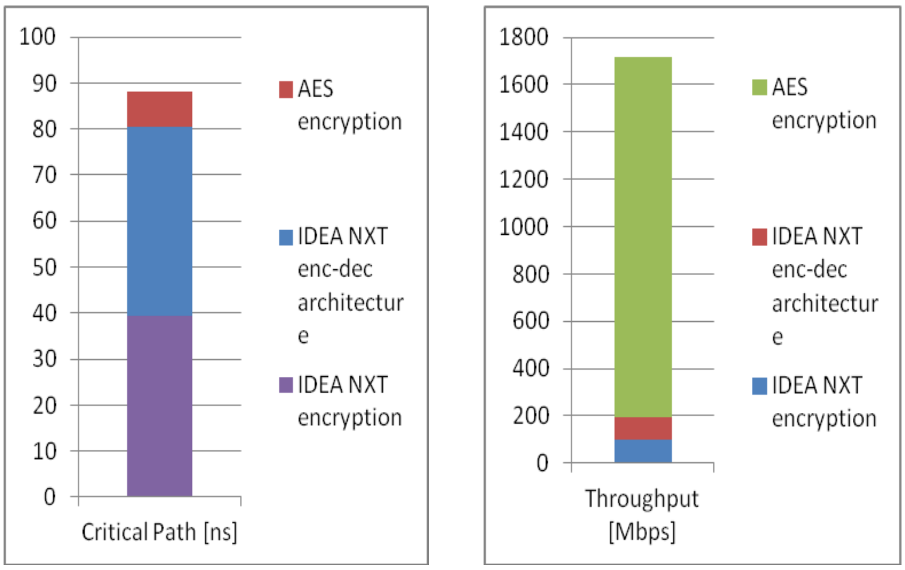


Fig 48 IDEA NXT vs AES base implementation Performance in terms of Critical Path & Throughptut

Table 4 IDEA NXT FEEDBACK LOOP TEST RESULTS

	Area [slices]	**Critical path [ns]**	**Throughput [Mbps]**	**Area / Throughput**
NXT base architecture	4705	39.33	101.68	46.27
Round-level NXT feedback loop testing	4769	40.07	99.82	47.77
	Area overhead: 1.36% Critical path overhead: 1.86 %			
Algorithm-level NXT feedback loop testing	4836	40.99	97.56	49.56
	Area overhead : 2.87% Critical path overhead: 4.22%			

Analyzing the critical path length of the encryption, IDEA NXT proves to be slower than the AES counterpart. The difference in terms of area overhead by comparison to AES comes mainly from the fact that the IDEA NXT is far more complex, more than two times larger in terms of the number of slices occupied on the FPGA.

As far as the throughput is concerned, there is a significant difference between AES and NXT, as can be seen in Fig. 48, because of the great difference of critical path (the inverse of the maximum frequency) of the two; however, the difference between the entire encryption-decryption process versus the plain encryption is not that significant.

Table V shows the area overhead as well as the latency degradation for both the round-level and the algorithm level feedback loop testing methods with respect to the base IDEA NXT architecture. The area overhead is not significant, as it is under 3% in both cases while the latency increase is under 5%.

The experimental results for the test architecture based on TPG are depicted in Table VI. It can be observed that the area overhead of the BIST solutions (with TPG implemented as Cellular Automata, Counter and LFSR) is higher than the one introduced for the feedback loop testing architectures, incurring an overhead of just 5.2%, respectively 5.3%. The critical path degradation ranges between 5.3% for the implementation of the TPG as a counter to 6.2% for when the TPG is implemented using a LFSR.

Table 5 IDEA NXT BIST TEST RESULTS

	Area [slices]	**Critical path [ns]**	**Throughput [Mbps]**	**Area / Throughput**
NXT base architecture	4705	39.338	101.68	46.27
Round-level NXT Cellular Automata testing	4950	41.744	95.82	51.65
	Area overhead: 5.20% Critical path overhead: 6.11 %			
Round-level NXT Counter testing	4959	40.844	97.93	50.63
	Area overhead: 5.39% Critical path overhead: 3.82%			
Round-level NXT LFSR testing	4950	41.777	95.74	51.70
	Area overhead: 5.20% Critical path overhead: 6.20%			

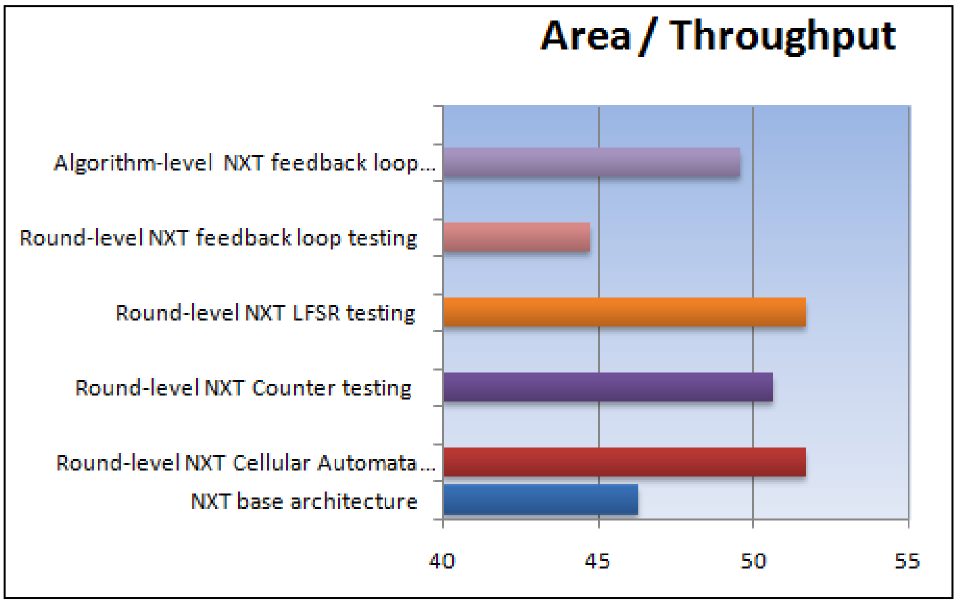


Fig 49 IDEA NXT Offline Architectures Results (for both BIST and Loop Interconnection) in terms of Area/Throughput in comparison to the base architecture

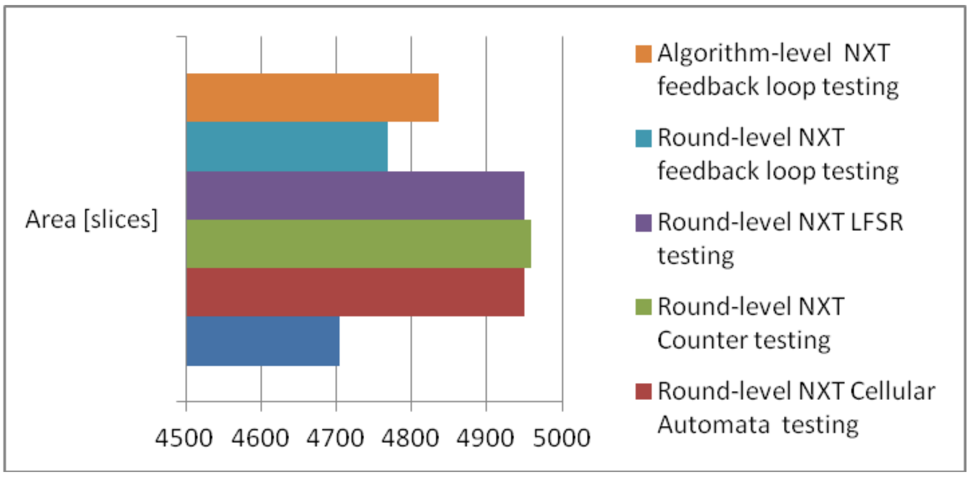


Fig 50 IDEA NXT Loop Architecture Area Results

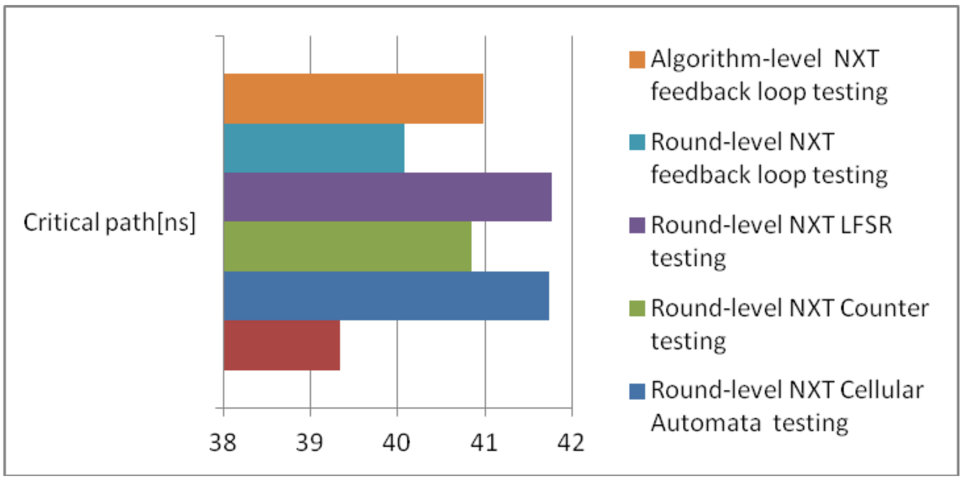


Fig 51 IDEA NXT Offline Architectures Results (both BIST and Loop Testing) in terms of Critical Path

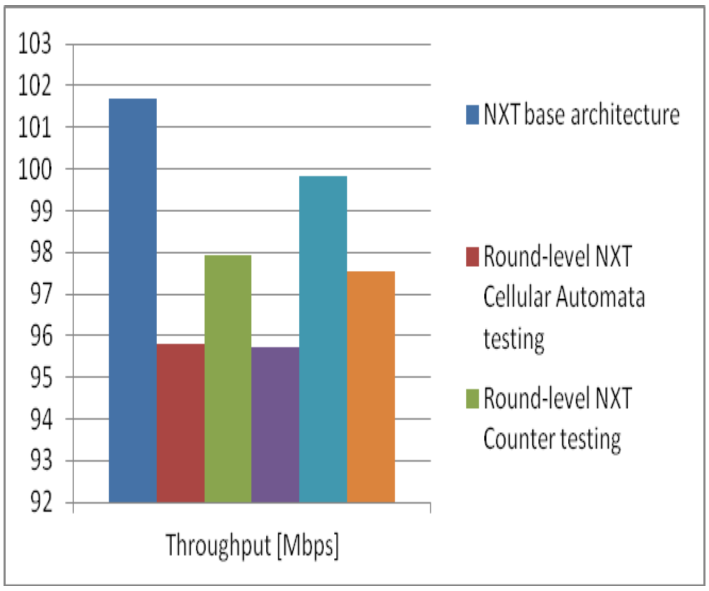


Fig 52 IDEA NXT Offline Architectures Results (both BIST and Loop Testing) in terms of Throughput

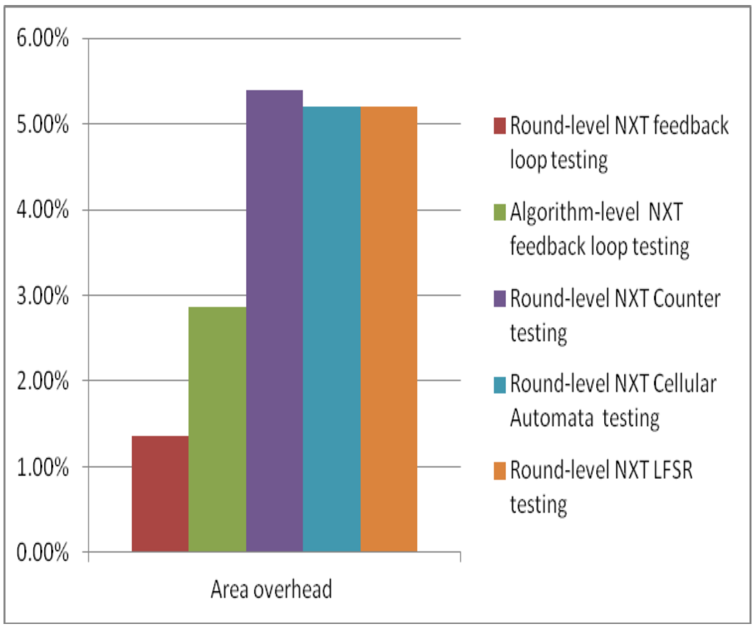


Fig 53 IDEA NXT BIST Results for induced Area Overhead

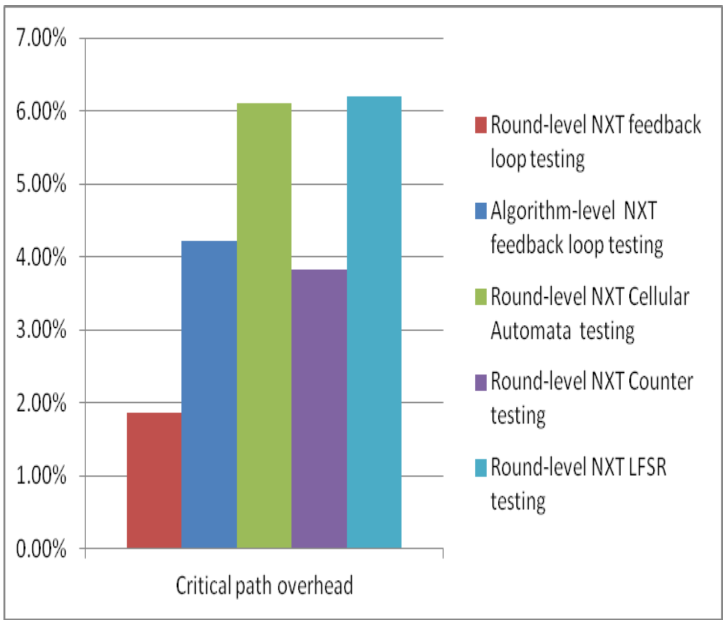


Fig 54 IDEA NXT BIST Results for induced Critical PathOverhead

As can be seen from the experimental results in Table VI, IDEA NXT's critical path's length is larger than AES's. The modules inside the key generation unit are responsible for the largest portion of the algorithm's critical path's length; namely: 2 layers of sigma4, an additional mu4 stage, a complete lmor64, followed by lmid64, besides several other XOR layers, as can be seen from Fig. 49-54.

I also defined a composed metric: Area divided by Throughput, in order to evaluate the efficiency of the error-detection schemes' implementations. This metric was chosen to be able to classify the implementation with respect to both the area requirements and the execution performance. The higher the obtained score for this metric the more efficient the usage of the FPGA resources is obtained for the respective design. As can be observed in Tables V and VI, all IDEA NXT testing designs perfomed better in comparison to the NXT base architecture, even though the difference is not that large (~5% in all cases).

In both the online scheme presented here as well as the experiments conducted for the offline architectures, I could not compare myself to other error-detection schemes for IDEA NXT as there were none I found in the literature at the moment of writing this thesis, so I went ahead and compared the values obtained for our testing architectures to the ones obtained for the NXT base architecture synthesis design on FPGA.

Figures 49-54 display in a graphical manner the experimental results comprised in Tables V and VI for all offline architectures built for the IDEA NXT 64 algorithm in regards to the chosen metrics as well as the introduced overhead in terms of area and critical path relative to the base version of the algorithm (which has no error-detection modules attached).

The best solution for implementing the BIST architecture in terms of critical path induced seems to be using a Counter as TPG instead of cellular Automata or LFSR; this is also obvious when having in mind that a counter is has the simplest structure of the three variants and takes the lest logical elements on the board; the other two are more complex and take more time due to their more complex operations taking place inside. The critical path of the single round-level feedback loop testing is clearly taking less time than letting all iterations of the algorithm run, but as can be seen in Fig. 54, the difference in critical path overhead between the two is just 50%, which is acceptable. The same can be said when analyzing these two architectures performance in terms of introduces area overhead, whereas in case of the three BIST error-detection architectures, the differences in area overhead are neglijable.

5.4 Experimental Results for IDEA NXT Parity-Based Test Architecture

The following paragraphs will analyze the experimental results obtained by synthesizing the on-line test architecture I designed for the IDEA-NXT64 algorithm. Apart from the parity-based error detection approach that associates 1 parity bit to each group of 4 bytes of the encryption process, I also evaluated the performance of the constructed architecture when using more redundancy bits. More precisely, I investigated the effect of using 1 parity bytes for each pair of 2 bytes as well as using 1 parity bit associated with each byte of the Datapath and Key Scheduler.

Besides the increased error detection capability associated with higher redundancy levels, because of the particular aspects of the IDEA NXT64 algorithm as well as of the concurrent error detection architecture, the higher the redundancy level of the architectures the faster it performs, at the expense of a larger design, as the experimental results reveal.

Table 6 IDEA NXT64 parity-based architectures synthesis results

IDEA-NXT Architecture	Max Frequency [MHz]	Area [Slices]	Throughput [Mbps]	Throughput/Area [Mbps/Slices]
Base	37.517	5189	150.068	0.029
Parity Checked 1 bit	34.732	6045	138.928	0.023
Parity Checked 2 bit	37.368	6064	149.472	0.025
Parity Checked 4 bit	37.474	6070	149.896	0.025

The architecture employing 1 bit of parity for each group of 4 bytes has the largest critical path compared with the solution employing 4 bits of parity for the same data size, as evident from Table VII.

The reason for the degradation of device's performance with reduction of the redundancy level, is partly due to the complexity associated with the *verifier* modules and, for the case of 1 parity bit, because of the parity prediction for the *ortho* module, depicted in Fig. 55 - Fig. 58. More precisely, for all implementations involving more than 1 parity bit associated to the 32 bits processed by *ortho*, as evident from equation (1), the parity of module's output can be directly predicted based solely input's parity. In consequence, the implementations using 2 and 4 bits of parity for the 32 bits processed by the *ortho* module, the final *verifier* unit checking the correctness of parity bits is not required.

Apart from this the higher the number of parity bits, the smaller the height of the XOR tree used inside the *verifier* module and thus the faster the parity verification. More precisely, when using a single bit of parity for a group of 32 bits, the *verifier* unit contains an XOR tree of height 5, whereas when a parity bit is associated with a byte, the XOR tree has a height of only 3 logic levels. The latency associated with the *verifier* modules justify also the faster performance of the 4 parity bit design compared to the 2 parity bit solution.

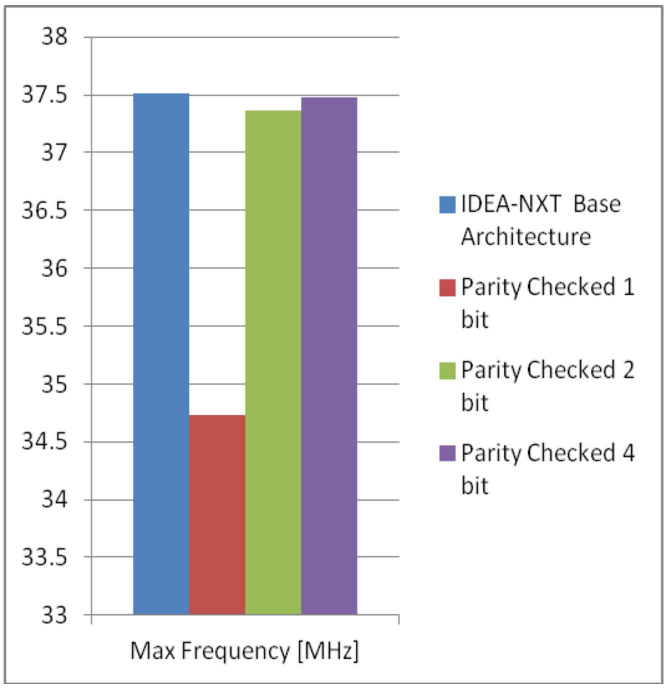


Fig 55 IDEA NXT parity-based architectures synthesis results for Max Frequency

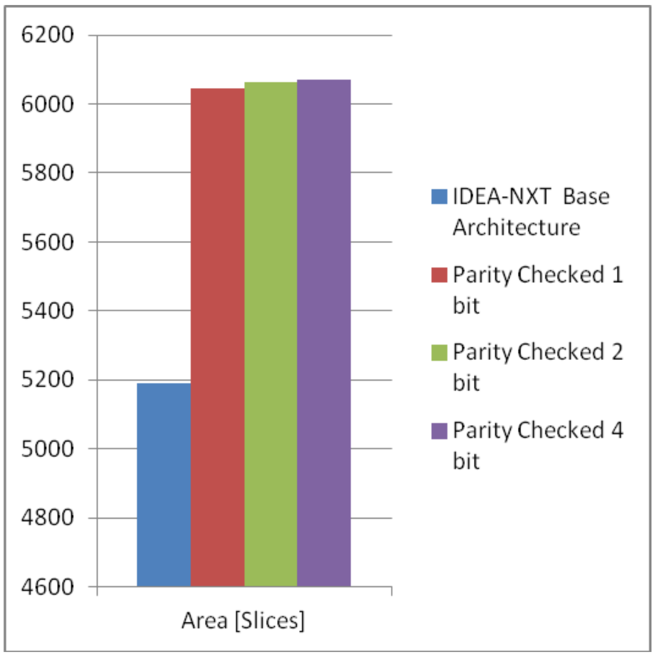


Fig 56 IDEA NXT parity-based architectures synthesis results for Area

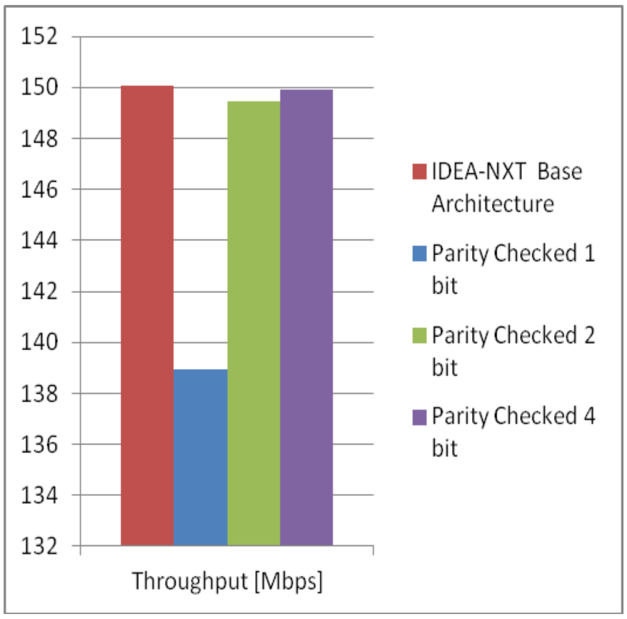


Fig 57 IDEA NXT parity-based architectures synthesis results in terms of Throughput

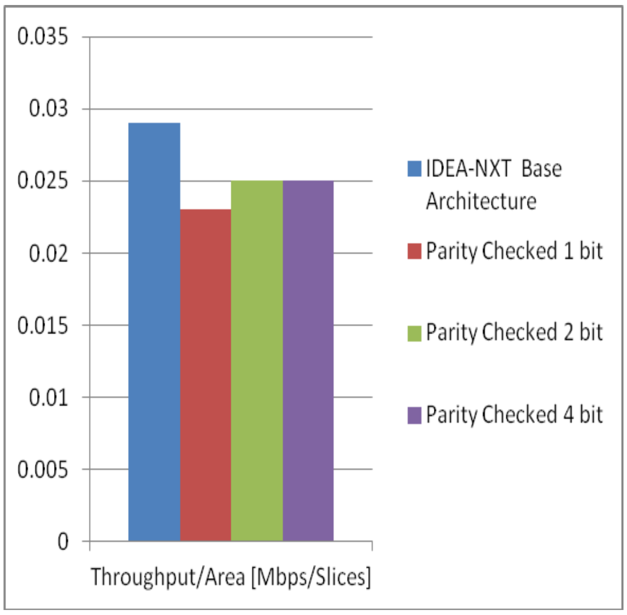


Fig 58 IDEA NXT parity-based architectures synthesis results in terms of Throughput/Area

As can be seen from Table VII, the effect of increasing the redundancy level over the area of the design is consistent. With respect to the combined metric Throughtput/Area, the 2 parity bit and 4 parity bit architectures have similar scores, also higher than the score for single parity bit design. This is the reason that IDEA NXT64 is better verified concurrently for errors by employing either 2 or 4 parity bits associated with each group of 4 bytes processed by the algorithm.

5.5 Error-Detection Rate for the Concurrent and Offline Testing Architectures built for IDEA NXT

Figures 57-61 show the error-detection rate for the BIST Error-Detection Schemes I have built for IDEA crypto-algorithm in the cases where 2, 10, 100, 250 and 500 faults have been injected in the architectures. It can be observed that in all cases the fault detection rate is above 99% when more than 200 thousands simulation of the algorithm have been performed.

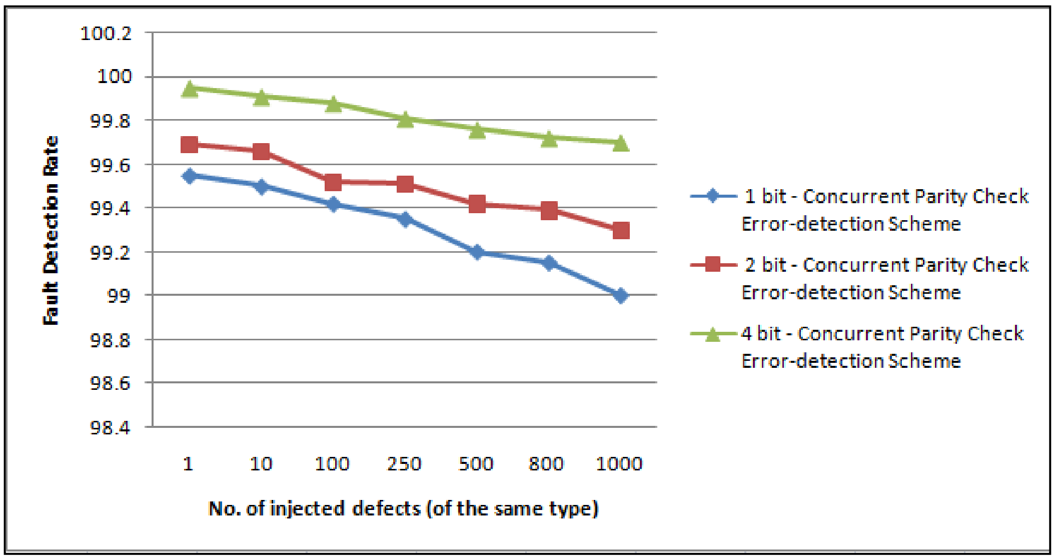


Fig 59 Detection Rate for stuck-at-0 defects of the same type injected into the parity-based on-line error-detection Architecture built for IDEA NXT

For the parity-based concurrent architectures I ran simulations for the cases where 1, 10, 100, 250, 500, 800 and 1000 stuck-at-0 defetcts were inserted inserted into the design and verified the error-detection rate after all algorithm rounds were run. The results are shown in fig. 59. The error-detection rate is best when 1 defect is injected into the concurrent architecture where bit of parity is associated with 4 bytes of the data processed by all units of the algorithm, reaching 100%. The case where 1000 defects of the same type are injected into the algorithm has the lowest detection rate in all 3 types of parity-based testing architectures, but it is still over 90% in all cases.

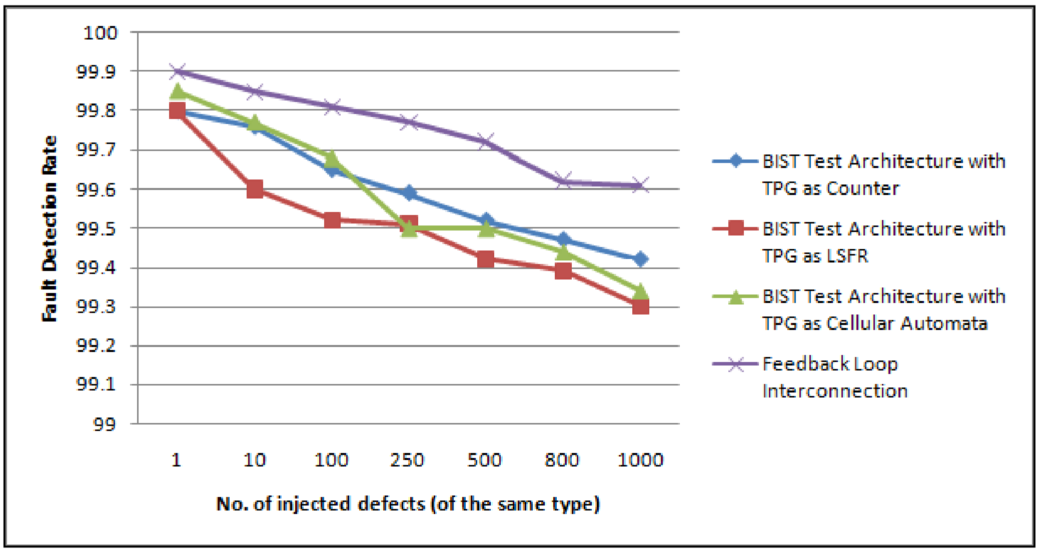


Fig 60 Detection Rate for stuck-at-0 defects of the same type injected into the parity-based on-line error-detection Architecture built for IDEA NXT

In fig. 60 it can be depicted the error-detection rate for the cases where 1, 10, 100, 250, 500, 800 and 1000 stuck-at-0 faults have been injected in the Feedback Loop Interconnection scheme and all 3 BIST-based error-detection schemes. The best results are obtained for the Loop Inteconnection rate, which reached 99.9% for the case where 1 defect is injected into the design, and the overall detection rate is above 99.3% which is a more than acceptable percent.

CONCLUSIONS

This research was concentrated on two main areas: cryptography and error-detection mechanisms.

The thesis starts with the problem of secure testing of cryptographic algorithms, focusing on hardware implementations of the new trends in cryptography. It gradually introduces the problem by first making an incursion in the cryptographic domain, the definitions of the basic taxonomies regarding dependability and safety of calculations and detailing crypto-algorithms theory and classification. It also introduces the next generation in encryption, the IDEA NXT algorithm, then it proceeds with original hardware implementations for it and the improvements which I made in the algorithm's structure in order to obtain better performances in terms of speed.

IDEA NXT was proven to be one of the most complex and secure crypto-algorithms at this moment [2] and so efforts to bring improvements to it are completely justified. Also, IDEA NXT raises the bar for designing encryption algorithms, providing a high level of algorithm individualization based on personal adjustment of the key and plain text lengths.

The particularities and mathematical structure of the IDEA NXT crypto-algorithm were presented also, analyzing the complexity of its non-linear operations and the role played by the LFSR in the key generator algorithm. This offered an opportunity for improvement, as a speed-up of the algorithm was build from here, improving the algorithm's overall performance. The encryption and decryption datapaths were design for a hardware implementation, as well as the key generation unit.

Once the algorithm is used in securing complex system and crypto-chips, the issue of assuring its continuous well-functioning appears. The next step of the research was to find ways to test the algorithm in its environment and find potential faults errors as soon as they appear, so that the cost of the repairs is small and the algorithm can be again functional in no time.

The second part of the thesis starts with an overview of the testing theory in general and VLSI testing in particular, in order to understand the concepts and the work performed in the field so far. It presents the main defects and fault models which can affect a device at every level of design, from gate to whole circuit level. The stuck-at model was detailed as representative for the lower-level defects, while bridging and shorts defects, as well as the fault models which derived from there were briefly discussed as alternatives for the cases not-covered by the stuck-at fault.

After consolidating the problematic of fault modeling, the thesis follows with introducing some fault tolerance concepts. A design must be trustable and dependable during its functioning or integration in a chip or higher-level system and so it must be continuously or periodically tested for errors. Hence a fully justified review of the design for Testability Architectures and Techniques was made, with a focus on scan design, boundary scan and test generation techniques. The challenges faced by the testing community are reminded and Logic BIST methodology is introduced as an appropriate solution for secure testing. The various implementation methods, including the LFSR-based approaches for test pattern generation and output response analysis which was later used in the custom error-detection architectures I built for the IDEA NXT hardware implementations, were presented along with a concurrent, parity-based online test architecture build for detecting the presence of faults in the IDEA NXT crypto-algorithm while the algorithm is running.

From the multiple offline BIST error-detection architectures I proposed, the LFSR-based solutions proved to be the more efficient in terms of coverage and area overhead.

In order to validate experimentally the custom testing architectures built for IDEA NXT I injected stuck-at faults into the designs using the technique of mutants (I built mutant architectures of the Verilog components which would simulate the behaviour of those components in the presence of faults).

The **personal, original contributions** of this research are highlighted in the following paragraphs:

- A speed improvement made to the pseudo-random key generator of the IDEA NXT encryption algorithm, achieved by modifying the way keys are processed when they reach the third layer of the Key Scheduler, denoted Diversification, which uses a stream of pseudo-random values produced by a 24-bit Linear Feedback Shift Register (LFSR). The proposal is to change the rules of the LFSR as to process six bytes per clock cycle instead of one byte per clock cycle as was the case in the original structure, so to not lose 6 clock cycles for the generation of a single round key, but to directly provide a round key in a single clock cycle. The solution for this is to directly shift the linear shift register with six positions at once. The experimental results performed on both my software and hardware implementations of IDEA NXT, showed a speedup of 10% when the modified version of the algorithm was used.
- The design of a parity-based testing architecture, that is completely independent from the algorithm itself, and which works for all versions of the algorithm, independent of the key and text length. It mainly consists of generating and processing a series of parity bits for both the Datapath and the Key Scheduler and checking their value at every step of the algorithm, by verifying the outputs of the Parity-predictor modules (built for complex operations where the output parity cannot be determined based on module's input parity) against a checker scheme – if the parity bit was incorrect, an error had been introduced at that stage. The proposed design was validated by implementing the error detection architecture in the Verilog hardware modeling language and synthesizing it for three different redundancy levels. The efficiency of the proposed solution was demonstrated by analyzing the obtained results in terms of area, frequency and throughput. The parity-prediction scheme is the first of this kind for IDEA NXT.
- The design of a series of non-concurrent test architectures for IDEA NXT. The proposed error-detection schemes are capable of verifying the integrity of a crypto-chip in an autonomous, non-concurrent manner. One of the testing architectures consists of stimulating the algorithm with test vectors and verifying the correctness of the output after a number of encryption or decryption iterations were run and the other one evaluates the response by compressing the results obtained after a predefined number of encryptions into a signature. The first solution is a particular Built-in Self-Test Design for Testability in which input vectors are generated at each round of the NXT algorithm and the output responses are validated through a feedback interconnection by the system in which the algorithm is running. The repeated execution of the algorithm is managed in conjunction with a counter for verifying the final signature. The second offline testing architecture evaluates the output of the encryption by comparing it to a so called "gold signature", which is a hardwired value obtained by simulations

of IDEA NXT. The output resulted after a run of the encryption algorithm of a single round is compacted, encompassing all the previous encryption outputs The test schemes I propose offer a good trade-off between the length of the test process and the storage requirements for the correct responses.

The thesis is based on two PhD reports submitted and presented in the Computer Science and Engineering Department of Computer and Automation Faculty, the Politehnica University of Timisoara:

- Andreea Bozesan, PhD Report I, Politehnica University of Timisoara, September 2014
- Andreea Bozesan, PhD Report II, Politehnica University of Timisoara, December 2014

As for the future research in the domain of crypto-algorithms and error-detection architectures I forsee the following possible directions:

- Extending the speed improvement brought to IDEA NXT as to find an alternative to the use of LFSRs and also improve security.
- Extending the simulation framework to support bridging and gate fault simulation models, not just the stuck-at.
- Building a general framework for a signature-based concurrent checking mechanism adaptable to encryption algorithms which are built on the same principles.
- Constructing a general framework for offline error detection which can easily be adapted for all symmetric algorithms.

REFERENCES

[1] Manoj Kumar, “A cryptographic study of some digital signature schemes”
[2] International Federation for Information Processing WG 10.4 on Dependable Computing and Fault Tolerance (established 1980, revised 1988)
[3] A. Menezes, P. van Oorschot, and S. Vanstone, “Handbook of Applied Cryptography”, CRC Press, 1996
[4] J-C.Laprie, “Dependability: Basic Concepts and Terminology”, 1992
[5] C. Cachin, J. Camenisch, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe. J.C. Laprie, J.C. Lebraud, D. Long, T. McCucheon, J. Muller, F. Petzold, B. Pfitzmann, D. Powell, B. “Malicious - and Accidental – Fault Tolerance in Internet Applications: Reference Model and Use Cases”, LAAS report no. 00280, MAFTIA, Project IST-1999-11583, p. 113, August 2000
[6] A. Avizienis, J-C. Laprie, B. Randell and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, IEEE Trans. On Dependable and Secure Computing, Vol.1, n.1, Jan.-March 2004
[7] Brian Selic, Staff, IMB, “Fault tolerance techniques for distributed systems”,
[8] Andrey Bogdanov, D. Khovratovich, and Christian Rechberger, “Biclique Cryptanalysis of the Full AES”, 2001
[9] Shain-Han Wu, “The Research on Masking Countermeasure Against Differential Power Analysis”, 2003
[10] S. Mangrad, “A Simple Power Analysis (SPA) Attack on Implementations of the AES Key Expansion”, in Proceedings of the International Conference on Information Security and Cryptography – ICISC 2002, p343-358, Springer - Verlag, 2002
[11] J. S. Coron, L. Goubin, “On Boolean and Arithmetic Masking Against Differential Power Analisys”, in Proceedings of Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000, p231-237, Springer - Verlag, 2000
[12] M. L. Akkar, L. Goubin, “A Generic Protection Against High-Order Differential Power Attack ”, in Proceedings of FSE 2003, Springer - Verlag, 2003
[13] P. Junod, S. Vaudenay, “FOX Specifications. Version 1.2”, Springer-Verlag. 2005 [14] F. Muller, “A new attack against Khazad”, in Advances in Cryptology - ASIACRYPT'03, volume 2894 of Lecture Notes in Computer Science, Pagina 347-358, Springer-Verlag, 2003
[15] N. Courtois, J. Pieprzyk, “Cryptanalysis of block ciphers with overdefined systems of equations”, in Advances in Cryptology - ASIACRYPT'02, volume 2501 of Lecture Notes in Computer Science, pages 267-287, Springer-Verlag, 2002
[16] IDEA NXT Next Generation Encryption, Mediacrypt, Switzerland, 2006
[17] Bozesan, Andreea; Opritoiu, Flavius; Vladutiu, Mircea –"Hardware implementation of the IDEA NXT crypto-algorithm," Design and Technology in Electronic Packaging (SIITME), 2013 IEEE 19th International Symposium for , vol., no., pp.35,38, 24-27 Oct. 2013
[18] W. Wolf, “FPGA Based Sistem Design”, Pretice Hall, 2004
[19] J. Blömer and J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES) " Lecture Notes in Computer Science, pp. 162-181: Springer, 2004.
[20] *Heinz Bonnenberg – „*Secure testing of VLSI cryptographic equipment”, PhD Thesis, 1993
[21] P. Junod, S. Vaudenay, "Perfect Diffusion Primitives for Block Ciphers," Selected Areas in Cryptography, Lecture Notes in Computer Science H. Handschuh and M. A. Hasan, eds., pp. 84-99: Springer Berlin Heidelberg, 2005.

[22] P. Junod, S. Vaudenay, "FOX Specifications version 1.2, 2005, p.5-40
[23] K. Chong Hee, and J. J. Quisquater, "Faults, Injection Methods, and Fault Attacks," *Design & Test of Computers, IEEE,* vol. 24, no. 6, pp. 544-545, 2007.
[24] M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Differential Fault Analysis Attack Resistant Architectures for the Advanced Encryption Standard," Smart Card Technologies and Applications, J.-J. Quisquater, P. Paradinas, Y. Deswarte et al., eds., pp. 177-192: Springer-Verlag, 2004.
[25] S. Burton Kaliski Jr., Matthew J. B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations", 1994C. Cachin, J. Camenisch, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe. J.C. Laprie, J.C. Lebraud, D. Long, T. McCucheon, J. Muller, F. Petzold, B. Pfitzmann, D.
[26] W. Meier , "On the Security of the IDEA block cipher, Advances in Cryptology", 1996
[27] J. S. Coron, L. Goubin, "On Boolean and Arithmetic Masking Against Differential Power Analisys", in Proceedings of Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000, p231-237, Springer - Verlag, 2000
[28] N. Courtois, J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations", in Advances in Cryptology - ASIACRYPT'02, volume 2501 of Lecture Notes in Computer Science, pages 267-287, Springer-Verlag, 2002J. Daemon, R. Govaerts, J. Vandervale, "Weak Keys of IDEA", Advances in Cryptology, CRYPTO 93 Proceedings, Lecture Notes in Computer Science Volume 773, 1994, pp 224-231
[29] J. Daemen, and V. Rijmen, The Design of Rijndael: Springer-Verlag New York, Inc., 2002.P. Kitsos, N. Sklavos, M.D. Galanis, O. Koufopavlou, "64-bit Block ciphers: hardware implementations and comparison analysis", VLSI Design Laboratory, Electrical and Computer Engineering Department, University of Patras, Greece, vol. 30, no. 8, 2004
[30] Bruce Schneier, "Applied Cryptography. Protocols, Algorithms and Source Code in C", J. Willey & Sons, New York, 1996
[31] Bozesan, A.; Opritoiu, F.; Vladutiu, M., "Speed Improvement for the IDEA NXT Crypto-Algorithm", AFCEA Europe 6th Student Symposium, 24 March 2014
[32] Rao, T.R.N., Fujiwara, E.:"Error-Control Coding for Computer Systems", Prentice-Hall International, 1989
[33] Steffen Tarnick, "Bounding error masking in linear output space compression schemes," Test Symposium, 1994., Proceedings of the Third Asian , vol., no., pp.27,32, 15-17 Nov 1994
[34] Sobeeh Almukhaizim and Yiorgos Makris, "Fault Tolerant Design of Random Logic based on a Parity Check Code", Electrical Engineering Department Yale University, 2001
[35] Opritoiu, F.; Vladutiu, M.; Udrescu, M.; Prodan, L., "Round-level concurrent error detection applied to Advanced Encryption Standard," Design and Diagnostics of Electronic Circuits & Systems, 2009. DDECS '09. 12th International Symposium on , vol., no., pp.270,275, 15-17 April 2009
[36] ***, "Security Requirements for Cryptographic Modules", Federal Information, Processing Standards Publication 140-2, December 2002
[37] F. Opritoiu, A. Bozesan, M. Vladutiu – "Pseudo-Random Self-Test Architecture for Advanced Encryption Standard", 19th International Symposium for Design and Technology in Electronic Packaging, IEEE, 2013
[38] A. Moradi, O. Mischke, and C. Paar, "One Attack to Rule Them All: Collision Timing Attack versus 42 AES ASIC Cores," *Computers, IEEE Transactions on,* vol. 62, no. 9, pp. 1786-1798, 2013.

[39] Mozaffari-Kermani, M.; Reyhani-Masoleh, A., "Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard," Computers, IEEE Transactions on , vol.59, no.5, pp.608,622, May 2010
[40] Ross J. Anderson – "Security Engineering - A Guide to Building Dependable Distributed Systems", 2001
[41] S. Burton Kaliski Jr., Matthew J. B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations", 1994
[42] C. Cachin, J. Camenisch, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe. J.C. Laprie, J.C. Lebraud, D. Long, T. McCucheon, J. Muller, F. Petzold, B. Pfitzmann, W. Meier , "On the Security of the IDEA block cipher, Advances in Cryptology", 1996
[43] S. Burton Kaliski Jr., Matthew J. B. Robshaw, "Linear Cryptanalysis Using Multiple Approximations", 1994
[44] C. Cachin, J. Camenisch, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe. J.C. Laprie, J.C. Lebraud, D. Long, T. McCucheon, J. Muller, F. Petzold, B. Pfitzmann, W. Meier , "On the Security of the IDEA block cipher, Advances in Cryptology", 1996
[45] N. Courtois, J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations", in Advances in Cryptology - ASIACRYPT'02, volume 2501 of Lecture Notes in Computer Science, pages 267-287, Springer-Verlag, 2002
[46] P. Kitsos, N. Sklavos, M.D. Galanis, O. Koufopavlou, "64-bit Block ciphers: hardware implementations and comparison analysis", VLSI Design Laboratory, Electrical and Computer Engineering Department, University of Patras, Greece, 2004
[47] Bozesan,Andreea; Opritoiu, Flavius; Vladutiu,Mircea - "Parity-based Concurrent Error-detection Architecture applied to the IDEA NXT crypto-algorithm", IEEE 6th International Workshop on Soft Computing Applications, June 2014
[48] Sachin Dhingra , "Comparison of LFSR and CA for BIST ", 2001
[49] A. Schubert, and W. Anheier, "On Random Pattern Testability of Cryptographic VLSI Cores," J. Electron. Test., vol. 16, no. 3, pp. 185-192, 2000.
[50] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *Computers, IEEE Transactions on,* vol. 52, no. 4, pp. 492-505, 2003.
[51] A. Pakstas, I. Schagaes, J. Zalewski , "Redundancy for Fault Tolerant Computing", 1999
[52] 11. L.-T. Wang, Y.-W. Chang and K.-T. Cheng, Electronic Design Automation: Synthesis, Verification, and Test: Morgan Kaufmann, 2009.
[53] 6. D. P. Siewiorek and R. S. Swarz, Reliable computer systems (3rd ed.): design and evaluation: A. K. Peters, Ltd., 1998.
[54] http://www.ece.unm.edu/~jimp/vlsi_test/slides/html/faults2.html
[55] Mei-Chen Hsueh, Timothy K.Tsai, Ravishankar K.Iyer, "Fault Injection Techniques and Tools", IEEE 1997
[56] Haissam Ziade, Rafic Ayoubi2, and Raoul Velazco, "A survey on Fault Injection techniques", The International Arab Journal of Information Technology, Vol. 1, No. 2, July 2004
[57] Opritoiu, Flavius Bozesan, Andreea; Vladutiu,Mircea - "Offline Error-Detection Strategies for the IDEA NXT Crypto-Algorithm", IEEE International Conference on System Theory, Control and Computing, October 2014\
[58] SIA, "The International Technology Roadmap for Semiconductors," http://public.itrs.net, 2004.
[59] F.-X. Standaert, L. Batina, E. D. Mulder, K. Lemke, S. Mangard, E. Oswald and G. Piret, „Electromagnetic Analysis and Fault Attacks: State of the Art," Technical Report, 2005.

[60] C. H. Kim and J.-J. Quisquater, "Faults, Injection Methods, and Fault Attacks," IEEE Design & Test, vol. 24, no. 6, pp. 544-545, 2007.
[61] D. P. Siewiorek and R. S. Swarz, „Reliable computer systems (3rd ed.): design and evaluation", A. K. Peters, Ltd., 1998.
[62] L.-T. Wang, C.-W. Wu and X. Wen, „VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)," Morgan Kaufmann Publishers, 2006.
[63] M. Bushnell and V. Agrawal, „Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits", Springer-Verlag, 2000.
[64] M. Abramovici, M. A. Breuer and A. D. Friedman, „Digital Systems Testing and Testable Design", Wiley-IEEE Press, 1994.
[65] F. Opritoiu and M. Vladutiu, "Research Concerning Configuration of Cryptochips with Testability Facilities," Diploma Project, 2007.
[66] N. K. Jha and S. Gupta, „Testing of Digital Systems" Cambridge University Press, 2002.
[67] L.-T. Wang, Y.-W. Chang and K.-T. Cheng, „Electronic Design Automation: Synthesis, Verification, and Test", Morgan Kaufmann, 2009.
[68] S. Matakias, Y. Tsiatouhas, T. Haniotakis and A. Arapoyanni, "A Current Mode, Parallel, Two-Rail Code Checker," IEEE Transactions on Computers, vol. 57, no. 8, pp. 1032-1045, 2008.
[69] N. Beniwitz, D. Calhoun, G. Alderson, J. Bauer and C.Joeckel, "An advanced fault isolation system for digital logic", IEEE Trans. Comput, 24(5), 1975
[70] M. Omana, D. Rossi and C. Metra, "Low Cost and High Speed Embedded Two-Rail Code Checker," IEEE Transactions on Computers, vol. 54, no. 2, pp. 153-164, 2005.
[78] W. J. Dally and J. W. Poulton, Digital systems engineering: Cambridge University Press, 1998.
[79] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," IEEE Transactions on Computers, vol. 52, no. 4, pp. 492-505, 2003.
[80] C. Wei-Yu, S. K. Gupta and M. A. Breuer, "Analytical models for crosstalk excitation and propagation in VLSI circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 10, pp. 1117-1131, 2002.
[81] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash and M. Hachinger, "A case study of ir-drop in structured at-speed testing," in Test Conference, 2003. Proceedings. ITC 2003. International, 2003, pp. 1098-1104.
[82] L.-T. Wang, C. Stroud and N. Touba, System-on-Chip Test Architectures: Morgan Kaufmann Publishing, 2007.
[83] S. L. Hurst, VLSI Testing: Digital and Mixed Analogue/Digital Techniques: The Institution of Engineering and Technology, 1999
[84] E. A. Amerasekera and F. N. Najm, Failure Mechanisms in Semiconductor Devices: Wiley Publishing, 1997.
[85] K. P. Parker, The Boundary-Scan Handbook: Springer-Verlag, 2003.
[86] C. E. Stroud, A Designer's Guide to Built-in Self-Test: Springer-Verlag, 2002.
[87] K. Stanley, "High-Accuracy Flush-and-Scan Software Diagnostic," IEEE Design & Test, vol. 18, no. 6, pp. 56-62, 2001.
[88] F. Opritoiu and M. Vladutiu, "Cryptochip implementations with Built-In Self Test features applied to AES standard," in The Claude Shannon Institute Workshop on Coding & Cryptography, Cork, Ireland, May 19–20, 2008.
[89] F. Opritoiu, M. Vladutiu, L. Prodan and M. Udrescu, "Built-In Self Test Applicability for the Non-Linear Operations of Advanced Encryption Standard," The 5th International Symposium on Applied Computational Intelligence and Informatics

pp. 307-312, May 28–29, 2009.
[90] D.Schnurmann, E. Lindbloom and R.G. carpenter, "The weighted random test pattern-generator", IEEE Transacyions on Computer, 24(7), 675-700, 1975.
[91] J. E. Gentle, Random Number Generation and Monte Carlo Methods, 2nd Ed: Springer, 2003.
[92] Wolfram, "Statistical mechanics of cellular automata", Rev.Mod.Phys., 55(3), 601-644, 1983
[93] T. Klove, Codes for Error Detection: World Scientific Publishing Company, 2007.
[94] S. Chang, "Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields," IEEE Transactions on Computers, vol. 58, pp. 1221-1237, 2009.
[94] K. K. Saluja and C.-F. See, "An Efficient Signature Computation Method," IEEE Design & Test, vol. 9, no. 4, pp. 22-26, 1992.
[95] R. Rajsuman, "System-on-a-Chip: Design and Test", Artech House Publishers 2000.
[96] Laung-Terng Wang, Cheng-wen Wu, K. Wen – "VLSI Test Principles and Architectures: Design for Testability", 2006
[97] M. Abramovici, M.A. Brauer and A.D. Friedman, "Digital Signal Testing and Testable Design", IEEE Press, NJ1994
[98] E.J. McClusky, "Logic Design Principles with emphasis on testable semi-custom circuits", Prentice Hall, Englewood Cliffs, 1986
[99] M. Gössel, V. Ocheretny, E. Sogomonyan and D. Marienfeld, "New Methods of Concurrent Checking", Springer-Verlag, 2008.
[100] I. Voyiatzis, A. Paschalis, D. Gizopoulos, C. Halatsis, F. S. Makri and M. Hatzimihail, "An Input Vector Monitoring Concurrent BIST Architecture Based on a Precomputed Test Set," IEEE Transactions on Computers, vol. 57, no. 8, pp. 1012-1022, 2008.
[101] Elguibaly, F. , El-Kharashi, M.W., "Multiple-input signature registers: an improved design"
[102] M. Nicolaidis and Y. Zorian, "On-Line Testing for VLSI - A Compendium of Approaches," Journal of Electronic Testing, vol. 12, no. 1-2, pp. 7-20, 1998.
[103] S. Mitra and E. J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?," in Proceedings of the 2000 IEEE International Test Conference, 2000, pp. 985.
[104] D. K. Pradhan, Fault-tolerant computer system design: Prentice-Hall, 1996.
[105] E. J. McCluskey and C.-W. Tseng, "Stuck-fault tests vs. actual defects," in Proceedings IEEE International Test Conference, October, 2000, pp. 336-343.
[106] W. Jue and E. M. Rudnick, "A Diagnostic Fault Simulator for Fast Diagnosis of Bridge Faults," in Proceedings of the 12th International Conference on VLSI Design - 'VLSI for the Information Appliance', 1999, pp. 498.
[107] D. Sokolov, J. Murphy, A. Bystrov and A. Yakovlev, "Design and Analysis of Dual-Rail Circuits for Security Applications," IEEE Transaction on Computers, vol. 54, no. 4, pp. 449-460, 2005.
[108] I. Koren and C. M. Krishna," Fault Tolerant Systems" Morgan Kaufmann Publishers, 2007.
[109] F. Opritoiu, M. Vladutiu, M. Udrescu and L. Prodan, "Round-Level Concurrent Error Detection Applied to Advanced Encryption Standard," The 12th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 270-275, April 15-17, 2009.
[110] K. Wu and R. Karri, "Algorithm Level RE-computing with Shifted Operands- A Register Transfer Level Concurrent Error Detection Technique," in Proceedings of the

2000 IEEE International Test Conference, 2000, pp. 971.
[111] Alfredo Benso, Maurizio Rebaudengo, Matteo Sonza Reorda, "Fault Injection for Embedded Microprocessor-based Systems", Journal of Computer Science, 1999
[112]http://www.cs.uiuc.edu/class/fa05/cs598yyz/slides/presentation/radu_fault_injection.pdf
[113] http://www.ece.unm.edu/~jimp/310/slides/8086_memory2.html
[114] A. Pakstast, I. Schagaev, J. Zalewski, "Redundancy Classification For Fault Tolerant Computer Design", IEEE Explore 2002
[115] Martin Shooman, "Reliability of Computer Systems and Networks - Fault Tolerance, Analysis and Design", Wiley 2002
[116]http://soc.cs.nchu.edu.tw/pllai/NCUT/VLSI%20Testing/Chapter%204%20DFT.pdf
[117] Algorithms, Key Sizes and Parameters Report, 2013 Recommendations, October 2013
[118] http://www.darkreading.com/attacks-breaches/criminals-control-cash-out-banks-atm-machines/d/d-id/1141320?
[119] J. Arlat, Validation de la Suˆrete´ de Fonctionnement par Injection de Fautes, Me´thode—Mise en Oeuvre—Application, The`se pre´sente´e a` L'Institut National Polytechnique de T oulouse, Rapport de Recherche LAAS No. 90-399, De´cembre 1990.
[120] D. Gil, J. Gracia*, J.C. Baraza, P.J. Gil, "Study, comparison and application of different VHDL-based fault injection techniques for the experimental validation of a fault-tolerant system", Microelectronics Journal 34, 2003, pag. 41–51
[121] J.C. Baraza, J. Gracia, D. Gil and P.J. Gil, "Improvement of Fault Injection Techniques Based on VHDL Code Modification", IEEE, 2005
[122] A.M. Amendola, A. Benso, F. Corno, L. Impagliazzo, P. Marmo, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, Fault behaviour observation of a microprocessor system through a VHDL simulationbased fault injection experiment, Proceedings of EuroDAC96, 1996.
[123] J.D. Hietala, "Hardware Versus Software: A Usability Comparison of Software-Based Encryption with Seagate Secure Hardware-Based Encryption", SANS Institute InfoSec Reading Room, 2007

List of Publications

Conference Proceedings

- Andreea BOZESAN, Flavius OPRITOIU, Mircea VLADUTIU.”Parity-based Concurrent Error-detection Architecture applied to the IDEA NXT Crypto-algorithm”, 2014 IEEE 6th International Workshop on Soft Computing Applications (SOFA)
- Andreea BOZESAN, Flavius OPRITOIU, Mircea VLADUTIU. “Offline Self-Testing Architectures for the IDEA NXT Crypto-Algorithm”, 2014 18th International Conference in System Theory, Control and Computing (ICSTCC), pp. 37-42 (ISI indexed)
- Andreea BOZESAN, Flavius OPRITOIU, Mircea VLADUTIU, Andrei MARGHESCU, “Speed Improvement in the Key Scheduler of the IDEA NXT Crypto-Algorithm “, 2014 International Conference Automation Quality and Testing Robotics (AQTR)
- Andrei MARGHESCU, Paul SVASTA, Andreea BOZESAN, “Variable Probability Pseudo Random Number Generator”, 2014 International Conference Automation Quality and Testing Robotics (AQTR)
- Andreea BOZESAN, Flavius OPRITOIU, Mircea VLADUTIU, „Hardware Implementation of the IDEA NXT Crypto-Algorithm”, 2013 IEEE 19th International Symposium for Design and Technology in Electronic Packaging (SIITME), pp. 35-38 (ISI indexed)
- Flavius OPRITOIU, Andreea BOZESAN, Mircea VLADUTIU. “Pseudo Random Self-Test Architecture for AES”, 2013 IEEE 19th International Symposium for Design and Technology in Electronic Packaging (SIITME), pp. 35-38 (ISI inedxed)

Distinctions and nominalisations

- Won the EXCELLENT POSTER AWARD FOR YOUNG SCIENTISTS at the SIITME Conference in 2013 for the papar „Hardware Implementation of the IDEA NXT Crypto-Algorithm”
- Nominalization for BEST PAPER AWARD FOR YOUNG SCIENTISTS at the ICSTCC Conference in 2014 for the paper ”Offline Self-Testing Architectures for the IDEA NXT Crypto-Algorithm”