POLITEHNICA UNIVERSITY OF TIMISOARA
AUTOMATION AND COMPUTER SCIENCE FACULTY

PHD THESIS

Reliable Implementations for Cryptographic Systems with Testability Facilities

Flavius Opriţoiu
Politehnica University of Timişoara – UPT
Romania

Thesis Supervisor: Prof. dr. ing. Mircea Vlăduţiu (UPT Timişoara)
Thesis Committee: Prof. dr. ing. Mircea Petrescu (UPB Bucureşti)
Prof. dr. ing. Daniela Popescu (UO Oradea)
Prof. dr. ing. Liviu Miclea (UTC Cluj)

ABSTRACT

Integrated circuits with security functions gained an ever increasing importance in human experience. Their main utility is defined in the context of assuring protection for sensitive data. Among the many applications of the cryptographic methods, one can enumerate: the secure communication in computer networks, pay-per-view television, banking cards, electronic payments and biometric passports, to name only a few.

Both the actuality and the appropriateness of the thesis research topics can be justified by the interest manifested by the european union toward advancing the research on the security domain. It can be recollected the continuous support for sustaining the collaborative research between academics and industrial segments: the Stork project followed by Ecrypt and later on by the Ecrypt II program settled the basis for european cryptographic research, promoting these projects to the level of european-funded programs.

The actuality of the VLSI hardware testing in general and of cryptochip testing in particular is supported by the growing interest manifested toward testability by the Semiconductor Industry Association, in its periodic "International Technology Roadmap for Semiconductors" reports.

The efficiency of the test facilities integrated within a design directly influences the production costs through the defect rate. Knowing that it is more costly to replace a defective component in a later stage of its integration into the final system, than to discard it after production, the incorporation of test mechanism in a design, capable of verifying the device's integrity both after fabrication and during its operational cycle, became mandatory.

The hardware testing, in the context of cryptographic devices, poses new challenges. On one hand, the testing facilities are essential for any integrated device while on the other hand they are absolutely necessary for a system vulnerable to attacks. The continuous refinement of the cryptanalytic methods led to the development of a wide range of cryptographic-systems attack strategies: from the passive mechanisms, relying on the analysis of the information inherently leaked by semiconductor devices (such as thermal radiation, electromagnetic radiation) to the invasive methods, aiming to modify the internal state of the circuit. In this context, the integration of self-test facilities represents a top priority for any cryptographic design. However, the testing mechanism chosen to protect the device can be inadequate, to the point that it can represent the very starting point of a successful attack.

The apparent contradiction between the need for integrating testability features and their potential vulnerability can be easily surpassed by taking into consideration the self-test strategies. By conferring the test process complete independence from user and by reducing the volume of information transmitted to the exterior during the test process, these solutions significantly improve the system's security.

The clear conclusion of the previous comments recommends the hardware test engineering as a domain with significant challenges and opportunities for research.

The issue of fault modeling is introduced, by presenting the fault models hierarchically, in accordance with the integrated device's hardware description levels. The fault models are gradually differentiated starting from those at the algorithm level, continuing to the models specific to the Register Transfer Level, followed by those defined at the logic level of circuit description and finalizing with the fault models particular for the transistor and the semiconductor layer. The presentation also investigates the extent to which transistor and physical faults can be mapped into defects at the logical level. The extent to which the low-level defects are covered by the test processes generated for the stuck-at fault model is investigated using SPICE simulations at the CMOS transistor level. The VLSI specific bridging fault model and delay fault model are also detailed, together with a discussion of their detectability at the gate level.

The thesis introduces the domain of hardware testing, starting with the economical premises associated with the test. The self-test strategies, practical for strengthening the security of the test process and for assuring the device's integrity are presented, starting with the off-line methods. The Built In Self Test paradigm is introduced together with the implementation details characteristic to a conventional off-line configuration. The convenient solutions, relying on Linear Feedback Shift Registers, for implementing the test vector generation unit and test response compression module are also discussed. It is also detailed the issue of characteristic polynomial selection for the two units in order to reduce the aliasing probability. The Built In Logic Block Observer strategy is described as an algorithmic attempt to transform a typical VLSI design into a structure integrating Built In Self Test features. The following topic presented in the thesis relates to on-line testing: the conventional concurrent testing mechanisms, such as those relying on hardware duplication, error detecting codes or time redundancy are analyzed following the model of the most authoritative references in the literature. The work also describes effective concurrent solutions for detecting the faults that cannot be identified using logic-level fault model, such as those relying on parametric monitoring of the implementation.

The cryptographic algorithm domain is also presented in the thesis, starting with a brief description of its development together with the incentives for its adaptation and further refinement in order to satisfy the current security requirements. The presentation is focused on symmetric key algorithms, and in particular on the Advanced Encryption Standard. Besides describing the algorithm, the exposition investigates also the solutions offered in the literature for accelerating the encryption and the decryption processes. The culmination of this section is represented by the high speed AES dual design, capable of executing both the encryption and the decryption algorithm, whilst reducing the design dimensions through hardware resources sharing between the 2 operations. The main differences between the proposed solution and the ones found in the literature are also presented.

The thesis details the on-line and off-line test solutions proposed by the author, which, cumulatively protect all elements of the AES algorithm against defects in general and invasive attacks in particular. The presentation contains a section dedicated to presenting the testability solutions applicable to AES, as identified in the literature. This section is followed by the detailed presentation of two on-line test mechanisms, one for protecting the AES round transformations and another one for detecting errors in the AES inversion unit, as well as a BIST architecture for off-line testing, as an alternative solution for protecting the non-linear operations of the AES.

The first test solution belongs to the concurrent error detection mechanisms. It facilitates identification of the errors affecting any AES round transformation using parity-based error detection techniques. The proposed architecture was obtained by refining a basic structure, constructed from the on-line test solutions found in the literature. The main disadvantage of the basic structure relates to the irregular parity prediction structure introduced by the ShiftRows operation. This obstacle is surpassed in the proposed architecture by computing the parity bytes row-wise with respect to AES's state matrix. The mathematical details for predicting the parity bytes for each AES operation are also described in the thesis. One advantage of the proposed solution relates to its non-invasive structure, being applicable regardless of the implementation details for the AES round. Moreover the hardware complexity for predicting the parity for the key generation unit is significantly reduced. The proposed architecture is compared with the basic design, while the experimental results reveal its preeminence over the basic one with respect to design's dimensions, power consumption and test process latency.

The second test solution detects the errors affecting the 3 non-linear operations of the AES using non-concurrent self-test mechanisms. A BIST architecture is constructed, using LFSR units in order to detect every single stuck-at error. The selection of the characteristic polynomials for the LFSR structures used for test vector set generation and test response compaction was guided by simulations. This section investigates also the adaptation of the presented technique into an off-line test mechanism relying on concurrent monitoring of the protected unit's inputs. The experimental results denote a multiple stuck-at fault detection rate higher than 99.77% for the first BIST architecture.

The last test design introduces an on-line test strategy for detecting intermittent faults affecting the AES inversion unit. The test process rely on a convenient mathematical property of the multiplicative inversion in finite fields, detailed in the thesis, based on which, the sum of any element with its inverse can be only one out of 128 possible results. This way, the error verification consists in performing the addition in the finite field between inversion's input and output, followed by the verification of the fact that the obtained vector is one of the 128 correct configurations. The experimental results confirm the applicability of the proposed method for detecting single and double intermittent faults. Moreover, the adaptation of the presented method into a non-concurrent architecture, allows to significantly reducing the test process latency to only 4 cycles, as opposed to the conventional BIST architectures requiring at least 152 cycles and at most 255 rounds, as documented in the previous test solution.

Because of the aliasing probability, the on-line architecture is not suited for detecting multiple intermittent faults. To overcome this problem, code redundancy techniques were employed, at the verifier level, by associating the sum between inversion's input and output to a signature computed from the input configuration. For the analyzed solution, a signature computed by spatially compacting the inversion's input into 4 bits was used. The newly obtained architecture is capable of detecting, with a probability higher than 93%, up to 100 intermittent errors. In order to evaluate the efficiency of the proposed solutions, they were analyzed with respect to the hardware duplication mechanism, compared to which, the propose designs have a significantly lower hardware overhead, while still providing a detection rate higher than 93%.

The thesis concludes with a brief presentation of the results achieved during the doctoral research program, marking the contributions as well as the future research directions.

REZUMAT

Circuitele integrate cu funcţie de securitate joacă un rol cu o importanţă crescândă în viaţa cotidiană. În principal, utilitatea acestora se defineşte în contextul asigurării securităţii informaţiilor sensibile. Printre multele aplicaţii ale metodelor criptografice se pot aminti: comunicarea sigură în reţele de calculatoare, distribuţia media în cadrul televiziunilor digitale în sistem pay-per-view, carduri bancare, plaţi electronice precum şi paşaport biometric.

Atât actualitatea cât mai ales oportunitatea temei de cercetare alese pot fi justificate aducând în discuţie interesul manifestat de comunitatea europeană asupra domeniului securităţii. În acest sens, amintim demersurile continue de finanţare a cercetării colaborative între instituţiile academice şi segmentul industrial. În acest sens, proiectul Stork, urmat de Ecrypt şi de actualul Ecrypt II, fundamentează cercetarea europeană în domeniul criptografic, ridicând-o la nivel de activitate finanţată prin proiecte cadru de cercetare.

Actualitatea domeniului testării in tehnologie VLSI în general şi a cryptocipurilor în particular, este justificată de interesul acordat ingineriei testării, de către Asociaţia Industriilor Semiconductoare în rapoartele întocmite periodic de către acest for internaţional.

Eficienţa facilităţilor de testarea integrate într-un dispozitiv influenţează direct costul de producţie prin intermediul ratei de defectare. Cunoscând că înlocuirea unei componente defecte este cu atât mai puţin costisitoare cu cât este mai rapid identificată în procesul de asamblare a sistemului final, este pe deplin justificată integrarea unor mecanisme de testare în interiorul designului, care să permită verificarea integrităţii dispozitivului atat după fabricaţie, cât şi pe durata exploatării lui.

Problematica testării în contextul dispozitivelor criptografice capăta valenţe noi. Pe de o parte facilităţile de testarea sunt esenţiale pentru orice dispozitiv integrat, iar pe de altă parte, sunt absolut esenţiale pentru un sistem vulnerabil în fata atacurilor. Rafinarea metodele criptanalitice a dus la construirea unor strategii de atac a sistemelor criptografice dintre cele mai variate: de la metode pasive, bazate pe analiza informaţiilor oferite inerent de către dispozitivele integrate (radiaţie termică, radiaţie electromagnetică) până la soluţii invazive, ţintind modificarea stării interne a circuitului. În acest context, integrarea facilităţilor de verificare a integrităţii modulelor criptografice devine o prioritate a oricărui design criptografic. Pot fi alese însă soluţii de testare necorespunzătoare, care să constitue tocmai punctul de pornire al unor atacuri reuşite.

Aparenta contradicţie între necesitatea integrării facilităţilor de testare şi vulnerabilitatea pe care o pot introduce, este uşor depăşită luând în considerare soluţiile de autotestare. Conferind autonomie procesului de testare şi reducând volumul de informaţii transmise spre exterior pe durata testului, aceste soluţii îmbunătăţesc securitatea implementării.

Concluzia evidentă a comentariilor anterioare recomanda ingineria testării hardware ca fiind o disciplină cu oportunităţi şi provocări actuale, semnificative, în contextul cercetării ştiinţifice.

În teză, este introdusă tematica modelelor de defecte, prezentate ierarhic în concordanţă cu nivelele de descriere hardware ale circuitelor digitale. Modelele de defecte sunt diferenţiate gradual începând cu cele de la nivelul algoritmilor, continuând cu cele specifice nivelului de descriere Register Transfer Level, prezentând apoi modelele caracteristice nivelului de descriere logică al circuitelor şi finalizând cu mai specializatele modele de defecte la nivelul tranzistorilor şi al substratului semiconductor. Prezentarea investighează de asemenea, posibilitatea de mapare a defectelor la nivelul tranzistorilor şi al substratului fizic în termenii defectelor logice. Ipotezele de acoperire a defectelor de nivel jos prin tehnici de testare specific defectelor stuck-at sunt validate prin simulări la nivelul transistorului în tehnologie CMOS, utilizând mediul de simulare SPICE. Sunt detaliate şi modelele de defect de tip bridging şi cele de tip întârziere, specifice tehnologiei VLSI, oferind o discuţie a metodelor de detecţie a acestora la nivelul logic.

Lucrarea abordează în continuare problematica ingineriei testării hardware, pornind de la premisele de natură economică asociate procesului de testare. Strategiile de testare autonomă, utile în asigurarea integrităţii sistemelor criptografice, sunt detaliate, începând cu metodele de testare off-line, non-concurentă. Paradigma Built-In Self-Test este introdusă, împreună cu detaliile de implementare ale unei soluţii off-line caracteristice, aşa cum sunt prezentate în literatură şi cum sunt implementate practic de către fabricanţii de circuite. Sunt discutate soluţiile convenabile de implementare a unităţilor de generare a vectorilor de test precum şi a modulelor de compactare a răspunsurilor, bazate pe elemente Linear Feedback Shift Registers, împreună cu problemele de selecţie a configuraţiei acestora pentru reducerea probabilităţii de aliasing. Strategia Built In Logic Block Observer este descrisă de asemenea, reprezentând o soluţie algoritmică de transformare a unui design VLSI într-o structură care încorporează facilităţi Built-In Self-Test. Lucrarea analizează şi strategiile de testare concurenta: mecanismele convenţionale de testare on-line, cum sunt cele bazate pe duplicarea hardware, a utilizării codurilor detectoare de erori şi a metodelor redundantei de timp sunt detaliate, urmărind modelul referinţelor autoritative din literatură. Sunt prezentate soluţiile concurente pentru detecţia defectele care nu sunt detectate prin modelele de la nivelul logic sau cel al tranzistorului, cum sunt cele bazate pe monitorizarea parametrilor fizici ai implementării.

Domeniul algoritmilor de criptare este de asemenea investigat în teză, fiind oferind un scurt istoric al dezvoltării acestora precum şi modalităţile curente de utilizare şi adaptare a lor la nevoile de securitate curente. Accentul este pus pe algoritmi de criptare simetrică, considerând algoritmul Advanced Encryption Standard. Pe lângă prezentarea propriu-zisă a algoritmului, teza investighează soluţiile oferite în literatură, pentru accelerarea procesului de criptare respectiv de decriptare. Culminarea observaţiilor acumulate prin studiul literaturii de specialitate o reprezintă arhitectura AES duala, capabilă să execute atât procesul de criptare cât şi pe cel de decriptare, a cărei sintetiză urmăreşte reducerea dimensiunii designului prin partajarea resurselor hardware comune celor 2 procese, în condiţiile obţinerii unei viteze de operare ridicate. Sunt detaliate diferenţele notabile între structura propusă şi soluţiile întâlnite în literatură.

Lucrarea prezintă soluţiile de testare, atât on-line cât şi off-line, propuse de autor, soluţii care cumulativ protejează toate elementele constitutive ale algoritmului AES împotriva defectelor în general şi a atacurilor invazive în particular. Este inclusă o secţiune destinată inventarierii soluţiilor de testare aplicabile algoritmului AES, aşa cum au fost întâlnite în literatură, şi continuă cu prezentarea a doua mecanisme de testare on-line, unul destinat operaţiilor rundei AES, iar altul *operaţiei de inversie în câmpul Galois, şi a unei arhitecturi BIST de testare off-line, ca o soluţie alternativă de protejare a operatiilor nelinieare ale algoritmului.*

Prima soluţie de testare aparţine clasei de mecanisme de detecţie concurentă a erorilor. Ea permite identificarea erorilor care afectează operaţiile rundei AES, bazându-se pe mecanisme de detecţie a erorilor prin predicţia parităţii. Arhitectura propusă a fost construită pornind de la o structură de bază, proiectată pe baza soluţiilor de testare on-line propuse în literatură. Dezavantajul arhitecturii de bază constă în neuniformitatea predicţiei parităţii pentru transformarea ShiftRows, impediment înlăturat în structura propusă de autor, care calculează octeţii de paritate transversal, în matricea de stare. Lucrarea prezintă detaliile matematice de predicţie a parităţii pentru transformările AES. Se remarcă caracterul non-intruziv al soluţiei construite, permiţându-i să fie aplicată independent de modalitatea de implementare a căii de date AES. În plus, pentru noua structură, complexitatea predicţie parităţii în cazul modulului de generare a cheilor de rundă este semnificativ redusă. Structura realizată este comparată în raport cu arhitectura de bază, rezultatele evidenţiind superioritatea ei privind dimensiunile designului, puterea consumată şi latenţa procesului de test.

A doua soluţie de testare urmăreste detecţia erorilor ce pot afecta cele 3 operaţii neliniare ale AES, prin metode de autotestare non-concurentă. Este construită o arhitectură BIST utilizând elemente LFSR care să asigure detecţia oricărui defect stuck-at singular. Alegerea polinoamelor caracteristice pentru structurile LFSR de generare a vectorilor de test şi respectiv, de compactare a răspunsurilor, a fost ghidată de simulări. În cadrul aceleaşi soluţii este analizată adaptare tehnicii prezentate prin monitorizarea continuă a intrărilor unităţii protejate, în vederea identificării vectorilor relevanţi procesului de testare. Experimentele indică o rată de detecţie a defectelor multiple, mai mare de 99.77% pentru prima soluţie BIST propusă.

Ultima arhitectură de test introduce o soluţie on-line de detecţie a defectelor intermitente în unitatea de inversie AES. Testarea se bazează pe o proprietate matematică convenabilă a operaţiei, demonstrată în lucrare, pe baza căreia suma între un element al câmpului şi inversul său poate fi unul din 128 de posibile rezultate. În acest sens modulul de verificare a integrităţii inversiei efectuează suma în câmpul finit între intrarea şi ieşirea modulului verificând dacă rezultatul este unul din cele 128 de configuraţii corecte. Rezultatele experimentale indică aplicabilitatea acestei metodei în detecţia defectelor intermitente singulare sau duble. Adaptarea mecanismului prezentat într-o arhitectură de testare non-concurentă, permite reducerea latenţei testului la numai 4 cicluri, spre deosebire de soluţiile BIST convenţionale, necesitând între 152 şi 255 de cicluri, după cum documentează anterioara soluţie de testare.

Datorită efectului aliasingului, metoda de test on-line nu este potrivită pentru detecţia defectelor intermitente multiple. Soluţia la această problemă se rezuma la includerea redundantei de cod, asociind suma dintre intrarea şi ieşirea unităţii de inversie cu o semnătură calculata pentru configuraţia de intrare. Semnătura, în soluţia analizată, este obţinută prin compactare spaţială pe 4 biti a intrarii. Noua arhitectură obţinută, detectează cu probabilitate mai mare de 93% defecte intermitente cu multiplicitate de până la 100. Pentru evaluarea eficienţei structurilor construite a fost considerată tehnica duplicării hardware, faţă de care designurile propuse au o complexitate semnificativ redusă, oferind în acelaşi timp o rată de detecţie mai mare de 93%.

În încheierea lucrarii sunt prezentate concluziile activităţii de cercetare si sunt marcate punctual contribuţiile acesteia împreuna cu posibilele direcţii de continuare a cercetării.

ACKNOWLEDGEMENTS

The work presented in this dissertation represents the culmination of a concerted effort upheld by its author and augmented by the support he received from family, teachers and friends. Since it would be impossible to express my gratefulness individually, to all the people who endorsed, encouraged, sustained or braced the accomplishment of this thesis, I feel compelled to make a final appeal to their understanding cordiality. I would like however to express my gratitude to those who crafted through their unabated efforts my pursuit in completing this work.

This thesis would not have been possible without the help, support and patience of my supervisor, and mentor, Professor Mircea Vladutiu, not to mention his advices and unsurpassed knowledge of computer hardware design and test engineering. His ability to help me concentrate on the research topics whilst allowing me the room to work in my own way constituted the ideal environment for developing the work presented in this thesis.

I would like to acknowledge the financial, academic and technical support of the Politehnica University of Timisoara, and its staff. In particular I would like to thank the rector of our university, Professor Nicolae Robu for the significant support provided in order to attend key conferences.

I owe thanks equally to my colleagues at the ACSA laboratory: Lucian Prodan and Mihai Udrescu. Their proficiency in the field of computer hardware design, as well as their invaluable assistance in preparing conference articles and presentations were essential for my work.

Last, but not least, I would like to thank my family, for their continued effort to provide me with the best in life. Needless to say, the caring support they constantly offered helped me surmount the grey moments throughout my studies. For their unequivocal support my mere expression of thanks likewise does not suffice.

This thesis was supported in part by the CNCSIS PNII IDEI 17/2007 research grant.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This PhD thesis describes the research activity carried on as part of the doctoral program entitled “Reliable Implementations for Cryptographic Systems with Testability Facilities”. Our work is mainly focused on constructing efficient testing architectures for cryptographic systems without degrading their security.

The doctoral program addresses the domain of Computer Science, with emphasis on the Computer Hardware Design and Secure Self Test subdomains, operating in the field of encryption algorithms and coding theory, the Self Test endeavor being addressed by implementing effective on-line and off-line test solutions for Advanced Encryption Standard hardware realizations.

The extent to which *cryptography* is emerging in evermore aspects of the digital systems can be understood from the hierarchical representation in Fig. 1.1. This figure depicts the typical cryptographic techniques together with the integration and organization of these methods into higher level mechanisms.

The opportuneness of cryptographic research is justified by a growing interest manifested by the European Union toward this domain, in general, and cryptographic hardware implementations, in particular. The academic research in the area of security is advanced to the level of European financed Research Framework Program. The “STORK” project (Strategic Roadmap for Cryptography) was the first step [1] in constructing a common research infrastructure and it was financed through a FP5 program. The next framework program, FP6, included the “Networks of Excellence” and settled the foundations for “ECRYPT” (European

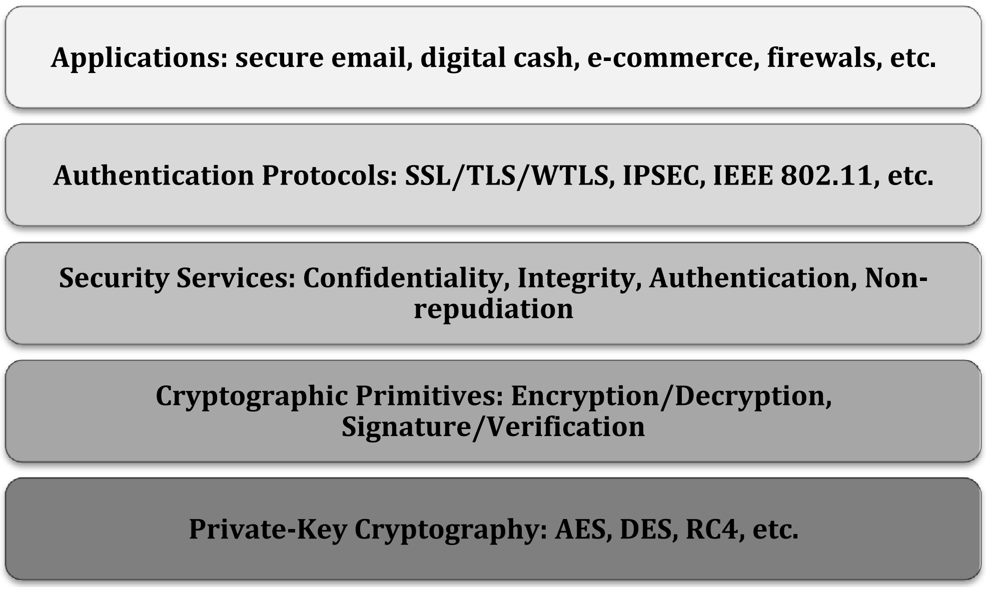

Applications: secure email, digital cash, e-commerce, firewals, etc.
Authentication Protocols: SSL/TLS/WTLS, IPSEC, IEEE 802.11, etc.
Security Services: Confidentiality, Integrity, Authentication, Non-repudiation
Cryptographic Primitives: Encryption/Decryption, Signature/Verification
Private-Key Cryptography: AES, DES, RC4, etc.

Figure 1.1 Cryptographic mechanism hierarchy adapted from [2]

Network of Excellence for Cryptology) project, started in 2004 as a 4 years project [3]. The current status of European investment toward this direction is represented by the "ECRYPT2" project [4] financed through an FP7 program.

The test engineering strategies represent one of the significant aspects of the semiconductor industry for its current and future development as revealed by the *Semiconductor Industry Association* in the *International Technology Roadmap for Semiconductors* [5]. The prospects of a sustained technological development, leading to reduced dimensions for the integrated semiconductors pose new challenges for testability.

The reason integrated circuit manufacturers consider including *testability features* into their digital implementations resorts to reducing product manufacturing costs. The *defect rate* represents an important element in this context. The implications of *defect rate* into current integration technology are evident when analyzing the case studies reported in [6] concerning the defect rate for the current high-end processors such as IBM Power5, Sun Niagara and AMD Opteron.

For the special case of cryptographic devices, yet another incentive for including testability measures within the architecture relates to their vulnerability to attacks and faults. In fact, a design enhanced with a poor testability mechanism is subject to malicious attacks, as a consequence of its security flaws. The terminology used by Fujiwara to presents the concept of testability offers a better understanding for this observation [7]. Testability is defined in [8] in terms of observability and controllability, referring to the available methods for enhancing the mechanisms by which one can observe and control the internal state of a device from exterior. This definition brings about an apparent contradiction between testability and security. Moreover, the literature contains references reporting the actual damage an attacker can do when facing a poorly designed testability [9]. One such example is reported in [10] in which is detailed the process of retrieving the secret encryption keys, required for unlocking encrypted television channels: the attacker makes use of an unsecured Scan Chain implementation.

Apart from the concerns associated with testability measures improperly adapted to the architecture of a security-enhancing system, the semiconductors are subject to defects as a consequence of their "wear lifespan" [11]. Moreover, after a system recovery from an attack, because of the techniques employed in order to circumvent the protection mechanism, the cryptographic module can be subjected to permanent defects. The fault attacks, aiming to purposely inject errors into the device, as a result of their invasive nature affect the system permanently. Some of the fault attack strategies are detailed in [11], [12], [13], [14], [15].

Considering the broad range of factors capable of altering internally a digital design, and in particular a crypto-system, and taking into account the generic strategies available for detecting and/or countering their effect, we conclude that a robust cryptographic implementation equally depends on concurrent and non-concurrent error detection mechanisms. Whereas the on-line solutions, operating concurrently with the device, allows for detecting errors manifesting into the most vulnerable elements of the architecture, the off-line strategies are intended for a wider specter of the affected modules. The non-concurrent test strategies, as will be presented, aim to verify correctness of all elements of the design and in consequence require the system to cease its normal operation.

The effectiveness of the proposed testing solutions is evaluated with respect to the hardware overhead entailed and the obtained error detection rate. The latter characterization is one of the relevant parameters for a testability measure, taking into account all possible errors scenarios affecting the system. As a consequence, all fault models are considered when evaluating the fault coverage for a given scheme. Corresponding to a particular fault manifestation mechanism, the actual detection rate of the verification scheme is approximated by simulating a finite set of all the possible erroneous conditions associated with the fault model.

The thesis is organized in 7 chapters. In the following a brief description of the content for each chapter is revealed.

Chapter 1 announces the domain of the thesis, discussing the domain's relevance with respect to other initiatives and research activities similar to our research.

Chapter 2 introduces the fault models hierarchically, corresponding to the levels of hardware abstraction. The faults models are gradually differentiated starting with the defects manifesting at the algorithm level, continuing with the faults at the Register Transfer Level, further at the logic level of description and specializing at the transistor and layout levels. The presentation also investigate the extent to which logic level fault models can be used in modeling or at least covering lower level defects. Additional fault models, relevant for the VLSI technology are described, together with their relationship and coverage consideration with respect to the already presented hierarchical fault models.

Chapter 3 covers the problematic of testing in a detailed manner, discussing the economic motivation of test. It also summarizes the test challenges foreseen by leading integrated circuitry manufacturers as the technology is progressing deeper into the nanoscale era. Off-line test strategy is then introduced together with our justifications, supported by the field's literature, regarding the applicability of the Built-In Self Test approach for the requirements of a secure, autonomous test environment. The BIST architecture is presented, detailing the test pattern generation and response compression mechanism based on Linear Feedback Shift Registers. The notable BIST configurations proposed in the literature are also briefly introduced. This chapter also explores the on-line testing strategies: the conventional self checking mechanisms are introduced such as hardware redundancy, code-based error detection and time redundancy mechanism. Moreover hardware parameter monitoring is characterized as well as the concurrent adaptation of the off-line testing schemes.

Chapter 4 examines the cryptographic domain and its applicability in the face of various existing security challenges. It also presents the AES algorithm in a greater detail, covering diverse hardware implementation and optimization aspects. The chapter culminates with the presentation of our high speed AES design.

Chapter 5 begins with a short description for the related work on secure testable implementation of the AES algorithm, as it is described in the literature. Apart from the related work, the chapter analyses our original contributions to the problem of testable design in the context of a high speed AES implementation.

Chapter 6 eventually draws a conclusion of the thesis, marking its contributions and future work.

Chapter 2
VLSI Faults

A brief description of the terminology and problematic of VLSI faults is presented in order to prepare the discussion for introducing the fault models. It also details the mechanism by which faults manifesting at various locations within a system are propagated and physically observed within system's interior or exterior environment, as the Fig. 2.1 depicts.

In order to properly analyze and devise effective countermeasures against the effects of the defects potentially affecting a VLSI integrated circuit, the basic terminology and definitions regarding the hardware dependability issue are provided. The following brief presentation is compliant with the widely accepted taxonomy in the domain [16]. Terms like *defect*, *error*, and *fault* describe incorrectness in a system from various point of views, and although it appears they can be used interchangeable, their formal definitions reveals differences and causality relations between them:

- A *failure or service failure* is encountered whenever the behavior of the system deviates from its correct functioning. The failure marks a transition in system's conduct to a condition in which the structure doesn't carry out its intended functionality. The failure is temporally delimited by the *service outage* and the *service restoration*, as the moments when the system's incorrect behavior begins and ends respectively [16].
- The states perceivable by system's collaborators, out of all system's states, are referred to as *external states*. The *system's service* is defined in terms of all system's external states. An *error* is characterized by the deviation of at least one system state from the correct service state. From these definitions, it can be observed that there is a causality relation between the system's service failure and an error in system's states. In other words, the *error* encompasses those of the system's states which may lead to a service failure [16].
- A *fault* is the "hypothesized caused" of an error [16]. The fault may constitute an internal or external cause for the error. Usually, a fault will initially lead to an error in the service state for an internal module of the system, leaving the external state of the system unaffected. A fault remains *dormant* when it doesn't cause a failure as a consequence of affecting an external state. A fault is said to be active, when its effect develops into a failure [16].

A suggestive depiction of the relation between faults, errors and failures is presented in Fig. 2.1. A subtle distinction can be made between an *external fault* –

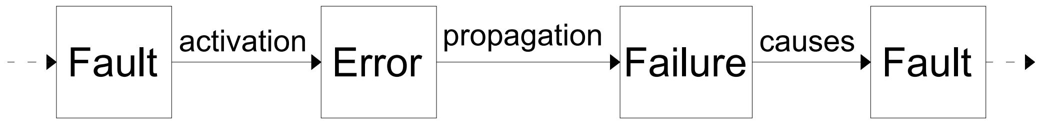


Figure 2.1 Fault propagation adapted from [16]

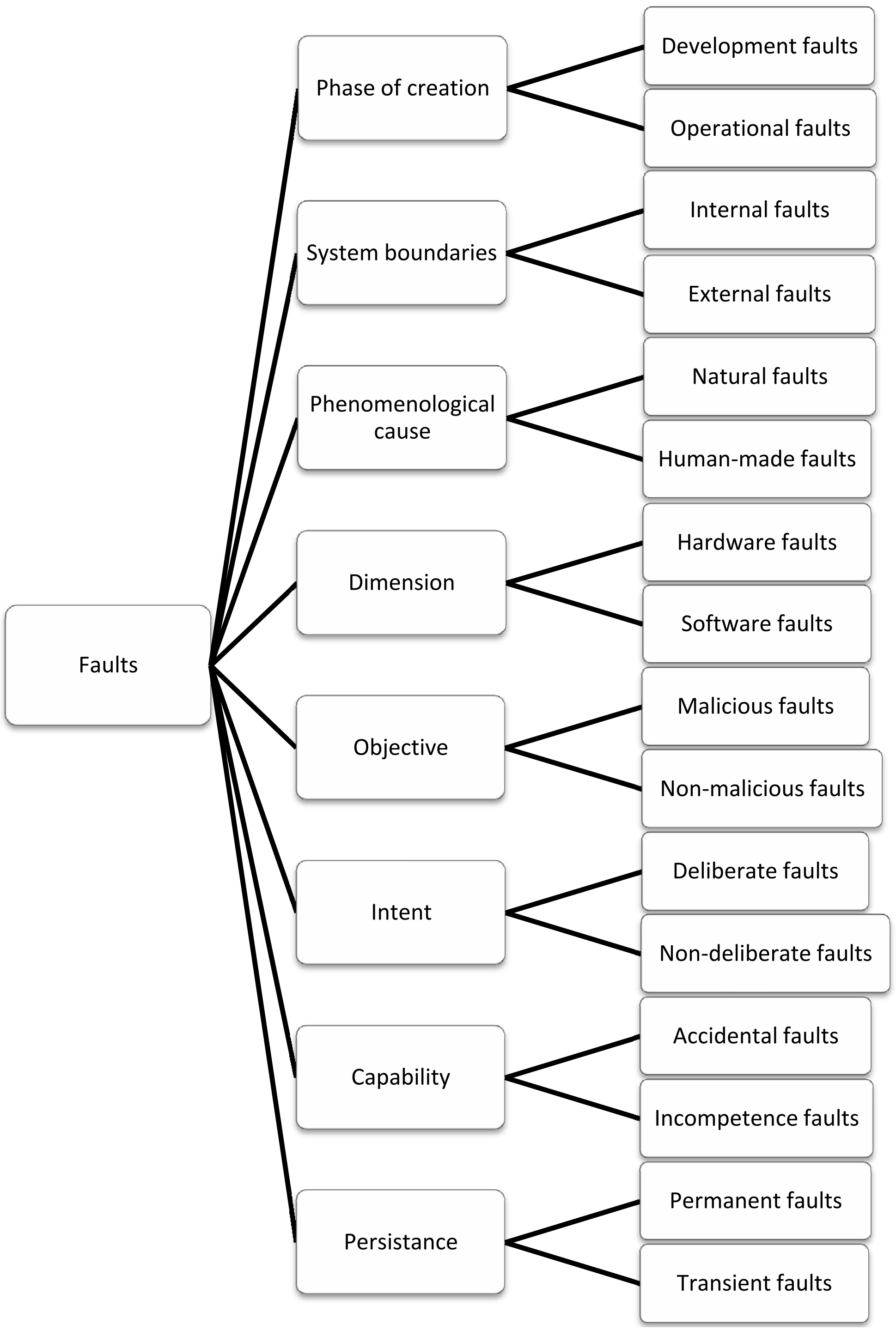


Figure 2.2 Fault taxonomy adapted from [16]

as the fault that manifests at the system's service interface, affecting system's inputs - and an internal fault. The latter is referred to as *vulnerability* because only by its presence within system's internals, an external fault can corrupt the system causing an error and possibly following failures.

In [16], the faults are classified with respect to various criteria and as a result the mentioned reference builds a complete fault taxonomy. The faults are classified in elementary classes graphically depicted in Fig. 2.2. It is important to note that the proposed classification does not provide any information regarding the physical or electrical conditions characteristic for the enumerated faults. The construction of the fault taxonomy allows for a particular fault to belong to more than a single class. As a consequence, the reference [16] continues with developing a combined fault taxonomy by evaluating all possible fault category intersections of the fault classifications. This results in the definition of 31 classes of combined faults which extend over the development, physical and interaction domains of a system. The physical domain encompasses all fault classes which specifically affect the hardware, while interaction domain for example includes the external faults.

The principal merit of the combined fault classification resides in its importance in outlining and establishing the relevant error detection mechanism with respect to VLSI testability and in particular considering cryptographic systems' reliability. The extended fault taxonomy together with the crypto-system's specification and its intended operational environment determines the possible fault context. For example, an important segment of the *human-made* faults [16] can be avoided provided that the specifications are correspondingly defined in order to reduce the possibility of *omission faults* (failing to perform a required action when the specific context demands it) or *commission faults* (wrong actions performed by the human factor). Human-made faults differentiated by the user's intent are classified into *malicious* and *non-malicious* faults [16]. The reasoning follows with distinguishing between those human-made malicious faults that appears during the development phase which will manifest during the system's use, and the malicious faults introduced during system's operation. Similar arguments are constructed for each fault category.

With respect to persistence, the faults are grouped into permanent and transient faults. A continuous and lasting fault is referred to as a *permanent fault* [17]. Generally, a permanent fault at the hardware layer marks an irreversible alteration of the semiconductor substrate. The permanent faults can be addressed also as *hard* or *strong* faults due to their reproducible nature [16]. *Transient faults* are determined by temporal environmental conditions [17]. They are referred to as soft faults, emphasizing the fact that they cannot be systematically reproduced, as a consequence of being subjected by the environmental context [16]. Yet another class of faults, encountered in the field's literature is represented by the *intermittent faults*. In [16], the category of intermittent faults cumulatively describe both transient faults and *elusive permanent faults*. The permanent faults, that surface as a result of an unstable hardware or software state ("unstable or marginally stable hardware" [17], or significant system load) are also referred to as *intermittent faults* [17].

The distinction line between intermittent and transient faults requires a special discussion [17]. The main difference lies in their quality of being repairable: the intermittent faults being determined by hardware's physical conditions, defective design at the hardware or software level or by a mere averse but stable environment can be repaired by replacing the defective module. In the latter case, component redesign can be performed in order to compensate for the biased environmental context [17]. However, the transient faults can't be repaired as the hardware structure is unharmed. Moreover, the momentary character of the environmental conditions wouldn't permit compensating for the effects of system's ambient.

The following sections undertake the problems of fault and fault modeling from the point of views associated with the classical development cycle, introducing the technology dependent defective conditions.

2.1 Fault Models at the Hardware Abstraction Levels

Previous section introduced the fault as the "hypothesized cause" for errors. The precise physical characterization of the fault is unlikely to be determined or even required for a particular fault condition, especially for intermittent faults. This is one justification for the hypothetical nature of the fault's physical description. The inherent complexity of analyzing and detecting system's faults at the physical level is ameliorated by modeling them at higher abstraction levels. In addition, the versatility of the hardware description languages in detailing a design at its various abstraction layers is favorable for modeling system's faults manifesting at precisely the same level.

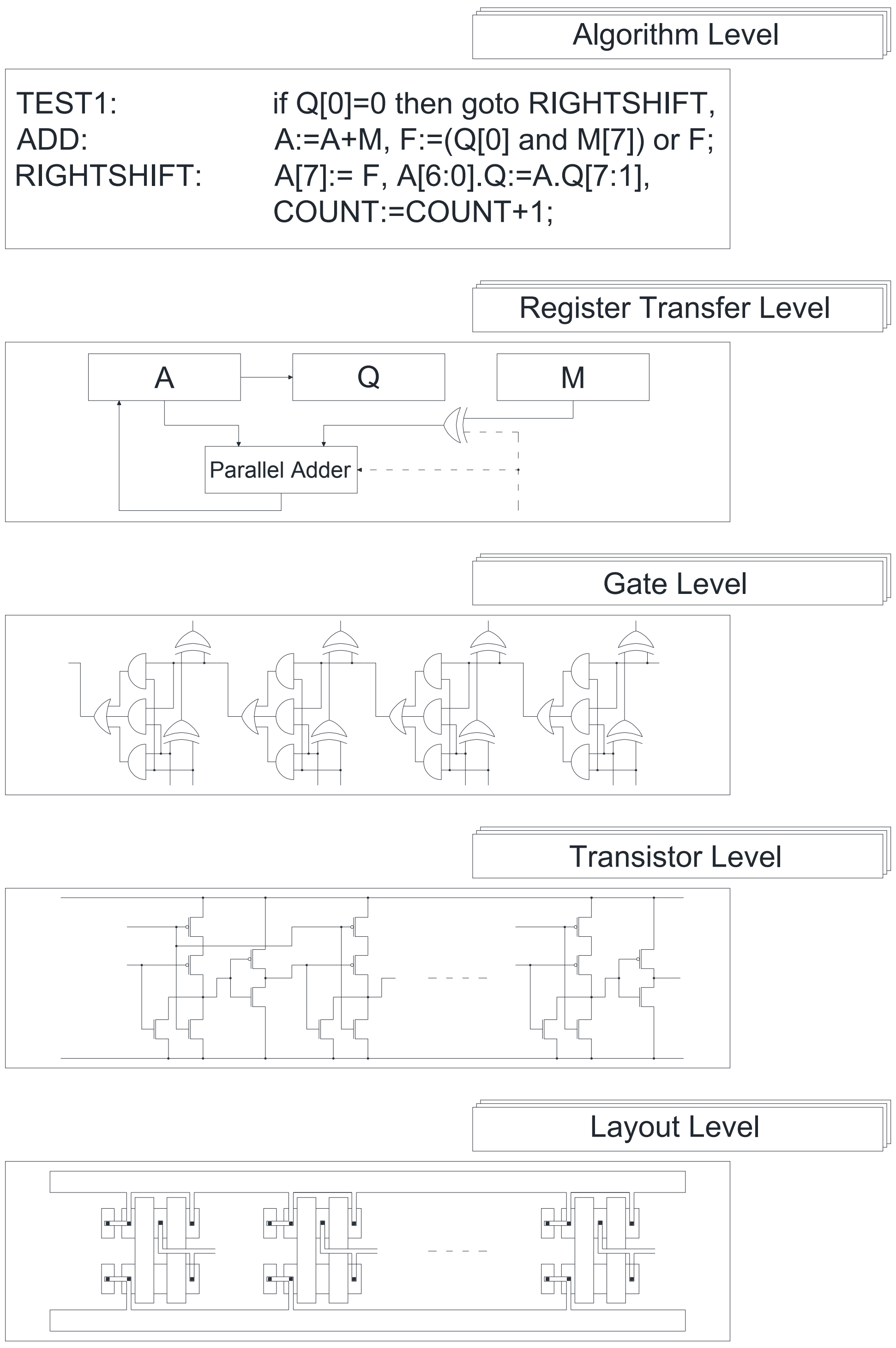
Fault modeling can be viewed as generalizing the real physical conditions of defects across the abstraction levels for a system [18], [19]. The motivation for this approach can be identified as being related to conveniently representing the complex physical faults into a form adapted for faster fault simulation and test generation. It must be noted that a diametrically opposed approach exists: the Inductive Fault Analysis, detailed in conjunction with the lower-level layout fault models.

Some criterions for evaluating the fault model appropriateness are presented in [20]. Ideally, all of the following requirements should be simultaneously satisfied for a fault model:

- The fault model should reflect the physical defect with maximum accuracy
- The model should be able to characterize every possible physical circumstance
- The fault model should exhibit computational efficiency by facilitating the fault simulation and test generation.

Unfortunately, given the intrinsic computational intensity associated with the simulation of semiconductor materials at the physical level and the requirement for rapid fault simulation for a particular fault model, no defect abstraction was found to cumulatively accomplish the above requirements. In order to address a wider range of possible defects, and considering the limitations of each fault model, for the purpose of testability, several fault models are usually taken into account when generating the test process. It must be noted that over time the fault simulation, and test vector generation for the conventional fault models sustained various improvements as evidence of a continuous effort to ameliorate the trade-off between fault model generalization and its associated computational effort.

Several fault models were proposed, which can be conveniently structured according to the typical system development abstractions. A comprehensive list of the fault models can be found in [21] and [22] respectively. The standard integrated circuits' development paradigm detailed in terms of the abstraction levels at which a system can be described is graphically depicted in Fig. 2.3.



```
TEST1:          if Q[0]=0 then goto RIGHTSHIFT,
ADD:            A:=A+M, F:=(Q[0] and M[7]) or F;
RIGHTSHIFT:     A[7]:= F, A[6:0].Q:=A.Q[7:1],
                COUNT:=COUNT+1;
```

Figure 2.3 Hardware design abstraction levels

Higher levels assume higher abstraction descriptions in Fig. 2.3: the algorithm level describes the functionality of the system in the most formal approach, while the geometric (or the layout) level embeds the system's logic into the semiconductor and connection layers. Corresponding to the increasing degrees of details encounter while traversing from top to bottom the abstraction hierarchy in Fig. 2.3, the design of an integrated circuit is structured in a top-down approach. A circuit is constructed in a repetitive refinement process, by adding the necessary details in order to migrate from a higher abstraction level to the next one. The higher the level of abstraction the lesser particular implementation details it covers and the more logic functionality it encapsulate [20]. Moreover the high level design entities, usually being part of the system's description in a hardware description language tend to include the much of the system's functionality.

When taking into account the possibility of organizing and associating the relevant fault models according to the level of abstraction, the same top-down development approach can be adapted to modeling the defective digital systems behaviors. More specifically, treating the possible faults hierarchically entails significant advantages regarding the computational effort for testing the circuit [23]. In [tt] is described such an approach of correlating integrated circuits' design phases with the testing by differentiating the test objectives according to design's various stages.

The benefit of handling the fault models at the appropriate level can be clearly understood when dealing with the problem of testing for faults at the layout level. It is almost impossible to check for logic or algorithmic errors at this level and layer, due to the intrinsic complexity of the layout netlist. A more appropriate attempt would be to perform a thoroughly verification after the algorithms were constructed and before moving to the next development phase of system refinement (Register Transfer Logic). Instead, for the physical level (the geometric abstraction) a more suitable verification approach would consist of checking only for incorrect placement of the geometric primitives (conductors, doped regions), the verification for excess polysilicon or similar rules and constrains characteristic for this lower level.

As a consequence, the fault simulation and its associated test vector generation should be performed starting from the highest abstraction level. The computational effort for fault verification increases with the reduction of the abstraction degree. For example, it is much less computational intensive to verify correctness for an algorithm, compared to verifying its physical implementation [23], [24]. An important observation pertains to the ability of the fault models addressing higher abstraction levels to cover faults at the lower level. Anticipating the fault models to be presented, the testing for errors at the algorithm level will inadvertently address also a portion of the stuck-at faults. This is because stuck-at faults are handled using an input vector for which the response is checked against the known correct result. It is easy to understand that some of the test cases or typical utilization scenarios employed for algorithm validation implies stimulating the algorithm with those input vectors which would detect stuck-at faults in the integrated device. Ideally, the fault models applicable to the higher level of the design hierarchy should be effective in detecting the majority of all possible faults. For such a scenario, the fault models characteristic to lower abstraction levels are expected to detect fewer and fewer defects as a consequence of the high fault coverage assured by the faults modeled at previous phases. In consequence it seems more efficient to first address faults at the higher abstraction level, evaluate the coverage of the testing process for the fault models situated at lower abstraction levels, and specifically address only the undetected fault at these lower abstraction layers [23]. Apart from the modularity of this hypothetical approach the speed of simulation and test generation represent another advantage as the algorithm is expected to be tested much faster than RTL descriptions, gate-level descriptions and all the other less abstract levels. However due to the imperfection of the available models [23] and the complexity entailed by constructing a comprehensive fault model to incorporate the characterization properties for all relevant fault models, a high percentage [23] of the possible faults not covered by higher level fault models remain unaddressed. This observations shows how loosely coupled the available fault models operate. Another explanation for the disjunctive nature of the fault models with respect to covering defects targeted by other defect models relates to the applicability of the test. Algorithm fault models, although being able to address faults at lower degrees of abstraction, are intended to asses algorithms' correctness in order to continue the process of design refinement for the next description level (RTL in this case). There is a high probability that the test vectors employed during the behavioral verification are discarded afterwards due to the impossibility of verifying defects for the design stages not yet reached.

2.1.1 Fault Models at the Algorithm Level

The algorithm level of design operates with system's specifications, building the general functionality of the circuit. It does this by incorporating the algorithms required to perform system's required functionalities. It is also referred to as the behavioral description level as a consequence of being the least implementation specific design stage. The end result of this stage describes the design in terms of a hardware description language (such as Verilog or VHDL).

The field literature undertake the fault models targeting the algorithm level in several references such as [25], [26], [27], [28], [29]. In the context of hardware description languages such as VHDL, Verilog as well as the newer SystemC and SystemVerilog, the testing for behavioral faults at the first design stages - algorithm level and Register Transfer Level - is performed using specific language constructs and is covered by the terminology of hardware description language *test benches*.

The possible sources of errors at the highest abstraction level can be handled using the following conditions [23] :

- For each variable used in the algorithm, its initial value must be taken into account. Usually variables are implemented using registers, which can be explicitly initialized with a specific configuration or implicitly by resetting all of its storage cells. Although the registers materialize only at the next design phase (RTL), it remains necessary to assure proper initialization for each variable.
- When dealing with integer values, close attention is required to distinguishing the unsigned and the signed variables. This distinction becomes important when extending values over a larger number of bits and when shifting or rotating operands.
- Another observation regarding algorithm variables, as remarked in [26], pertains to the situation in which a variable's value remains "stuck" at its maximal or minimal value. For example, an 8-bit long unsigned integer could have its value fixed at 255 or at 0. From the description of this condition, the ability of higher level fault models to cover lower level defects became evident. The very description of this defect implies that

along the design hierarchy at lower levels (gate, transistor or layout level) a condition determines a line or a bus to remain stuck at a value regardless of its driving value. At the gate description level such a situation is modeled using the stuck-at fault paradigm. Another important observation pertains to the effectiveness of testing for a defect using a fault model more abstract than the situation requires. For example, suppose that at the behavioral level the intention is to test for all possible stuck situations, i.e. verify that a particular variable remain stuck at any possible configuration. For an 8-bit integer value, it will be verified against all 256 possible configurations whereas at the gate level, a maximum of 16 faulty conditions need to be tested for identifying any stuck condition: 2 stuck conditions for each line. However, at the algorithm level, by testing for fixed values at domain's boundary (maximal and minimal values), these 2 configurations can detect defects characteristic for lower levels.

- A synchronization dependent operation could behave faulty and be executed or elude execution independent of the synchronization signal [23], [26].
- Fault models can be derived for the algorithm's statements as presented in [26]:
    - For a function call, the returned value can be subjected to the same defective behavior as a variable, thus checking it for fixed values.
    - In an iterative construct, such as the *for* instruction, the operations within the loop could be executed or could be entirely avoided regardless of the actual value of loop's condition.
    - For a selection control statement such as the *switch* construct, the following faulty behaviors should be taken into consideration: all cases are executed, no case is selected, only the cases corresponding to the extreme values of the control variable are selected
    - For a conditional statement, i.e. *if-then-else* construct, the algorithm execution can encounter the following erroneous situations: the code expected to execute only for the true condition is permanently executed, the code associated with the false value of the condition is permanently executed, the code associated with the true value of condition is executed when the condition is false while the true condition determine execution of the code associated with the false condition.
    - For the *assignment* instruction, the left hand variable remains unchanged or is changed to the maximal or minimal values of its range.

A possible shortcoming of the above mentioned fault model relates to their limited scope in considering the possible faulty conditions, i.e. taking into consideration only the boundaries, or the extreme values for the domain of a variable [23]. For example, the possible faulty execution of a loop instruction is limited only to infinite execution or no execution at all. One motivation for this limitation is that the model has to be able to practically check the algorithm against the proposed defective situations. Taking into consideration all the intermediate values for a variable would make the test unnecessarily long. Although the possibility to check against some of the intermediate values is recommended, the second problem would be to correctly determine the representative value with respect to loop execution counter. The same observation is relevant for the limitation of the range of a faulty variable, a returned value of a function, control statement, conditional statement and assignments.

As already mentioned, the specific faulty situations treated at the algorithm level for a design and in particular the verification details for the fault models proposed, implicitly cover defects addressed by lower level fault models. The above mentioned hypothetic situation in which stuck-at faults are covered by the testing process associated with algorithm variable's fault models is relevant in this context. In fact reference [23] formulate a percentage of almost 85% of the lower level defects being detected by the algorithm level fault models.

2.1.2 Fault Models at the Register Transfer Level

At the RTL abstraction level (also referred to as functional level), the system is described as a collection of components. Each component is considered a black-box, from designer's point of view. The functionality of each component is described using high level behavioral modeling constructs. In consequence, the task of verifying the design can be easily separated for each module because at this level, the faults at module's interfaces are not taken into account. The possible defective behaviors are similar to those for algorithm's fault models. For example, in the case of a full adder cell module, the functional verification, when using RTL fault models, needs to assure that the adder cell computes the sum and the carry out bit correctly in every circumstance. Because the truth table for this example is of reduced dimensions, all input configurations can be taken into account. For simple design modules, such as an adder cell, multiplexer, decoder or any module with similar complexity, exhaustive testing is preferred because it can detect faults at lower levels (as will be presented as the N-detect test vector sets methodology). The number of entity's inputs determines the complexity of the test simulation which is why, for more complex modules, the exhaustive test approach is less practical. In a iterative logic array [23] like the ripple carry adder, the full adder cell is repeated for a number of times. In consequence, although the number of input configurations for a 32-bit adder is extremely large for exhaustive simulation, because of all cells being exact copies of the same basic type, a simple test set composed of 8 vectors is required for completely testing the design: each of the 32 cells receives the same input vector, thus requiring only 8 vectors for the cell's 3 input. For modules, not manageable using this approach trade-off solutions are presented in the field literature, capable of reducing the dimensions of the test process by selecting a set of the inputs observable at one of the outputs [23], or using a more radical solution of adding new test points to the module, similar to the ad-hoc design for testability approach with the same name presented in [30].

Concerning the majority of the components, for which fault verification acceleration techniques are yet to be developed, the test process follows a different course. It can be said that the majority of the integrated modules are not designed at the RTL level. Some of the design rationale for current integration process relates to being able to reduce system's costs and increase their reliability [24]. Not only that the integration technology facilitated an ever increasing degree of integration, but together with the advance of analog and mixed signal design paradigm, the diversity of the modules capable of being integrated expanded rapidly. The complexity of the underlying structure for current integrated circuits can range from that of an microcontroller to that of a DSP incorporating blocks as diverse as memories, PLLs, analog-digital converters [24] to name only a few.

The testing approach for complex integrated systems requires a different handling, as reference [24] emphasize, due to the intractability to the lower design abstractions (transistor-level). Notwithstanding the clear advances in computing power and simulation tools, the fault simulation at the discrete level of transistor fault models is not feasible. In consequence the testing requirements are segmentalized and dispatched to each of the system's component blocks. The smaller components can thus be more thoroughly verified and involving a modest computational power. Because the components describes parts of the final architecture (compared to code blocks or procedures, involved with algorithm level descriptions), these components are described by RTL entities, requiring specific fault modeling at this level. It is evident that such an approach is favorable with respect to test generation tractability and design modularity, especially when considering using third parties specially designed IP cores, extensively tested by the provider, and accompanied by the test vectors. However, this segmentation approach raises some problems regarding the interface between system's components and the unpredicted inter-component interaction faults, which would have been treated by a system-wise simulation. Nonetheless, it is considered [24] that many of the effects of faults modeled at the switching network level can be mapped into characteristic behaviors (or fault models) at the functional level. In other words the complexity of lower level simulation is delegated to the more abstract levels. Only after the correspondence between lower level faults and their specific manifestation at the RTL level was established, the test pattern generation can proceed. Needless to say, the mapping between the specific low level fault manifestation and the defective behavior at the functional level is technology dependent, meaning that the process needs to be repeated for each integration technology.

A typical use of functional fault models is for testing RAM modules [24]. The results presented in reference [24] reveal that the majority of all faults related to the memory's decoder and read-write logic can be effectively addressed in terms of faults at the memory's cell level, which can be detected by functional fault models. The defective situations can be modeled as stuck-at faults at the gate-level or short (coupling) faults at the transistor level. Although this modeling is performed in terms of the gate-level fault models and transistor's fault models, the test process which detects the faults uses the RTL description. It is easier to understand the mobility of the fault models at various levels when realizing that a fault at a lower level has influences at higher levels as well. The fault testing at the functional level of those defects modeled at lower abstraction levels was proposed and used, with literature references of RAM functional modeling such as [31]. There are similar reports dealing with fault modeling for complex designs, such as microprocessors, at the functional level [32].

2.1.3 Fault Models at the Gate Level

The gate level, also referred to as the logic level, describes the design in terms of primitive Boolean operators. Each constituent modules of an RTL design is described in terms of these primitives. The logic level allows expressing the behavior of system components using Boolean relations. Moreover, the computation of the test vectors is facilitated by the ability to express module outputs using Boolean

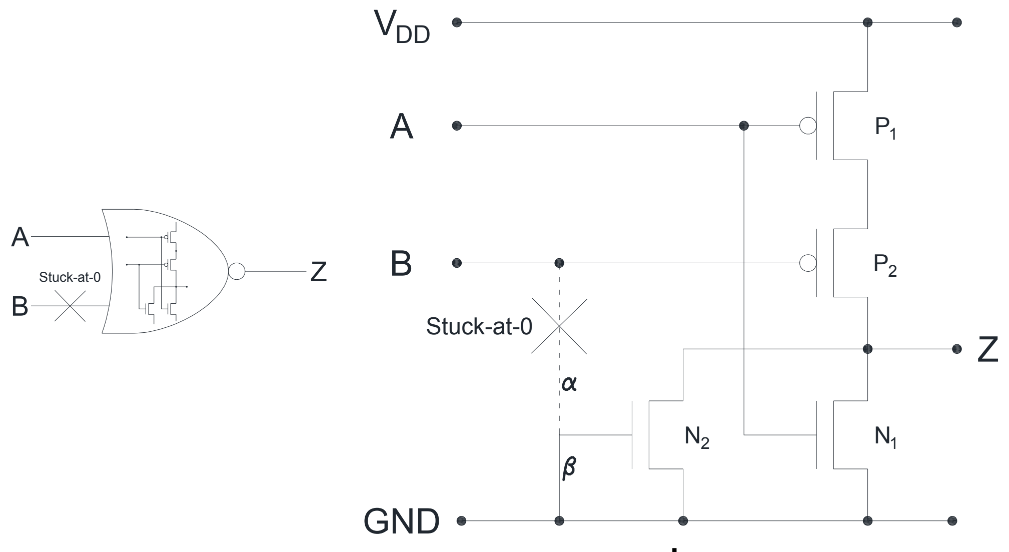


Figure 2.4 Mapping transistor level defects into stuck-at faults

equations [24]. It is the final stage of the design which remains technology independent. The fault models and the test vectors generated at this level are portable across implementation technology. The gate level fault models were introduced early into the development of integrated circuitry: the circuit's reduced complexity and integration scale permitted to formulate the simplifying assumption that any fault condition will influence at least one variable of the Boolean equations (each wire correspond to a variable in the Boolean equations describing the circuit), yielding a faulty response [24].

The actual issue for the logic level fault models relates to mapping the lower level faults into faults at the gate level. A brief graphical representation of how the gate-level stuck-at fault model is interpreted at the lower abstraction levels is depicted in Fig. 2.4. In fact the stuck-at fault model is illustrated by the general condition in which a wire connecting two gates of the design is interrupted and instead the severed connector stuck to either the ground or the power line potential. For example, the fault affecting the input B of the NOR gate in Fig. 2.4a, is translated at the lower level, as the Fig. 2.4b depicts, into an open affecting the line α and a short to the ground of the N_2 MOSFET transistor's gate electrode, denoted as β in the figure. Although such a situation can occur, it is not the only one that at the gate level manifests as a stuck-at fault, as this section reveals. However such inter-level correspondence needs to be performed for each technology individually. Apart from the technology independence of the model, the cardinality of the possible fault set is significantly reduced: considering the NAND gate in TTL technology, depicted in Fig. 2.6, at the switching level, the number of faults depends on the number of failing mechanisms for each of the 9 elements of circuits in Fig. 2.6, as will be presented later in this section, whereas when modeling the design at the gate level, the failing mechanism reduces to a maximum of 6 gate terminals errors.

The stuck-at fault model represents the most used gate-level defect modeling tool. It owes this widespread due to its simplicity and intuitive behavior. It was first introduced by Poage in 1963 [24] and is widely used collectively both in academia and in industry together with other gate and switch level fault models. Because the perspective from which it describes defects remains technology independent the faults are considered to originate in the gate interconnections, from which its name – stuck-at or stuck-line fault. The advantages of this fault model can be summarized as follows:

- it can model many faults at the transistor and layout level [22].
- independence of the integration technology
    - the constructed test vectors for detecting stuck-at faults can be reused regardless of the integration technology
- the ability to detect unmodeled faults when the test vectors were generated as N-detect vector sets [33]
- the speed of fault simulation and test vector generation
- the abundance of automated tools for generating the test vectors, both commercial and academic [24]

Inputs		**Output (Z)**						
A	**B**	**Fault-free**	**A s-a-0**	**A s-a-1**	**B s-a-0**	**B s-a-1**	**Z s-a-0**	**Z s-a-1**
0	0	1	1	1	1	1	0	1
0	1	1	1	0	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	1	0	1	0	0	1

Table 2.1 Stuck-at faults affecting a 2-inputs NAND gate

Table 2.1 details the defective behavior for a 2-inputs NAND gate when a single stuck-at fault affects the gate. It depicts comparatively the gate behavior for the fault-free circuit and for all 6 possible stuck-at faults. The colored cells mark the deviation of the defective circuit's response from the correct output. The table reveals the mechanism by which the fault is identified: in this example each stuck-at condition is recognized by stimulating the circuit with at least one input configuration, referred to as *input stimuli* or *test vector*. The stuck-at-0 affecting output Z can be detected by more than a single test vectors while a particular test input configuration can detect more than a single defect such as the input (A=1, B=1) capable of revealing A and B stuck-at-0 and Z stuck-at-1 defects. This observation will be further capitalized upon when dealing with test generation acceleration techniques. As the previous example reveals, for each line of the circuit there are two possible faulty conditions *stuck-at-1* and *stuck-at-0*. At the gate level their presence is described as forcing the affected line to remain at the respective logic level regardless of gate's input stimuli.

When dealing with typical gate interconnection networks it is important to take into consideration the various locations at which stuck-at faults can occur, and the associated implications. More precisely, every fanout branch is considered as a different wire apart from its driver. For example, in Fig. 2.5, the driver α (the output of the NOR gate) is directly connected to its fanout branches: the input β of the NAND gate and input γ of the XOR gate. A stuck-at fault affecting α determine the faulty value for β and γ rendering the NAND and XOR gates' output erroneous, whereas when a stuck-at fault affects the input β of the NAND gate, the XOR gate's output remain unaffected. The same applies for the γ input. It follows that each

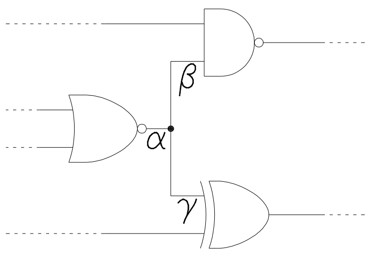


Figure 2.5 Stuck-at faults for circuit fanout

terminal of a gate needs to be accounted separately for fanout branches when evaluating the possible circuit location where stuck-at faults can manifest. The only simplifying condition remains when the output of a gate drives a single input.

The number $n_{Stuck\text{-}at}$ of the lines of a design that can be affected by the stuck-at faults is computed based on the fanout for each driver. In a circuit, both the primary inputs of the circuit and the outputs of each gate represent drivers. The approach presented in [22] for determining the number of wires takes into account a parameter f_i for each gate representing the number of other gate inputs commanded by the respective gate output. Considering the design D, consisting of N_D gates, referred to as g_i, with $i=1..N_D$, we define the function $w(g_i)$ representing the number of wires associated with each gate g_i. Function $w(g_i)$ is defined in terms of the gate g_i's fanout f_i: for a fanout of 1 $w(g_i)$ is 1, while for a fanout f_i>1, $w(g_i)$ is f_i+1. The number of lines $n_{Stuck\text{-}at}$, which can be affected by the stuck-at faults is expressed in (1) according to the approach in [22].

$$n_{Stuck-at} = \sum_{g_i \in D} w(g_i) \tag{1}$$

In terms of the circuit simulation, the stuck-at defective condition is easily handled by converting the affected gate into a simplifying Boolean primitive. For the case of a NAND gate, as reported in Table 2.1, if affected by a stuck-at-0 on one of its inputs, the entire gate will be replaced with a connector to the power line because a 0 forced unto a NAND gate input will set the output to 1. If a stuck-at-1 affect the A input, the NAND gate can be reduced to a simple inverter: the inverter's input consist of the NAND gate's unaffected input. Similar circuit simplifications can be performed using elementary Boolean equations. Moreover the circuit reduction needs to be performed iteratively in order to incrementally reduce the circuit. For example if a stuck-at-0 affects the NAND output, the respective NAND gate can be entirely removed together with all its drivers not commanding other gates. Because NAND gates output remain fixed at the logic value of 0, this will determine further simplification and possible further reductions (if the element driven by the NAND's output are also NAND or AND gates).

With respect to the fault multiplicity, *single stuck-at fault* model and *multiple stuck-at faults* model are defined. Whereas the faults modeled at the functional or architectural level usually account for defects spread across multiple lines (a variable stuck at the maximum domain's value or the incorrect results of an adder) at the gate level a distinction is made between single and multiple affected lines. The single stuck-at faults can be detected using one of the numerous algorithms or CAD tools available. There are, as already said, algorithms (such as the D-

Algorithm) which guarantee that a test vector is generated for a fault if the fault is detectable. However, the restriction that a component is affected by single stuck-at faults represents a simplification not entirely justified in the context of VLSI device failing mechanisms.

The complexity of the entire system can be conveniently decomposed and assigned correspondingly to system's component, but in order to detect a fault, the component's input needs to be controllable and its outputs to be observable for the test process to effectively identify it. This requirement can be met by conveniently extending the design with testability measures [34]. However these mechanisms will increase device's complexity by widening the accessibility of the system, and additionally, a more complex design implies higher risk of faults.

Because every gate of a system is susceptible to being affected by defects, and especially as a consequence of increased complexity, multiple defects modeled as stuck-at faults can simultaneously influence the device. In a design with *n* connection lines that can be affected by the stuck-at faults there are *2n* distinct single stuck-at faults because in a this scenario each terminal of each gate can be either stuck-at-0 or stuck-at-1, but because only one defect can manifest at a given time there are *2n* distinct conditions. When considering multiple stuck-at faults, for the same design, there are a total of 3^n-1 distinct fault situations. This result is obtained as follows: every terminal of a gate can be in one of three possible defect states, namely being affected by stuck-at-0, being affected by stuck-at-1, or being fault-free; accounting for all the *n* wires into the design, there are 3^n distinct states out of which one state is fault-free.

The exponential growth of the number of different multiple stuck-at fault configurations, compared to the linear increase of the total number of single stuck-at fault conditions represents an important disadvantage for the multiple stuck-at fault model. Although the single stuck-at fault model does not address the 3^n-2n-1 possible defective situations there exist, however, some motives justifying the relevance of the single stuck-at faults for most of the unaddressed multiple fault conditions:

- The device can be partitioned in discrete components and in consequence the extent to which multiple stuck-at faults can manifest is considerable reduced. In fact the immediate result of circuit segmentation is the reduction in the number of lines that can be affected. Moreover, the probability that multiple stuck-at faults would cluster together in the same region of the design is reduced compared to the probability of disparate stuck-at faults. In consequence for smaller components the probability of multiple stuck-at faults and especially closely located multiple stuck-at faults is smaller.
- When using a "frequent testing strategy" [22]. This approach increases the frequency of the test procedure in order to prevent the semiconductor substrate to develop faults. The premise is that when the interval between two consecutive tests is small enough the probability of multiple faults (at least 2) to develop is reasonably small.

Regarding the second justification however there are situations in which multiple faults can appear concurrently into the design such as:

- Attacks circumstances in which a malicious user attempts to circumvent some of the device's function. For example, when developing a crypto-system the designers need to take into account the fault attacks in which attackers deliberately induce faults into the design. Not knowing precisely the affected location within device's internal structure, the attacker usually injects multiple errors and operates the erroneous results using differential cryptanalysis in order to retrieve sensitive information. The field literature contains references describing mechanism for injecting defects into integrated circuits [11], [12], [13], [14] and [15].

- Another possible situation in which the device can be subjected to multiple stuck-at faults in between two consecutive test processes relates to the peculiarities of integrated circuits failing mechanisms: as reference [22] confirms, and the following sections reveals, there are fault conditions at the physical level, which manifests as multiple (at least 2) higher level defects.
- Multiple faults can be present in a design before its production test. An important percentage of the manufactured devices are defective and thus discarded. After the manufacturing is completed extensive test are performed in order to verify devices' correct behavior. In such conditions, it is to be expected that more than a single defect to be present in the design.

There are literature references that exclusively treat the single fault models (stuck-at fault model as well as other fault models: stuck-on, stuck-open). Such an example is reference [35], dealing with two-rail code checkers. Apart from the simplified conditions required for analyzing the checker's self-checking property and effectiveness in detecting gate-level and transistor-level defects, the reference takes into consideration the assumption that the single faults' distance in time is sufficiently large as to allow the checker to finalize a codeword verification cycle. Similar approaches (also referenced in [35]) regarding the single faults assumption are [36] and [37]. However, references [22] and [24] assert that the multiple stuck-at faults are often easier to detect. The multiple defects are more probably to trigger an erroneous response detectable by single stuck-at fault tests [24]. Reference [22] concludes in this aspect that "in most cases, a multiple fault can be detected ". This observation, however, applies only for non-concurrent testing mechanisms.

As with the faults at the higher abstraction levels, it is important to evaluate the degree of coverage the stuck-at fault models offers for faults manifesting at lower levels. The ability of stuck-at fault model to cover real defects is relevant because many of the higher fault models were described in terms of the stuck-at fault model as already presented in the previous sections. When treating transistor level faults, however, the technological factor gains importance.

In the following the mechanisms by which faults at the TTL and CMOS transistors level are modeled by the stuck-at defects will be presented. For the 3-input TTL NAND gate depicted in Fig. 2.6, the majority of the physical faults affecting any of the integrated components are detectable by means of the stuck-at fault model.

The manifestation of a low level fault at the gate level offers insights regarding the effectiveness of the stuck-at fault model in covering physical defects. A close analysis reveals the following observations [23]:

- If the resistor R_1 is affected by an open defect the circuit output behaves as influenced by a stuck-at-0 fault.
- The comportment at the gate level of an open affecting any of the R_2, R_3, D_1 or D_2 elements cannot be modeled as a stuck-at fault and thus such a fault is undetectable using only the stuck-at fault model.

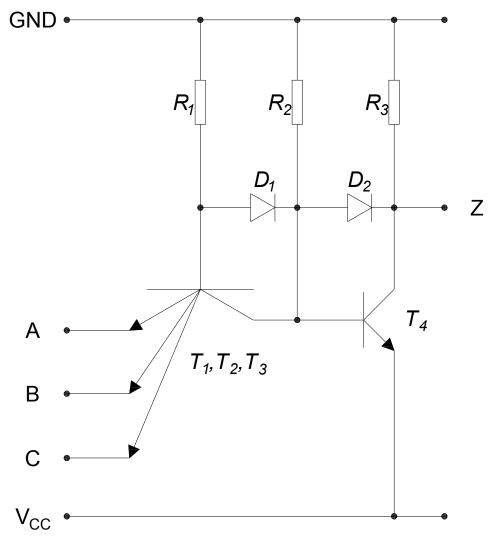


Figure 2.6 Three inputs TTL NAND gate adapted from [23]

- For transistors T_1, T_2 and T_3 an open emitter manifests as the respective input would be stuck-at-1, while a disconnected collector or a collector-emitter short manifests as a stuck-at-0 on Z. Still, disconnected base and collector-emitter short are undetectable in terms of the stuck-at model.
- The base-emitter short defects for T_1, T_2 and T_3 don't behave as stuck-at faults. However these defects are detectable when considering the stuck-at-1 defect for each of the 3 gate inputs.
- Defects of T_4 such as disconnected emitter, collector or base and shorted base-emitter junction behaves as stuck-at-1 output. The collector-base short and a collector-emitter short manifests by forcing the output at the 0 logic value.

The presented scenarios are intended to justify the effectiveness of stuck-at fault in modeling faults at the TTL physical layer. The majority of the integrated elements' faults can be detected when testing the circuit (the NAND gate in this case) against this fault model. Not all of the detectable defective conditions actually maps to a stuck-at defect. Such an example is a base-emitter short for T_1. However, when testing the circuit against stuck-at-1 for the input A, the defect is revealed. Of importance are the undetected defects. For example, if R_2 is disconnected none of the gate's ports remain stuck; however, the response of the gate is delayed. Specific fault models were constructed to reflect such behavior, otherwise correct from a strictly logic point of view, but considered defective when taking into consideration the gate's latency.

Regarding the mapping of CMOS transistor level defects to the stuck-at fault model, we consider the NOR gate depicted in Fig. 2.4. A first observation pertains to the simplicity of the circuit compared to the TTL technology represented in Fig. 2.6. The defects that can affect a CMOS gate can be structured using an approach similar to the one used for the TTL technology faults. Mainly, for each transistor, there are six possible defects to be taken into consideration:

- Three open defects, one for each terminal
- Three short paths: gate-source, gate-drain and source drain

The short defect conditions are characterized by the presence of an additional resistive path between the respective nodes, allowing the current to bypass its correct flow and follow the defective path. The gate-source and gate-drain conditions are referred to as *gate oxide short* defects [24], [38].

It is generally accepted that a gate oxide short does not necessarily modify the behavior of the circuit at the logic level; however it modifies it at the parametric level. Such defects usually affect the quiescent current or I_{DDQ}, modify the nodes' voltages or induce delays. The I_{DD} current for the circuit's quiescent state [39], [40], [41], [42] is a measure of the circuit's consumed power. For a resistive short in the design, the voltage between the shorted nodes increases the I_{DD} current. The difference between the I_{DDQ} current for the correct integrated device and the defective device is sufficiently large in order to detect the faulty one [42].

Moreover due to the degradation of the fault over time the defect initially not affecting the logic functionality of the circuit can end up by altering its correctness. Whereas it is important to detect the fault when it modifies correct functioning, it's even more important to detect a device for which the fault is degrading in time by shifting its effects from a timing related problem to a logical fault. The early detection of such conditions assures system's reliability. The fault manifested at the logic level can be easily identified after manufacturing using the stuck-at fault model whereas a latent logic defect as the gate oxide short requires parametric testing for fault detection and stress testing for accelerating the degradation process.

In some conditions however the gate oxide shorts determine at the gate level a behavior easily identified by stuck-at tests. More precisely, dependent on the actual resistance of the short, the gate oxide short can influence the output's potential to such an extent that the fault is discernable at the logic level. It must be noted that stuck-at fault models for gate oxide shorts doesn't guaranty fault detection in any circumstances. For this cause the presented literature references on gate oxide shorts recommend a single stuck-at test process with 100% fault coverage needs to be complemented with an I_{DDQ} test.

With respect to the source-drain shorts, the same observation stands regarding their nonobligatory logic level manifestation. If transistor N_1 of the Fig. 2.4 is affected by this fault, and if the resistance of the short is considerably lower than the equivalent resistance of the P_1 ad P_2 transistors, both conducting, the potential of the output when applying the inputs (A=0, and B=0) will remain fixed at the lower level. In consequence a stuck-at test for the Z stuck-at 0 will detect this faulty condition [24]. However, due to the extremely small on-resistance of the MOSFET transistors, the conditions of this scenario are not easily met. Similarly a short between the drain of transistor P_2 and the source of transistor P_1 causes the output to remain stuck-at 1 provided that the resistance of the short is lower than the resistive path from the output node to the ground potential. When both N-MOSFET transistors are conducting, their equivalent resistance is smaller than the on-resistance of each of them separately and as a result this defect is less probable to be detected when stimulating the circuit with the input vector (A=1, B=1) than for the case that only one input is at logic 1, such as (A=1, B=0). This situation makes the short fault detectable by either A stuck-at-1 or by B stuck-at-1. However none of the two stuck-at defects can actually model the defect but they can detect it. If the transistor P_1 is affected by a source-drain short, the fault cannot be modeled using the stuck-at fault paradigm. For the inputs (A=1, B=0), transistor P_2 is in the active regime, N_1 is also in the active regime. The short resistance connected between V_{DD} and the source of P_2 together with the resistance of P_1 which is saturated and the resistance of N_1 determines a resistive divider. Although the specific transistors' characteristic parameters and the short's resistance are able to shift the potential of the output, as our experiments presented in the following section reveal, the fault usually remains undetected by means of stuck-at fault tests. The quiescent current analysis however will indicate significant information because of the current increase for the case (A=1, B=0), which allows to establish an otherwise impossible path from V_{DD} to the ground (the short resistor, transistor P_2 and transistor N_1). The potential established at the output cannot be predicted exactly; however an objective characterization of the condition can be obtained by analyzing the output of the gates connected at the affected node. This analysis is in turn performed by evaluating the logic levels at the output (similar to the evaluation of stuck-at defects), by taking into account the following gate's correct response. If the next gate's output value is set correctly the defect is undetectable, whereas an incorrect output value is the evidence that the potential of affected gate's output was incorrect. Similar analysis can be performed for source-drain short on the remaining transistors. When addressing the open defects of CMOS circuits (for each of the source, drain and gate terminals), the stuck-at model is unsuitable for detecting them.

Regarding the test vector generation, the ability to detect a stuck-at fault is related to the concept of *accessibility* of a node of the tested structure. The concept of accessibility is defined in terms of *observability* and *controllability*. In order to detect a node that remain fixed at either 1 or 0 logic, the test process is required to stimulate the fault location by trying to set at that node the complementary logic value. For a stuck-at-0 fault, the input variables are required to establish at the node the high logic value and if this scenario is feasible for the particular structure of the circuit, it is called *1-controlable* [43]. The same rational can be applied for defining the *0-controlability*.

In order to properly detect the defect, the logic value at the affected node must influence the result at the output lines. This concept is referred to as observability, and allows the internal logic value at a node to be revealed at the outputs. This is not merely a property of the circuit, but requires specific computation for constructing the appropriate input vector so that the logic values of the other circuit components to not mask the value of the node of interest on the path to the outputs. The process by which an input vector propagate the logic value at a particular node in the circuit to the outputs allowing the respective node to directly influence the result is known as *path sensitization* [43]. In fact, for the combinational networks, the above mentioned requirements, of controllability and observability, or collectively known accessibility, need to be simultaneously accomplished in order to detect a fault.

Based on the conditions requiring controllability at a particular node and that node's observability at the outputs, a set of input vectors can be determined, each of which being able to detect the fault. The input vectors are expected on one hand to force the complementary logic value at the suspected defective location by conveniently activating the required paths from the inputs to the locations - these paths are also referred to by the concept of *fault cones* [21], or *logic cones* [44]. The same input vector will facilitate the path sensitization from the faulty node to at least one output. By comparing the logic response at output against the response offered by the correct circuit, a stuck-at fault affecting the suspected node is detected. Because of the possible multitude of input and output activation paths, one input vector can be suited for detecting more than a single defect. In

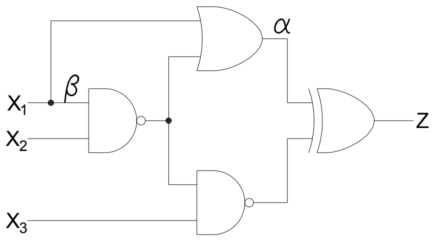


Figure 2.7 Undetectable stuck-at faults adapted from [43]

consequence, the test generation process is required to select the minimum number of input vectors, capable of detecting all stuck-at faults (for 100% fault coverage). Well established algorithms were developed for finding the input vectors which assure detection for each stuck-at fault. The process of vector generation and minimal input vector set selection is performed in many commercial and academic tools, such as ATALANTA [45].

However, there are circuit conditions which prevent a node to be controllable from the device's inputs or observable at the circuit's outputs. Such an example is depicted in Fig. 2.7. Suppose the circuit is affected by the fault α manifested as stuck-at-1. This defect cannot be detected because a proper input vector to stimulate it cannot be constructed. The controllability requirements demand for an input test vector to establish the logic value 0 at the α location in order to observe the supposed incorrect logic value at α by inspecting the outputs. However, this is not possible because of the circuit structure: more precisely, a closer look to the equation of line α reveals, after Boolean reduction, that its value is permanently 1 for the fault-free circuit, and thus it is impossible to command it to 0. This is a case of undetectable fault due to missing 0-controlability for location α. Similarly, the observability of a node to the output can be hindered by circuit's structure. For the same circuit in Fig. 2.7, a stuck-at-1 defect affecting line β cannot be observed at the outputs and thus cannot be detected. Although this defect can be controlled from circuit's inputs (X_1=0, X_2=1), no path from the line β to the output Z is available. Actually, because of the circuit structure, both paths for sensitizing β to the output present a masking effect. This is because a test vector is expected to generate different output for the fault-free and the faulty conditions. In fact the XOR gate input is (1, 0) in the fault-free case and (0, 1) in the defective situation. As a consequence the output is 1 for both scenarios rendering the fault impossible to observe and thus detect. The circuits which contain location undetectable for stuck-at faults are said to be *redundant* [43].

With respect to the size of the possible stuck-at faults, we already computed the number of all possible stuck-at location using equation (1). However, with respect to the test generation problem, the computational effort can be significantly accelerated when avoiding unnecessary computation. For example, considering the NOR gate from Fig. 2.4, we can observe that any input vector (which is actually a singular pattern) that detects a stuck-at-1 condition for one of the inputs, also detects the stuck-at-1 fault for the other input and the stuck-at-0 condition for the output. As a consequence, the concept of *fault equivalence* was introduced in order to condense the number of stuck-at conditions required to be addressed for attaining full fault coverage. It follows that two stuck-at faults are said to be *equivalent* if and only if, the circuit, being affected by each of them, has identical output functions [21]. For the NOR gate, each of the stuck-at-1 conditions for the 2 inputs and the stuck-at-0 for the output are equivalent. The direct consequence is that for an *n*-input NOR gate out of the *2(n+1)* possible stuck-at faults the test process will take into account only *n+2* faults: *n+1* for stuck-at-0 of each input for which the test vectors also cover the stuck-at-1 of the output and the last fault which covers any stuck-at-1 for inputs and the stuck-at-0 for the output. The process of eliminating the redundant faulty conditions is referred to as *fault collapsing* [21]. As a consequence by eliminating the equivalent faults from the set of all possible faults, the *equivalent collapsing set* of faults is obtained [21]. It involves characterizing each primitive gate in terms of the so-called *fault equivalent classes* and selecting a single fault condition for each class as relevant for the test generation process. The same rational can be followed for the other primitive gates: after identifying the equivalent faults. Although not a primitive gate, the EXCLUSIVE OR operator, due to its linearity, has fault equivalent classes formed of only of a single defect. As a consequence the fault collapsing doesn't reduce the number of defects needed to be considered for this type of gate. A detailed description is given in [21] with respect to the equivalence classes for each of the primitive gates including the inverter, as well as for the typical structures such as fanout nodes and the simple connection lines.

Another observation pertains to the extent to which a fault modifies a circuit. Considering the same NOR gate from Fig. 2.4, it is evident the difference in detectability for a stuck-at-1 influencing the output of the gate and a stuck-at-0 affecting any of its inputs. It can be observed that while output stuck-at-1 can be detected using any of the 3 possible input vectors (A=0, B=1), (A=1, B=0), (A=1, B=1), the test vectors that detects an input stuck-at-0 detects also the output stuck-at-1. This situation is different than the fault equivalence and is called *fault dominance* [21]. A stuck-at fault f_1 is said to dominate the stuck-at fault f_2 if all test vector detecting f_2 also detect f_1. In other words the fault f_1 is covered by at least all the test vectors covering $f_{2,}$ possible by others also. The dominating faults are further eliminated from the equivalence collapsing set in order to accelerate the testing process. Following a similarly rationing process for the other primitive gates, it can be deduced that for an *n*-input gate there are only *n+1* stuck-at faults needed to be taken into account in order to obtain a full coverage test set. For the NOR gate in Fig. 2.4 the minimal set of the *n+1* relevant faults is composed of 2 stuck-at-0 faults for the two inputs any of which also detecting the stuck-at-1 output and the stuck-at-1 for any of the two inputs. In [21] is also presented the *checkpoint theorem* which, by using the observation already discussed, compute the minimum number of connection lines required to be analyzed for stuck-at faults so that a fully covering single-fault test vector set to be obtained.

Some of the shortcomings for the stuck-at fault model were already mentioned. Despite its universal use, simplicity in simulation and test vector generation, a circuit for which the test process fully covers all single stuck-at faults is not guaranteed to be fault-free.

One problem of the model is its inability to neither model nor detect all open-faults for the gates designed in the CMOS technology. If none of the N-MOSFET or P-MOSFET transistors set the potential on the gate output, the output holds its previous value, this situation being referred to as *high impedance*. This observation is conveniently used in "data storage and discrete signal processing" [24]. Moreover, the high impedance property represents the premise for a design in which a line can have more than a single driver. It is usually the case for a system bus for which multiple sources can establish its value. However, when the open is the result of a fault, at the gate level the condition is expected to be detected, which, using stuck-at model cannot be accomplished. Further details regarding the detection strategy are offered in the following section.

Secondly, the stuck-at fault model cannot accurately describe all possible short defects. As already presented some of the short defects, can be detected using the stuck-at fault model. However, for the short defects not addressed, an appropriate detection mechanism is required.

As observed in [24], [43], [46] the circuit description at the gate level and the circuit representation at the transistor level are *topologically different*. At the transistor level there are connections lines which cannot be represented in a gate-level description. Although the reverse case is also possible – namely, having lines at the gate level not required at the transistor level – the former situation is more important with respect to the fault conditions because of the lower level defects unable to be modeled or detected by the gate level faults models. An example can illustrate this observation: the gate level description doesn't represent the power and ground lines for each gate. An open affecting either of the two lines translates to a fault which cannot be modeled using stuck-at faults, although such a situation can be detected.

Faults in CMOS integrated devices at the transistor level can also modify the implemented function rather than forcing a connection line to remain at a specific logic value [46]. As a consequence, the very fundamental assumption that the gate level description of the circuit is an abstract view of the transistor level structure becomes inexact for such a scenario. In order to understand this situation we consider the transistor level implementation of the function $Z = \overline{(A+B)\cdot(C+D)+E\cdot F}$ graphically depicted in Fig. 2.8.

Let the connection α be affected by an open fault causing the two pairs of N-MOSFET transistors to be separated. The behavior at the gate level of the circuit affected by the open fault α is described by a new structure which implements the function $Z_{faulty} = \overline{(A+B)\cdot(B+C)\cdot(C+D)+E\cdot F}$. The analytical expression for the function implemented by the faulty circuit in Fig. 2.8 was determined experimentally: the defective design was analyzed using the SPICE simulator. For each input configuration the voltage at the Z output obtained through simulation was converted into logic values while taking into consideration the technology factors (the MOSFET transistor model, the V_{DD} voltage and the threshold voltage) and from these the Boolean expression for the faulty circuit was derived using the ESPRESSO logic minimizer. In Fig. 2.9a is represented the gate level abstraction of the circuit in Fig. 2.8 without the defect while Fig. 2.9b depicts the gate structure which models the behavior of the same circuit when affected by the α open fault. The problem with CMOS defects that change the implemented function is that, although the correct value of the output is known (computed using the structure in Fig. 2.9a) the locations susceptible to stuck-at faults are unknown as a result of a changed structure. This example also illustrates the above mentioned topological differences between transistor-level modeling and gate-level description: the line mapping is not consistent in the sense that most of the connections from Fig. 2.8 are absent in the structures of Fig. 2.9. However, as this example reveals the opposite is also true. This is because of the specific design methodology in the CMOS technology: the intended function is implemented by a P-MOSFET transistors network joint with an N-MOSFET transistor structure. The N-MOSFET network will

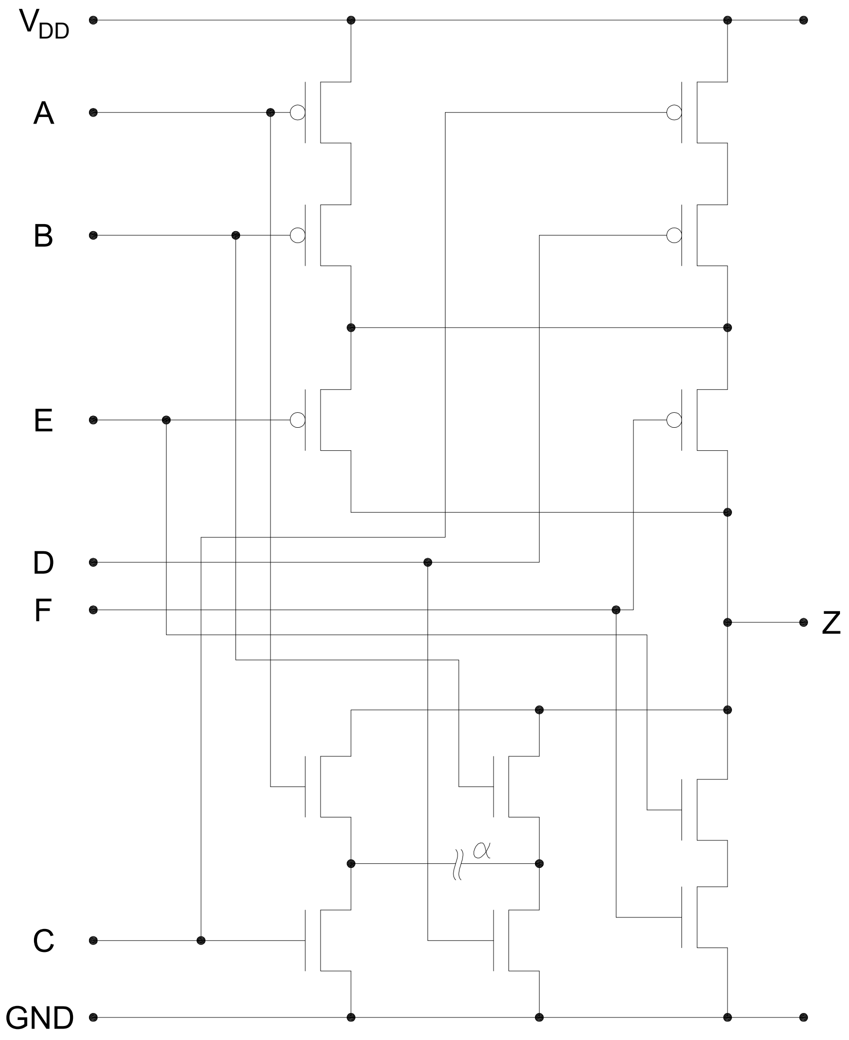


Figure 2.8 Transistor-level CMOS implementation of the function $Z = \overline{(A+B)\cdot(C+D)+E\cdot F}$

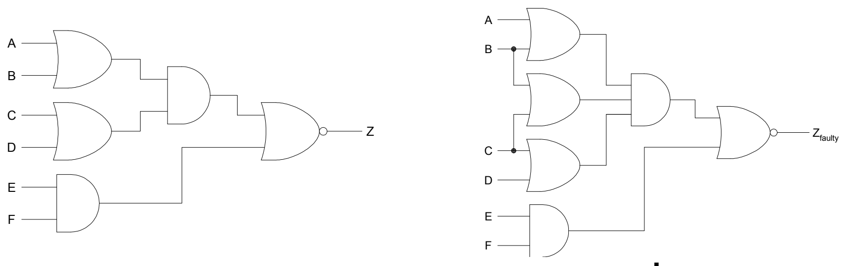


Figure 2.9 Effect of the open fault in Fig. 2.8 at the gate level

set the lower potential on the output and the P-MOSFET network will set the higher potential. Consequently the connections between primitive gates at the logic level have no direct correspondence in the transistor substrate.

Despite its shortcomings the stuck-at fault model remains widely used although it is not capable to specifically model transistor level defects. The field literature contains reports suggesting that test vector sets particularly constructed for a circuit (it is the case of N-detect test vector sets) can actually detect the majority of transistor and layout fault models. The straightforward test generation, easy simulation and "established practice" [47] recommends the stuck-at fault as a primary VLSI fault model.

2.1.4 Fault Models at the Transistor Level

Modeling the defects at transistor level is a result of stuck-at model's inability to cover or at least detect all faulty conditions. As already stated, the analysis at this level is technologically dependent. Due to the large utilization of CMOS technology for its numerous advantages [47], the following presentation is directed toward transistor faults in the CMOS technology. The defective conditions that are better described at this level were already encountered in the previous section, when the coverage of the stuck-at faults and its ability to map real defects were discussed. The main fault models at this level are transistor *open faults* also referred to as *stuck-open* transistor, and transistor *short faults* [23]. The situation in which the short determines the transistor to be permanently in conduction is known as transistor *stuck-on*. A distinction can be made for these fault conditions regarding the associated terminology. In [43] is described a condition similar to the stuck-on defect, called *stuck-closed* together with a fault situation called *stuck-off* described as an open defect.

A *stuck-open* defect is described as a faulty condition which renders the transistor unable to conduct. In fact an open for the gate, drain or source of the transistor lead to this defect because the MOSFET transistors' output is determined by the relation between the voltage difference between its gate and source terminals and transistor's threshold voltage. In other words, due to a specific fault affecting the source to drain path or due to floating terminals, the MOSFET transistor can be modeled as being in the blocked regime. Such conditions can be described at the transistor representation of the circuit by defects like f_1, f_3, f_5, f_8, f_{10}, f_{12}, f_{14}, f_{16}, f_{18}, f_{20} or f_{22} from Fig. 2.10. In [43], the stuck-open situation is interpreted as a permanent blocked source-drain connection. If the resistance of the source-drain path is close to that of a fault-free but blocked transistor the defect is referred to as stuck-open, whereas substantially higher impedance defines the stuck-off defect. However the two defects, stuck-open and stuck-off behave similarly with respect to their detection. The difference is perceived only when investigating the quiescent current.

Considering the defect f_{20} to affect the N_1 MOSFET transistor, we apply the input vector (A=0, B=0). Both P-MOSFET transistors are in the active mode, conducting which allows the output to be settled at the higher potential, interpreted as logic 1. The response is correct, because the output potential is established by the P-MOSFET transistors. If the next input vector is (A=1, B=0), the output potential is established by the defective transistor because neither N_2 nor P_1 are conducting. As a consequence, with transistor N_1 not being able to set the output potential, the gate's output logic value remains unchanged from its previous switching due to the high impedance property of the CMOS designs. In consequence, the output value for the second vector remains at logic 1, which is evidently, erroneous. The 2 vectors employed for analyzing the defective behavior detected the stuck-open defect, however if the first vector would have been (A=0, B=1), the result, although affected by the defect, would have been correct. It is possible for the previous input vector to mask the presence of an open fault.

In order to detect stuck-open faults two input patterns are required to be applied, as opposed to the stuck-at fault model which requires a single input vector: the first vector is referred to as the *initialization pattern* and the second as the *test vector* [23]. The test vector is required to render evident the defect: the input configuration will activate the specific path of the current flow, which in the correct device would allow the defective transistor to influence the output. The initialization

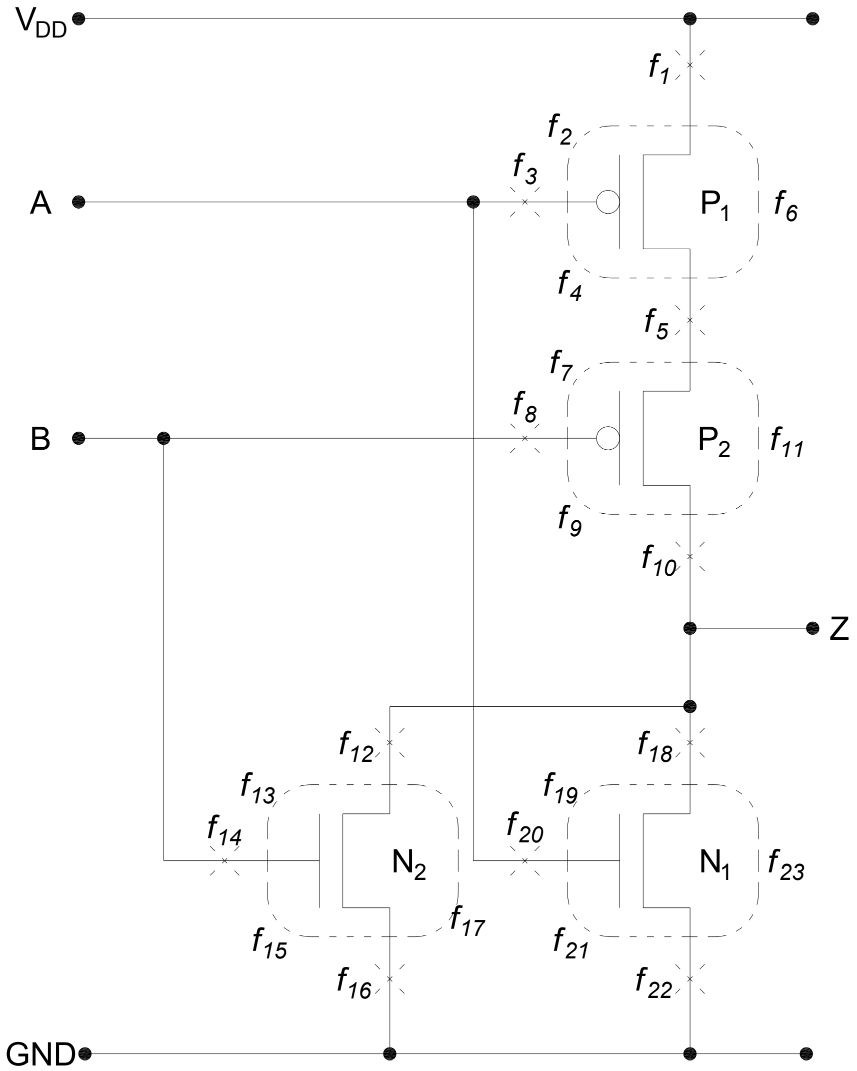


Figure 2.10 CMOS transistor-level faults for a 2-inputs NOR gate

pattern is expected to establish a logic value at the output complementary to the value established by the second vector for the correct device. The failure of the second input pattern to modify the logic value at the output is then an evidence of the transistor stuck-open defect.

A closer look at this defect reveals its sequential behavior: the value of the output is retained from the previous computation due to CMOS's high impedance property. In a sequential design this behavior can be interpreted as similar to a storage element operating for at least one clock cycle.

Another transistor-specific defect is the *stuck-on* transistor. The transistor, in this condition, is permanently in the active mode, or conduction. This behavior is modeled as a permanent short between the source and the drain of the transistor. The resistance of the stuck-on is comparable to the on-resistance of the fault-free transistor when conducting. Provided that the drain-source resistance is considerably lower, for the faulty situation, than the resistance of the source-drain connection for the fault-free transistor, the defect is identified as *stuck-closed*. Another physical cause for the stuck-on behavior is the absence of the polysilicon layer [23]. The stuck-on condition was analyzed in the previous section with respect to the ability of stuck-at faults to cover the lower-level defects. As already state, dependent on the resistive properties of the short and parametric behavior of the remaining components, there are situation for which the defect can be detected by observing the logic value of the result (referred to as logic monitoring [23]). However, there are situations in which due to the actual resistive characteristics of the transistors and the short, the logic value at the output remains correct. However, it was showed that monitoring the "steady state current consumption" [24] or the quiescent current, in most cases, is conclusive in detecting stuck-on defects [23], [24]: a resistive ladder between the power line and the ground potential increases the quiescent current by some orders of magnitude.

The analysis of transistor level defects reveals a gap between stuck-at fault model coverage and the actual extent of possible defects at the switching layer. For this reason it is important for the test process to take into account the possible transistor defects when building a test vector set or constructing an on-line fault detection strategy. As already presented, the gate layer and the transistor layer for a design are topologically different. As a consequence, when operating with the transistor fault models, the circuit is required to be described in terms of the underlying integration technology, namely the MOSFET transistors for the CMOS paradigm.

The typical implementation methodology for a design described at the Register Transfer Level when targeting the ASIC platform include design's *synthesis* followed by the *place* and *route* steps [48]. The synthesis typically uses, as building blocks, predefined simple components, extensively tested, for which functional parameters such as logic, timing, physical, and electrical properties are completely specified. The basic components are referred to as standard cells, and are delivered in so-called standard cell libraries. The synthesis process maps the RTL description to instances of the standard cells delivering a cell-based circuit representation as its result. The standard cells however, can still be logically modeled in terms of technology-independent constructs. The placing process determines the location of each standard cell instance within the design's final technological-dependent netlist. Its result is usually influenced by various constrains such as throughput, latency, power consumption. The routing process generates the necessary connections between the already placed components. The final netlist represents the adequate stage for transistor fault model testing.

The transistor fault models can be applied also at the smaller scope of the standard cells. For example, the 2-input NOR gate represented in Fig. 2.4a represent a standard cell component of the [49] standard cell library. For the NOR gate in Fig. 2.4a the CMOS transistor level representation is given in Fig. 2.4b. Similarly the component in Fig. 2.9a can describe a standard cell element with its CMOS implementation detailed in Fig. 2.8.

The observation that transistor faults can be detected at the standard cell level, coupled with the minimal standard cell library of [49], composed of primitive gates and storage elements, permits the transistor fault models to be used, even at the gate-level circuit description: the synthesis step maps the RTL constructs into standard cells, which in turn represent basic blocks for which the underlying structure is already known. In consequence the result of the synthesis can be interpreted both as a transistor level representation and as a gate level representation.

The transistor faults can be simulated using a different approach compared to the stuck-at faults simulation, because of switching level technology-dependent nature which requires simulation closer to semiconductor material level. The process is exemplified for the 2-input NOR-gate depicted in Fig. 2.10 by taking into account all transistor defective conditions already discussed: stuck-open and short defects (the latter including the stuck-on faults). The analysis of gate behavior was performed with the SPICE simulator using TSMC's 180 nm BSIM3v3 MOSFET model [50]. In order for the simulation to be reliable with respect to the actual performance of real CMOS devices, we commanded the 2 inputs of the NOR gate using CMOS inverters and we indirectly analyzed the output of the NOR gate through the output of another inverter connected on NOR gate's output. In this manner, we assured the logic levels, which depend on the threshold voltage of the component transistors, are correctly interpreted. All inverter gates and the NOR gate were constructed using the same parameters for the P-MOSFET and N-MOSFET transistors. "Real life" integrated devices exhibits a wider range of process variation, and with respect to this aspect, our simulations lack the transistors' parameter heterogeneity. Nevertheless the experiment allowed observing the effect of faulty conditions over the quiescent current as well as the influence of the short resistance over the electrical and especially over the logic behavior of the circuit. The short resistance was considered to be in the range 1kΩ-20kΩ, according to what the authors of [47] report to have found in the laboratory experiments. Another reference providing experimental boundaries for the short resistivity is [51], in which the short resistance was found to range in a interval comparable to the one in [47]: between 100Ω and 2kΩ, although it remarks that "dependent on the degree of transistor mismatch" the limits can vary.

Table 2.2 summarizes the transistor-level fault simulations performed. The table illustrates the logic behavior of the CMOS 2-input NOR gate when affected by each of the faults represented in Fig. 2.10. Moreover, as already stated, the specific behavior of the bridging fault is highly dependent on the particular physical parameters of the affected nodes and its neighboring lines. In consequence, the defect does not modify the logic behavior whenever the output voltage of the affected node maintains the correct relation with respect to the threshold potential of the transistors it commands. Because each short defect is characterized, besides its location, also by its resistance, the *critical resistance* is defined as the short resistance for which the correct logic behavior of the circuit can still be distinguished from the faulty logic behavior. The R_{short} column presents the value of the critical

Faulty behaviors			Inputs	A=0 B=0	A=0 B=1	A=1 B=0	A=1 B=1
Fault No.	Fault type	Fault activation condition					
Fault-free			Output	1	0	0	0
f_1	P_1 Open			Q_n	0	0	0
f_2	P_1 GOS	R_{short}<=1kΩ		0	0	0	0
f_3	P_1 Open			Q_n	0	0	0
f_4	P_1 GOS	R_{short}<=1.8kΩ		0	0	0	0
f_5	P_1/P_2 Open			Q_n	0	0	0
f_6	P_1 SD Short			1	0	0	0
f_7	P_2 GOS	R_{short}<=1.8kΩ		0	0	0	0
f_8	P_2 Open			Q_n	0	0	0
f_9	P_2 GOS	R_{short}<=3kΩ		0	0	0	0
f_{10}	P_2 Open			Q_n	0	0	0
f_{11}	P_2 SD Short			1	0	0	0
f_{12}	N_2 Open			1	Q_n	0	0
f_{13}	N_2 GOS	R_{short}<=3kΩ		0	0	0	0
f_{14}	N_2 Open			1	Q_n	0	0
f_{15}	N_2 GOS	R_{short}<=1.6 kΩ		1	1	0	0
f_{16}	N_2 Open			1	Q_n	0	0
f_{17}	N_2 SD Short	R_{short}<=3kΩ		0	0	0	0
f_{18}	N_1 Open			1	0	Q_n	0
f_{19}	N_1 GOS	R_{short}<=3kΩ		0	0	0	0
f_{20}	N_1 Open			1	0	Q_n	0
f_{21}	N_1 GOS	R_{short}<=1.7kΩ		1	0	1	0
f_{22}	N_1 Open			1	0	Q_n	0
f_{23}	N_1 SD Short	R_{short}<=3kΩ		0	0	0	0

Table 2.2 CMOS transistor-level faults manifestation for the 2-input NOR gate in Fig. 2.10
Q_n stands for the output's previous value
GOS stands for gate oxide short
SD Short stands for source-drain shorts

resistance obtained experimentally through SPICE simulations. It can be observed that some of the faulty conditions can be detected using gate level fault models. For all stuck-open defects the detection relies on a test vector pair as already discussed. With the exception of fault f_6 and f_{11}, the remaining defects exhibit deviations from the correct behavior at the logic level. The two faults however are detected by measuring the I_{DD} current in circuit's quiescent state. From the simulations performed it was observed that the I_{DDQ} method guaranties to detect transistor shorts with certainty higher than the stuck-at fault model, as also reported in [23], [24], [46], [47]. The influence of transistors defects and in particularly of short f_6 from Fig. 2.10 over I_{DD} is described graphically in Fig. 2.11 and Fig. 2.12. The V_{DD} potential was set to V_{DD}=1.8V and the parameters for the MOSFET transistor model were choose so that the threshold potential to be V_T=780mV; the short resistance is 1.8kΩ. The correct circuit behavior is depicted in Fig. 2.11. In Fig. 2.12 as compared to Fig. 2.11, the quiescent I_{DD} current is emphasized. As already mentioned the I_{DD}

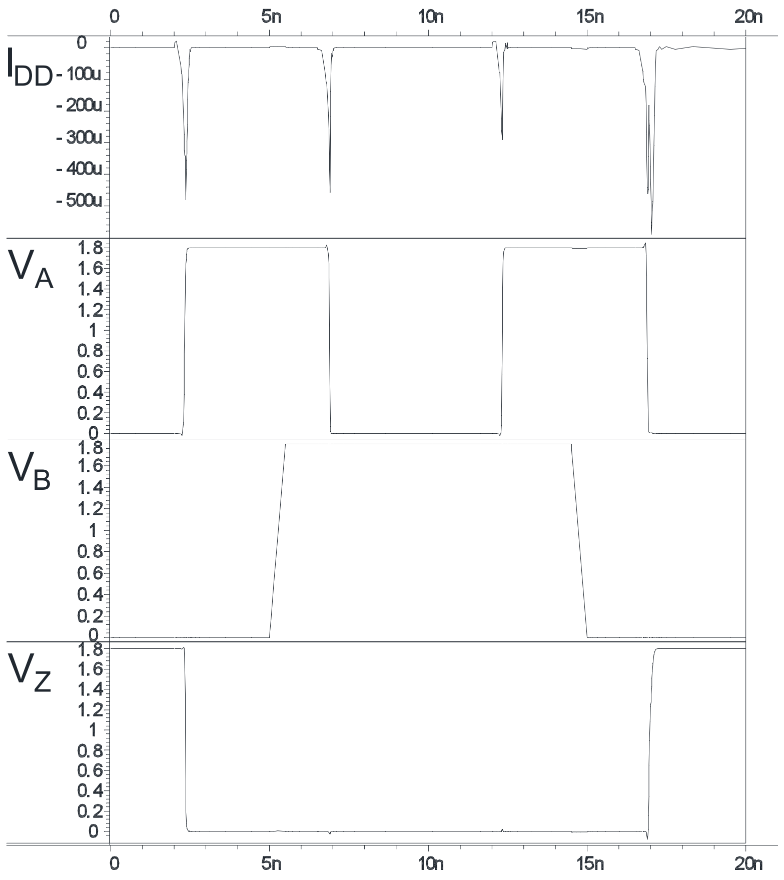


Figure 2.11 SPICE simulation of the 2-inputs NOR gate from Fig. 2.10

current is to be measured after the transient regime is over and the signals are stable. The Fig. 2.12 also marks out the effect of short f_6 over the output potential, V_Z.

In the previous section we concluded that in particular situations, given the specific resistance of the transistors, when in conduction or in the blocked regime respectively, and the resistivity of the short, the fault can be identified at the logic level or not. However, the detection is based on the assertion that a particular relation between the parameters of the circuit must be met. For example considering defect f_6, it can be detected by the (A=1, B=0) input vector only if the equivalent resistance of the short connected in series with the on-resistance of transistor P_2 is lower than the on-resistance of the N_1 transistor and the resistive divisor shifts the potential of the output toward V_{DD}. While given the parametric variance in "real life" integrated devices, this scenario can occur. In our simulation environment, with the same parametric properties for all transistors, this is not feasible because of the perfectly symmetric characteristic of the P-MOSFET and N-MOSFET transistors. In Fig. 2.12, the effect of the short resistance, "trying" to pull the output potential toward V_{DD} is also marked out as can be observed when comparing the V_Z potential to the output potential of the correct circuit from Fig. 2.11.

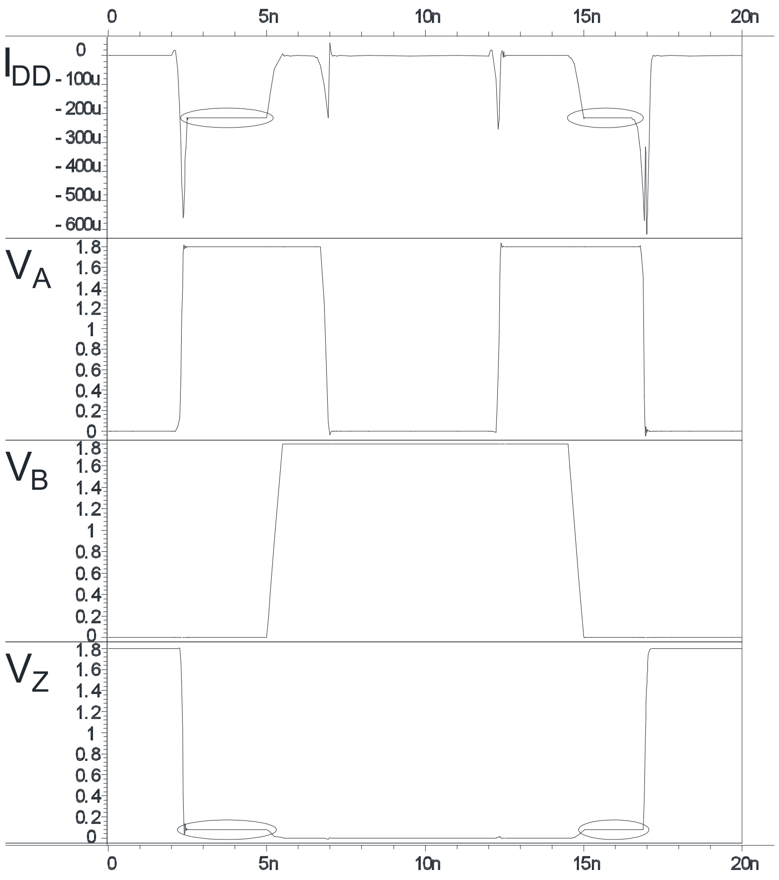


Figure 2.12 SPICE simulation of the 2-inputs NOR gate from Fig. 2.10 affected by the short f_6

In Table 2.2, the open defects' characteristic of conserving the previous' operation value is denoted by Q_n. The open defects behavior, as already presented, can be detected at the gate level because it requires two distinct consecutive input vectors: the first one for setting the output's initial logic value and the second one to command the output in the complementary logic state. As a consequence, provided that the previous input vector of the stuck-at test set, correspond to the initialization pattern, and the current test vector triggers the affected cell's output to the complementary level, the respective stuck-at test vector set assures transistor's open detection.

In short, this section presented the behavior and detection mechanisms for the conventional fault models at the transistor level which can be grouped according to the common transistor failure modes into:

- open defects: detectable by a test vector sequence conveniently constructed
- short faults: detected with high certainty by means of quiescent current although there are situations in which the short can be detected by a stuck-at fault.

2.1.5 Fault Models at the Layout Level

The layout level for a design is obtained at the end of the place and route process and describes the circuit in terms of geometric primitives specific to the physical implementation level. Fault models at this very low level were introduced as a result of the inability of higher level fault models to properly address layout defective conditions. In [46] are described possible defects originating in the circuit's physical layer:

- A surplus of polysilicon (conductive material) can connect any physical point of the netlist to either the power lines, the ground potential or to other points in the design. Although these faults can be usually mapped at the transistor level as typical short faults, the diverse scenarios for defective connections occurring between standard cell instances justifies the need for low-level physical defect descriptions.
- The omission of the oxide (electrical insulator) between the gate and the bulk, source or drain can be especially difficult to model in terms of the transistor-level defects, not to mention the stuck-at faults. For example a missing oxide manifesting as a contact surface between the gate and the source for an N-MOSFET transistor could determine the input line connected at the gate to be permanently stuck-at-0 while affecting also the functionality of the current standard cell.
- A defect condition specific to the geometric primitive level is a bridging fault translated at the transistor level as multiple shorts, effectively bypassing a transistor and shorting two connection lines.

One mechanism for handling the complexity of fault modeling at the layout level is the Inductive Fault Analysis [23]. Contrary to the approach we used when presenting the fault models hierarchically, the Inductive Fault Analysis starts from the lower level faults and subsequently refines the physical defects in terms of faulty behaviors which can be described using higher level fault models. The higher level defects are thus "induced" through analysis by describing the lower level faults at the higher abstraction level. The Inductive Fault Analysis is directed by the following principles [23]:

- The range of all physical faults is assembled statistically using experimental data from the integration process.
- The effects of the physical faults at higher levels (gate level, for example) are determined
- Faults at the higher levels are classified and ranked based on their occurrence probability. The process of ranking higher level defects is performed based on the number of physical faults manifesting as the respective high level defect. Moreover the effort toward a better coverage can be directed for the highest ranked gate level faults, in order to make the test process efficient.

The physical faults which are probable to affect the design are gathered statistically from the integrated designs. The physical fault is considered to manifest in the area of a so-called *spot* or *point defect* [23]. The size of the spot and its frequency per unit of area are defined as probability density functions [23]. Further details regarding the Inductive Fault Analysis technique are offered in [23], [24] and [46].

2.2 Bridging Fault Model

A bridge fault is defined as a short between two lines. The short fault model discussed in the previous section refers to shorts within a standard cell. However the model can be extended at the gate level, and describes resistive connections between gates, as the one illustrated in Fig. 2.13a. The transistor level description of the defective condition in Fig. 2.13a is depicted in Fig. 2.13b. The electrical analysis of the fault, with respect to the transistor level condition, is performed using the model of the resistive ladder in Fig. 2.13c. In this section, in accordance to the field literature, we will refer to the bridge defects as short conditions manifesting at the gate level, such as the one in Fig. 2.13a. In [22] it is discriminated between *feedback bridging faults* and *non-feedback bridging faults*: feedback bridging faults are manifested as a short between the gate's output and at least one of its inputs giving the circuit a sequential or even an oscillating behavior and making the detection issue more difficult. Non-feedback bridging faults on the other hand do not incorporate feedback loops.

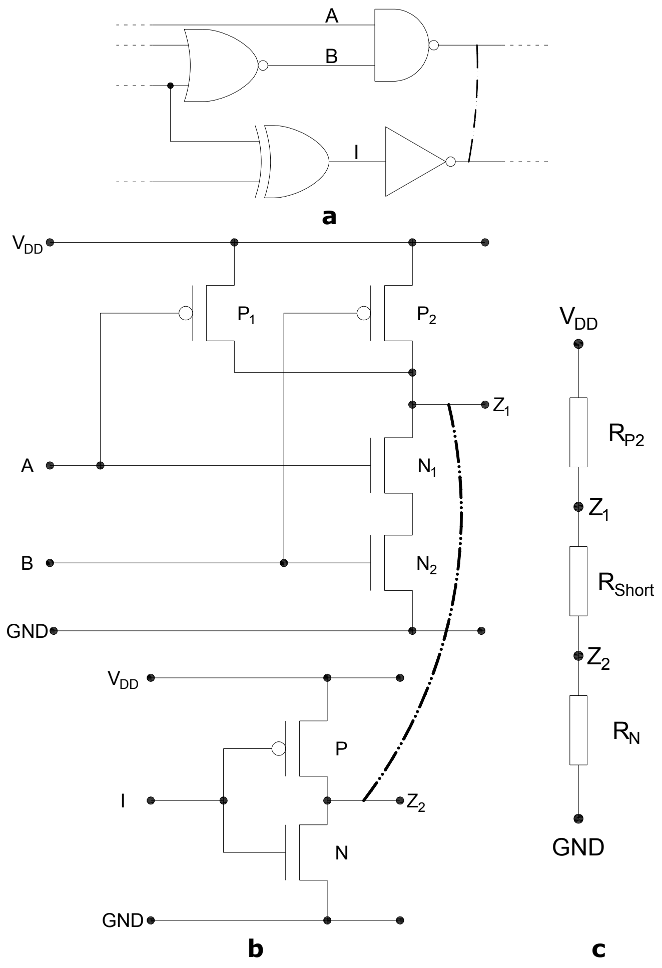


Figure 2.13 Bridging fault adapted from [24]

In bipolar technology such as the TTL and ECL technologies, shorts between gates' output are said to implement a *wired-and* or *wired-or* function. This behavior is generated by the fact that one of the two logic levels is "stronger" [47] and forces its associated potential to the node established at the oposite logic level. This behavior is not observed in the CMOS technology. The study [52] reveals, however, a weak correlation between the CMOS bridging fault model and the wire-and or wire-or fault model [47]. The typical CMOS bridging behavior is the so-called *dominant bridging fault*, for which one node forces the potential on the other node of the short. In [53] are introduced the *dominant-and* and *dominant-or* bridging fault models in which one node manifests as dominant but only for a particular logic value.

The behavior of a CMOS transistor fault, as already presented in the previous sections, depends on the resistive divisor formed between the power line and the ground potential. The resistive equivalent divisor for the structure in Fig. 2.13b is represented in Fig. 2.13c. One observation is that the smaller the short resistance the closer the Z_1 and Z_2 potentials are. The most important aspect relates to the manifestation of bridging faults. It is obvious that if both Z_1 and Z_2 are set at the same potential, be it ground's or power's potential, the bridging fault has no influence over the two outputs. However, when they differ, and only then, the explicit semiconductor's parameters determine the potentials at the affected nodes. For the input (A=1, B=1, I=1) both Z_1 and Z_2 have a near-zero potential. Yet for the input (A=1, B=0, C=1) the unaffected Z_1 and Z_2 would have complementary logic values.

In ideal conditions, the fanout for a gate doesn't influence the number of the possible bridging faults. This is justified by the observation that in a circuit, as the one presented in Fig. 2.14, the nodes α_1, α_2, β_1 and β_2 are interpreted at the same logic level. In the following exposition, the potential at a node as expected for the correct circuit is denoted by the *driver value*, according to [22]. If the driver values at the α and β nodes are equal, corresponding to the previous discussion, the circuit is performing correctly whether the short is present or not. If the driver values at the two nodes are different, provided that the affected potentials are in different regions with respect to the commanded gates' threshold values, the short is not affecting the logic behavior of the circuit. If the driver values at the two nodes are different and the affected potentials are in the same region with respect to the commanded gates' threshold potential, the bridging defect affects the circuit's logic behavior and all fanout lines (namely α_1 and α_2, and β_1 and β_2 respectively) are set at the same potential.

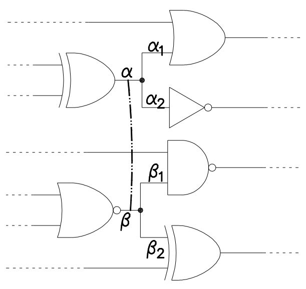


Figure 2.14 Bridging fault fanout

The consequence of the above observation is that the only circuit lines taken into account as possible being affected by the bridging defects are the gates' outputs, and the circuit's primary inputs because the internal gates' inputs are covered by the gates' output fanout. In [22] the number of all possible bridging faults affecting a design depends only on the number of gate outputs and the number of primary inputs. This gate-level fault model of the bridging defects is known as the *voting model*, in which the gates' parameter variation is neglected and the potential of the shorted nodes is set by the strongest driver (the one which "drives more current" [47]). This model is an adaptation of the bipolar *wire-and* and *wire-or* models to the CMOS technology.

The above analysis is sound with respect to ideal CMOS realizations. However, the physical transistors' threshold voltage is not uniform and according to [54] the physical gates can interpret the same logic value differently. In this case the same potential at the α_1 and α_2 lines can be interpreted different by the following OR and NOT gates. In this situation, the number of all theoretical possible bridging situations depends on the gates' output fanout. This bridge fault model is known as the *biased voting model*, because it takes into account the inherent variance in transistors' sensitivity (the transistor's threshold voltage). For this scenario all lines within a design can be affected by a bridging fault. The total number of bridging faults, expressed by (2), is equal to the 2-combinations of the number of all possible stuck-at defects as computed in (1).

$$n_{Bridging} = \binom{\sum_{g_i \in D} w(g_i)}{2} \tag{2}$$

As remarked in [22] the figure in (2) does not reflect the physical reality, because it assumes that every line can be shorted to every other line within a circuit. More precisely, at the layout level, after the place and route process have been performed, it is evident that not every line is physically close to any other line. As a consequence a bridging fault cannot appear between any two lines of the design and thus the exact number of the possible short conditions is much smaller.

The number of bridging faults is higher than the number of stuck-at faults. Intuitively, this is supported by equation (2) in comparison to equation (1). Although not all bridging faults are physically feasible, the defect Pareto charts identifies the bridging faults as the most frequent defect in most manufacturing processes [51].

A reliable detection mechanism for bridging faults is the quiescent current analysis similar to the short faults affecting standard cells. In [47] the stuck-at fault model, extended with I_{DDQ} current measuring techniques is denoted as the "pseudo stuck-at fault model". With respect to the covering of the bridging fault models by the stuck-at fault models, the test process using N-detect test vector sets is effective in detecting the majority of the bridging faults [20]. In this context, the reference [55] is relevant: experimental results revealed that even for test vector sets not fully covering all stuck-at faults, but only 99.5% of them, the bridge fault coverage is 95%. Moreover, results presented in reference [24] indicate that the open defects are responsible for only 1.21% of the tested devices which avoided being detected when using a test vector set fully covering all stuck-at defects.

2.3 Delay Fault Model

There are physical faults, as presented in the previous sections, which does not affect the logic behavior of the circuit. However these circumstances remain faulty because of their effect over the quiescent current or the signal propagation latency. The defects which alter the temporal behavior of a design are generally referred to as *delay faults*. The delay faults are related to signals' transitions rather than signals' values as for the previous fault models [51]. They are especially important as the integration technology evolves: the effect of delay faults is more pronounced with the decrease of the feature size.

The *fault size* is characteristic for a delay fault, and it is the amount of time required for the affected node or affected path to propagate an input transition by generating an output transition. Two categories of delay faults are proposed: those concerning gate delays and those describing delays accumulated over signals' paths.

A *gate delay* models the latency incurred at a gate, which "operates slower than it is expected" [51]. The defect is called sometime *transition fault* [23]. A graphical depiction of this situation is presented in Fig. 2.15. The *slow-to-fall* and *slow-to-rise* conditions can be identified with respect to the affected output, Z, although a gate mustn't be affected by both defects simultaneously.

It is evident that in order to identify the defective transition latencies, the gate inputs need to be correspondingly prepared. For example in Fig. 2.15, a value of 1 on the second gate's input would not allow detecting the defective behavior. Another important aspect regarding the delay fault detection can be identified from Fig. 2.15: in order to detect a gate delay fault, the gate's output needs to be settled at a particular value by an *initialization pattern* (at logic 1 for the slow-to-fall defect and at logic 0 for slow-to-rise defect). The second input vector is required to trigger the tested transition and also must assure its effect is propagated to the output.

A path delay fault addresses the transition latency by analyzing it over a larger circuit section. For this model, the transition delay is affected by all the gates included within the analyzed path of the circuit; it is composed by incremental latencies of the component gates, as depicted in Fig. 2.16. It can be observed from the figure the cumulative nature of the delay faults at each gate. The same observation stands, regarding the preparation of the transition phase, which needs to assure the signal propagation throughout the selected route. The detection employs an initialization vector and a transition triggering and propagation vector.

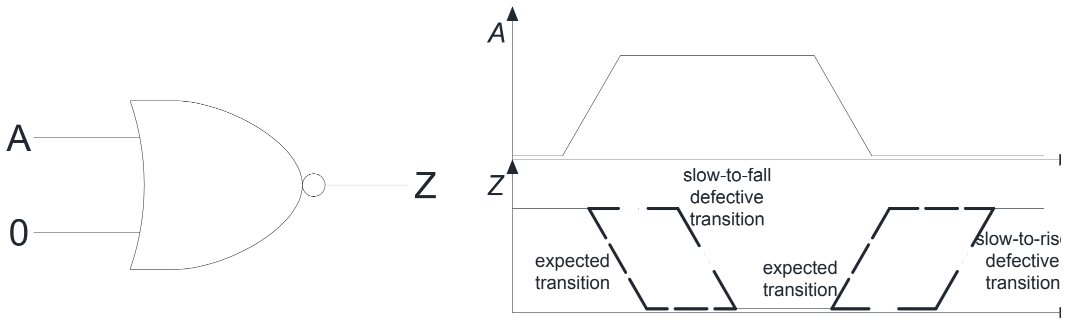

A
0
Z
A
Z
slow-to-fall defective transition
expected transition
expected transition
slow-to-ris defective transition

Figure 2.15 Gate delay fault adapted from [51]

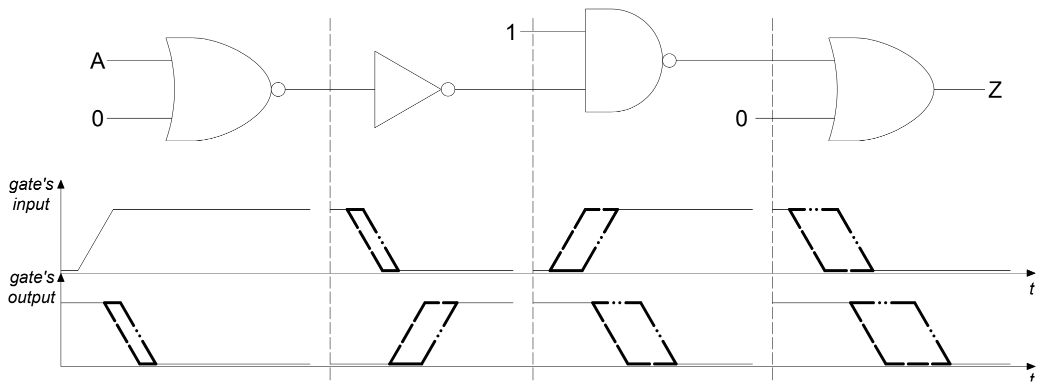


Figure 2.16 Path delay fault adapted from [51]

The detection of a path delay is said to be robust if another possible delay fault doesn't interfere with the initial fault's detection. The condition for a gate whose latency determines all paths containing the respective gate to fails due to the signal transition latency is referred to as a *gross gate delay fault*. It can be assumed that if any path containing the gate fails to perform the transition correctly, than the delay at that particular gate is larger than the clock period [23]. The fault size for the conventional gate delay faults does not necessarily exceed the clock period. The path delay faults are considered to model the temporal defects of a circuit more reliable than the gate delay fault model [23].

A delay fault which manifest only at certain clock frequencies, different from the normal operating frequency, does not provoke circuit failures, being only a potential fault. However, when the delay faults are manifesting at the operational clock frequency, the logic behavior of the circuit is altered as a consequence of the high latency. This is observable when inspecting the sequential modules surrounding combination networks. The transition for the combinational segment is triggered by the clock's first edge, and provided that the delay's fault size, along at least one path, is larger than the clock frequency, the next active edge of the clock will store the incorrect value in the register. The logic faults provoked by delay faults are not guaranteed to be detected by the stuck-at or transistor level fault models, as the field literature suggest.

However there are references marking out the relations between the stuck-at faults coverage and the path delay defects. Such an example is reference [56], in which the robust test vectors for a path delay fault can be obtained from the test vectors detecting stuck-at faults. Similar references investigate the coverage of N-detects stuck-at test vector sets for the delay faults. The N-propagation test pair set is constructed in [57] from the N-detect stuck-at test vector set and the applicability to single testable path delay faults and robust path delay faults is evaluated. The experimental results yielded considerable path delay coverage, thus linking the stuck-at fault model to the path delay fault models.

Chapter 3
Test Engineering

As the security demands are increasing, specialized encryption modules are integrated in more and more systems: ranging from low-power, low-speed Radio Frequency Identification modules up to the high speed disk controllers. Security modules are especially sensitive to faults. As a consequence of their nature, more precisely due to the *diffusion* property, a single bit modified during the execution of a cryptographic algorithm will determine a shuffled result when compared to the expected one, because of the spread of modified bit's effect through the entire state, process driven by encryption's very own operators. In fact, for the case of AES, in [58] is experimentally obtained the mean number of output's modified bits as a result of injecting a single fault into one of the 10 rounds. With the excepting of the last round, by injecting a single fault, the mean number of modified output bits is 64, more precisely: half of the output data block is changed. This behavior was described in order to introduce the need for better protection in the context of cryptosystems. For example, if the encryption module is affected by a fault in a secured encryption-capable disk controller, the chances to recover the stored data, taking into account the above fault behavior, are reduced. Although for some specific applications, security is more important than availability (or even reliability) the majority of cryptographic modules are required to undergo system tests. The motivation for testing an encryption system is driven on one hand by the typical VLSI manufacturing practice and on the other hand by the motivation to assess device's integrity during and after its normal operations.

3.1 VLSI Testing Considerations

An important reference point for the VLSI era is the Moore's law, regarding the scale of integrated circuits which tends to double every 18 month, property which is expected to hold on for at least two decades. Two distinct directions influence the integration process: the increase in circuit's die area and decrease in integrated elements' dimensions (also known as feature size). Taking into account these tendencies the *signal propagation delay* (potential delay faults at high operating frequencies manifest at the logic level) remains the most important factor in limiting the operation speed, even for smaller feature size [59]. Another technological issue refers to the *signal integrity* problem: as the effect of the crosstalk noise prevails in the context of thicker interconnections (the connection lines' section is increased in order to reduce signal propagation delay) [60]. Connected with these issues and amplified by the pace of physical integration is the *power integrity* problem: the progressive reduction of supply voltage and the increase in performance by raising the clock frequency makes the power supply drops, due to connection line's inductance, this being actually a critical issue. Moreover, the process of testing for high current pikes' worst case scenario proves to be challenging [61].

The actuality of test engineering is revealed into the *International Technology Roadmap for Semiconductors*, published by the *Semiconductor Industry*

Association in 2004 [5]. The document contains an updated description of the test process and test equipment "through the year 2010 and beyond" [62]. The *Semiconductor Industry Association* is a global cooperative effort among the manufacturers and suppliers in the field of integrated devices including manufacturers, universities and governments. Its declared objectives are to advance the performance of integration process as well as identify and assess the challenges to be faced as the industry is moving on toward new technologies. Among the challenges still requiring solutions, for designs with feature size less than 45 nm, the report mentions the following [62]:

- *automated test equipment interface* for the devices under test
- proper test methodologies
- reliable device analysis
- failure analysis
- disruptive device technologies

The specific test challenges are summarized in the following issues [62]:

- effective speed testing for designs with higher core frequencies and widespread adoption of GHz input/output protocols
- a growing gap between the existing Design for Testability measures and provisioned devices' complexity
- the impact over design's quality and yield of the limitation of the test process
- testing for signal integrity as well as for possible new fault models
- enhanced methods for diagnostic, reliability and yield
- developing online test methods capable of scaling increasing design complexities

Also the concept of *Design for Testability* and *Design for Manufacturability* will more likely require further refining and modification in order to accommodate for the new testing techniques imposed by nanoscale designs.

The growing interest for defining a roadmap for semiconductor industry led to the updated versions of the *International Technology Roadmap for Semiconductors* in 2005 [63] and 2006 [64]. Among the elements predicted to require new technological solutions the following were added [62]:

- *distributed test,* as a form for relaxing the test process complexity
- improved test for yield predictions
- screening for reliability,

as well as introducing new process-dependent faulty conditions. Possible opportunities, related to the test process are [62]:

- test program automation
- test interface hardware integration within the design flow
- "convergence of test and system reliability solutions"

The mention of *International Technology Roadmap for Semiconductors* documents and their interest in the future test methodology was intended for highlighting the importance offered to the test process by the leading semiconductor developers, as well as the challenges which await new technical solutions from the test engineering domain.

The test challenges for state-of-the-art and future VLSI integrated devices are only a fragment of the possible fault causes, referred to as *failure mechanisms*. A more general taxonomy of the failure mechanisms is presented in [65] and graphically depicted in Fig 15, taking into account a global physical and electrical failure approach.

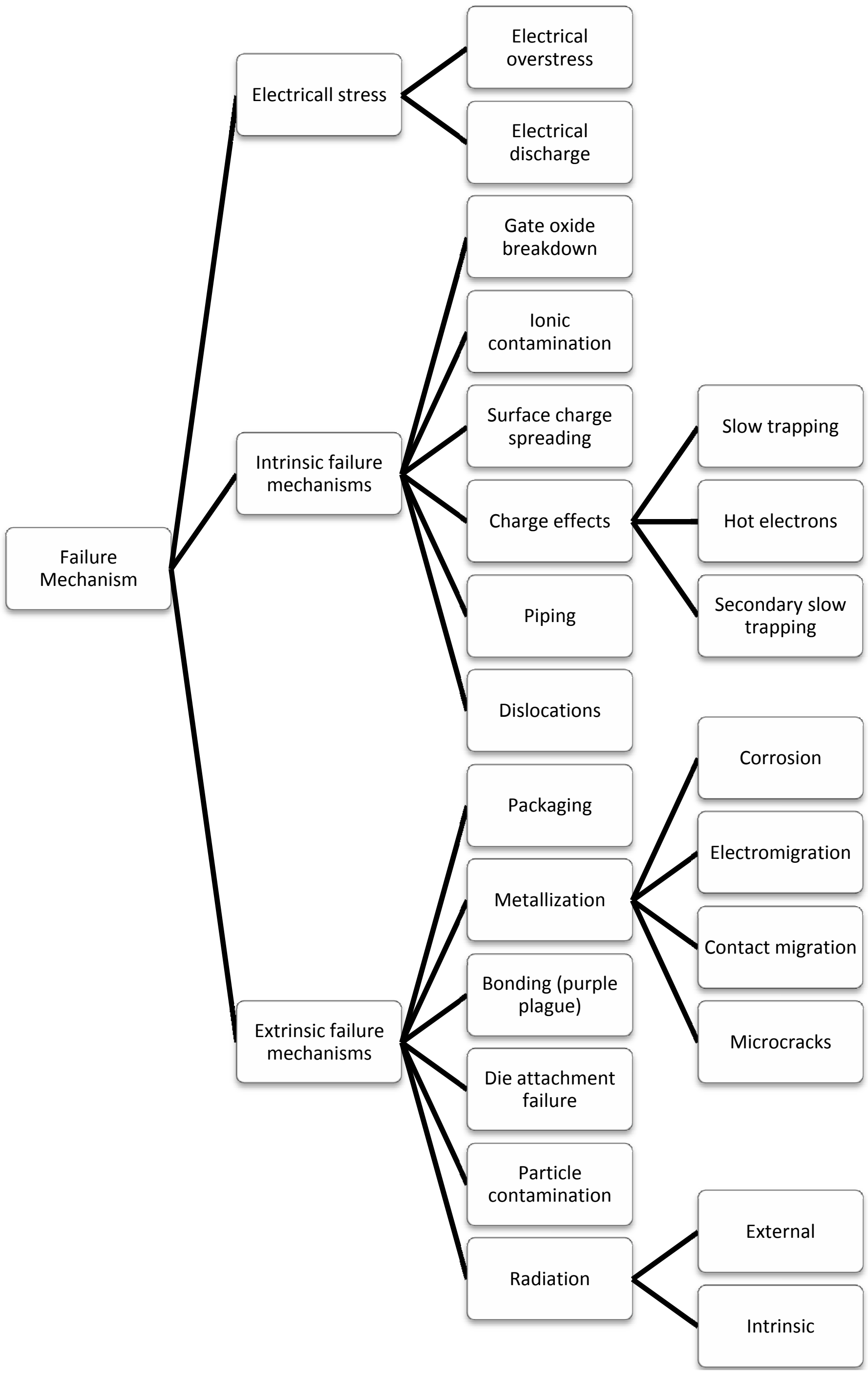


Figure 3.1 Failure mechanism taxonomy adapted from [65]

The test of digital systems encompasses both a logical test and an electrical test. Whereas logical tests assess logical correctness of its behavior (the integrated device performs the intended function with respect to its inputs and outputs), the electrical characterization verify electrical properties of the signals such as the voltage levels, the value of the quiescent current, the influence of the clock signal and the clock skew on the information propagated over the data lines (especially effective, as already presented in the previous chapter at identifying transistor level defects and delay faults), or the drive conditions (the ability of the device to drive other inputs). During our research, we were especially interested in constructing efficient mechanisms for logic testing, pursuing limited electrical characterization of the design which involved semiconductor simulations (the SPICE simulations presented in the previous chapter). Moreover, the electrical testing is usually performed after manufacturing in order to verify that circuit's correct electrical parameters are met with each produced device. On the other hand, logical testing is performed not only after production but also as a mechanism for incoming inspection [21], and more important, during unit's normal operation, at the user's request in order to reassess its correctness.

The testing phase can be performed during module's normal operation, in which case is referred to as an *on-line* or *concurrent* test. However, the test procedure is expected not to interfere with module's internal control and data flow. Some techniques that can be used for concurrent testing are: concurrent error detection codes (parity codes, cyclic redundancy codes) hardware redundancy (such as n-modular redundancy for high reliability systems). Usually, a thoroughly verification requires some test procedures to be performed that would modify the normal control flow and device's state. This type of test is called an *off-line* or *non-*

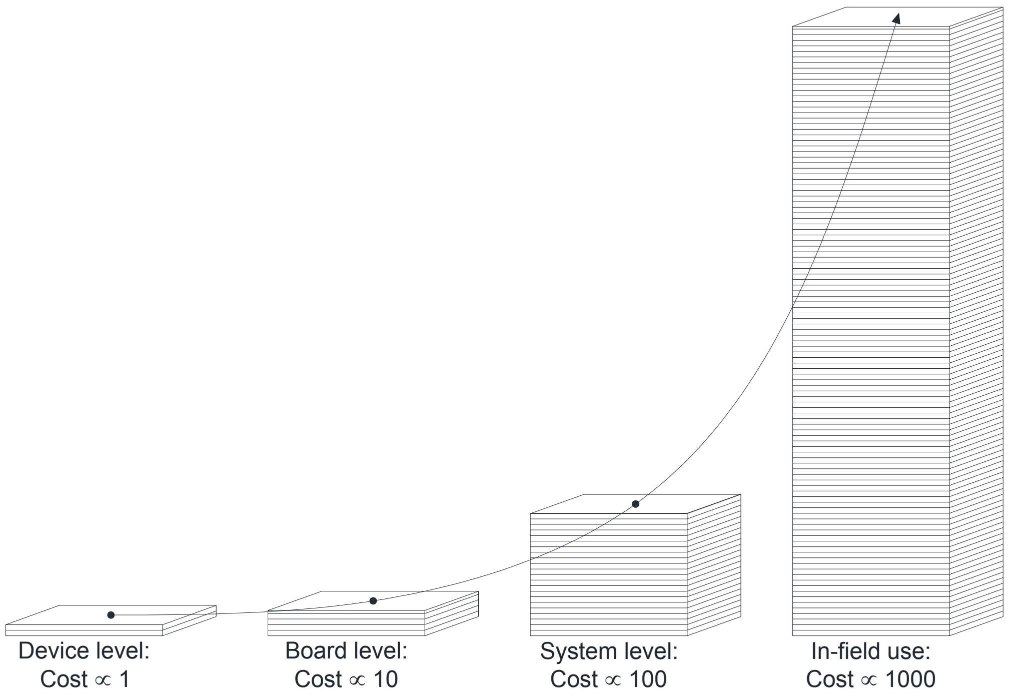


Figure 3.2 Rule of ten for integrated circuit repairing costs adapted from [23]

concurrent test. For their application, the system usually is brought into a so-called test-mode, interrupting system's normal operation.

The difficult problems posed by the advance of integration process are reflected in the process of validating and testing devices' correctness, as already presented. It can be imagined that the challenges of the new integration process are required to manage and resolve the low yields and high production costs. The importance of eliminating the faulty units as soon as possible relates to test economics. A well established standard, familiar to test engineers, illustrate the cost for repairing a defective chip along its lifetime phases: immediately after the component was manufactured, after its integration into a board, after being integrated (within the board) into a system, and finally in system's intended operational environment. This illustration suggests that the test and repair costs respect the *rule of ten* graphically represented in Fig. 3.2. The rule gives an inner-view of the test economics governing the production costs; it states that if the cost to detect and replace a faulty chip immediately after its manufacture is $0.30 then the cost to discover and replace the same faulty module after it was assembled into a board is $3, the cost to detect and replace the very same unit after the board was integrated into the final system is 30$ and the cost to discover and replace the same defective circuit during system's operational life is 300$. A possible explanation for the escalation of the repairing cost is offered in [23] and relates to the increased difficulty to detect the fault (it is much simpler at the production line than inside an operational system) and the repair costs (travel time, reintegration costs). Both a poor interest in test process as well as an over-investment in verification and testing phase can prove unprofitable. The relation between the total cost and test quality can be understood from the graphical representation in Fig. 3.3, as well as the optimum investment in manufacturing and maintenance costs.

Yet another reason for testing is connected with liability [23]. In case a system produces damages (injuries, economical loss) as a result of an integrated circuit's failure, the designer is legally responsible for the consequences, partly or entirely. Typical examples are from the automobile industry (accidents due to defective car control modules) and medical systems (due to a faulty component a curing radiation-based system produces overdose injuries).

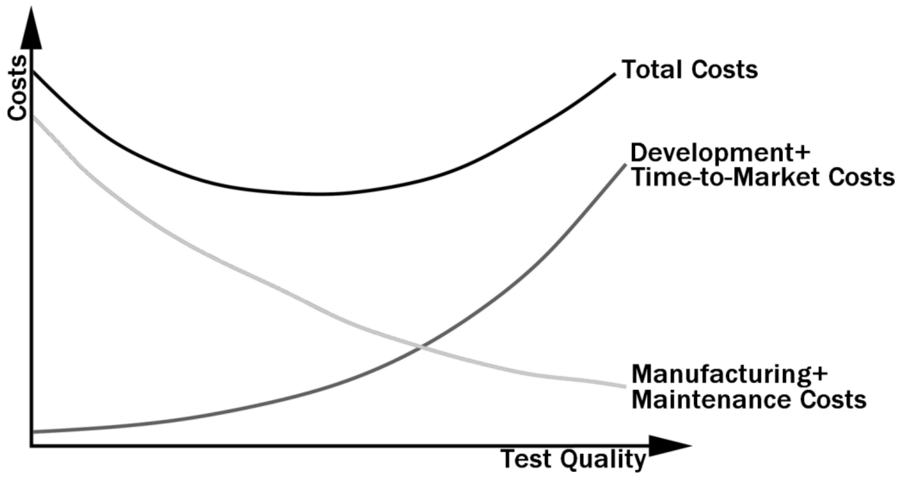


Figure 3.3 Total costs for producing an integrated circuit adapted from [23]

In the context of cryptographic systems, the subject of liability and costs caused by system's failure gained an important perspective. Shipping a defective security-enforcing device (such as a crypto-system) would represent an acute legal problem for its developer. On the other hand, as with any other electronic integrated circuitry, an encryption module can be affected in various ways by faults (the failure mechanisms being presented in Fig. 3.1): especially after extensive use over a large period of time, due to semiconductor's "wearing" the devices will ultimately become useless. Any cryptosystem design is required to take into account the "wear lifespan" [11] of semiconductors and consequently provide mechanisms for validating system's integrity during its operational cycle.

In addition to the possibility of a fault affecting the cryptographic module, due to their security-preserving properties, they are prone to being attacked. Cryptographic devices usually protect valuable information (secret keys such as those stored in Automated Teller Machines), or at least a service requiring a payment (Prepayment Electricity Meters). In both cases, a successful attack on the system, allowing a malicious user to bypass the security restrictions, could have serious financial implications for cryptographic system's user. Literature proves that by injecting faults into a cryptographic unit, enough information can be obtained in order to significantly reduce the efforts for finding the sensitive data [12]. Usually mounting such an attack requires direct access to the cryptographic chip in order to manipulate its inputs and observe the outputs. In fact, for the case of Prepayment Electricity Meters systems, such an attempt is reasonable to imagine.

Moreover, cryptanalysis, settled as a distinct research direction, with the declared objective of strengthening the existing cryptographic algorithms and protocols through its attempts of breaking security algorithms. The cryptanalytic techniques developed over the time represent an important concern for security modules. Techniques such as *side-channel* analysis, requires no intrusive actions but to observe the inherently information offered by the unit such as power, thermal and electromagnetic radiation [7]. Also, the timing of module's operations could offer important information regarding the actual configuration of the key. In order to protect a system against this attacks, special design consideration is required in order to balance the amount of information emanated by chip in each stage of its execution so that the passively gathered data (power, thermal and electromagnetic radiations) to reveal no useful information regarding the inner operations.

Cryptanalysis also refined the fault injection technique: the fault could be forced into the design through various mechanisms [7] such as:

- insert a spike into module's power supply by varying the supply voltage
- inject a glitch by rapidly varying the clock frequency
- subject the unit to extreme condition (near failure situations) by overheating or freezing it, in order for an existing defect to manifest itself
- physical injection of an error through a focused laser beam (or X-ray beam as well)

The serious threat posed by fault injection is that it leaves no trace regarding the tampering actions performed against the device (especially when injecting faults by light radiation). Moreover, the injected fault is a *transient* fault, meaning that at the next iteration of the encryption round, if the timing of the injected radiation is correspondingly adjusted, the defect will not manifest, however its effects already resides into the modified state. This behavior allows an attacker to perform numerous such attempts, each of them bringing him closer to the objective of finding the secret encryption keys. In fact reference [66] describes an

AES attack methodology that can reveal all bits of the key stored in a cryptographic device even when the system is *protected* by concurrent error detection strategies. The solution, although simple, require attacker to have access to the encryption module in order to be able to deliver data to its inputs and receive its outputs. The experiment starts by setting the input block as all-zero bytes. After the very first round (as will be presented in the following chapter, the initial AES rounds implies a simple addition of the encryption key to the data block) attacker injects an error into one bit of the state matrix forcing its value to become 0. If the affected bit's correct value was 1, the attacker would notice an error status, triggered by the detection of a fault by the concurrent verification mechanism; otherwise by correctly performing the entire algorithm, the attacker would also know the correct value for that bit was 0. Repeatedly performing this action, the entire key can be retrieve. An efficient countermeasure for such attacks would be to offer no error status but to randomly craft an output response, in order to mislead the attacker.

In consequence, the encryption modules should include mechanisms for concurrent error detection in order to permanently reassess its integrity. The presented scenario reveal that a successful protection scheme relies on both off-line and on-line test mechanisms. The protection can be performed using external testing modules (be it in-system or external to the system, such as *Automated Test Equipment*) or in-chip, relying on self-test mechanisms.

It is generally accepted that achieving high fault coverage with external testing equipment (*Automated Test Equipment*) in the context of current high speed and complex designs became harder and especially very costly. In consequence the solution is to resort to structured testing mechanisms that allow the testing process to be performed on-chip with at-speed performance. Such solutions are well established *Design for Testability* measures such as *Built-In Self Test* and *Scan Chain*.

3.2 Off-line Testing

The off-line test mechanism, as already mentioned, assumes that the entire digital system or at least part of it, is "taken out of service" [20] in order to allow the test process to perform. As a consequence of test's non-concurrent nature, the implied hardware overhead can be significantly reduced because the off-line strategies are not required to maintain the system in its normal operation mode, such as for the on-line methods. In consequence, requiring smaller area, the off-line test solution can be design to cover all, or as much of the device's area as possible. Not the same can be said about the concurrent measures, for which the associated hardware investment, being significant, is usually reserved for critical modules and units, especially vulnerable to faults.

Off-line test can be used also in conjunction with the on-line test for fault localization [20]. Provided that the concurrent checking mechanism detected an error, the off-line test can be used as a *diagnostic* mechanism, in order to detect the location of the failing module or the extent to which the error spread. In a modular design, composed of discrete units (such as a circuit board) the off-line test strategy is effective at locating the component to be replaced.

Generally, because of the manner the non-concurrent tests are applied, they are more appropriate in detecting defects at a larger set of locations, because of their wide applicability as already discussed. However, the strategy requires the inputs of the system and its state to be controllable. In consequence an off-line test would be triggered in “low-demand periods” [20], in idle cycles or immediately after a system repair [20] to ensure system’s overall correctness.

Typical testing processes, as presented in the previous chapter, rely on constructing test vector sets relevant for a given fault model or set of fault models. The test sets are, as already described, specifically design for maximizing the coverage (for single fault conditions they usually assure complete fault coverage) while minimizing the test application time. Moreover, the specific sequence of test vectors (especially for stuck-open defects and delay faults) required to be presented at the circuit’s inputs demands a mechanism for applying the vectors directly to

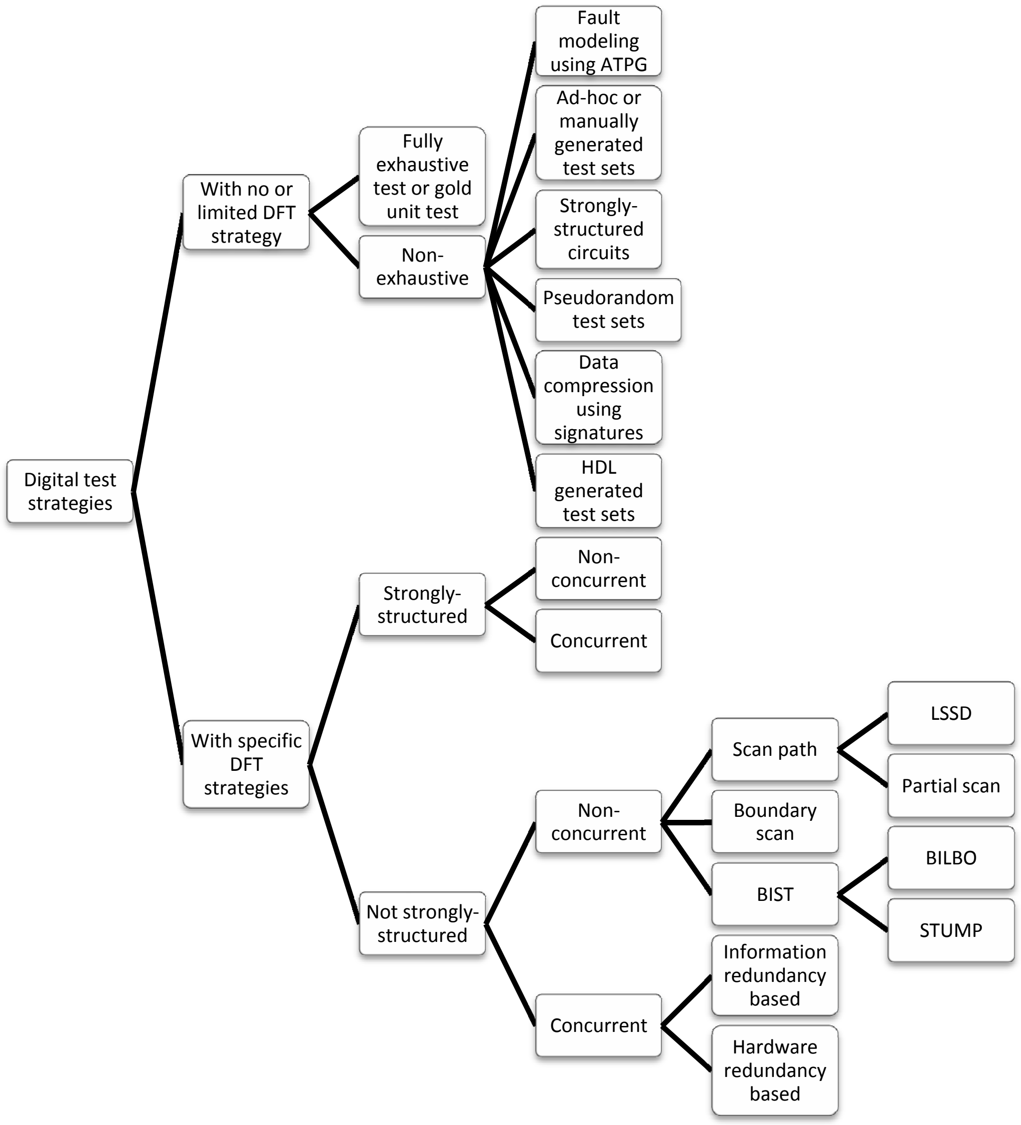

Digital test strategies
With no or limited DFT strategy
Fully exhaustive test or gold unit test
Non-exhaustive
Fault modeling using ATPG
Ad-hoc or manually generated test sets
Strongly-structured circuits
Pseudorandom test sets
Data compression using signatures
HDL generated test sets
With specific DFT strategies
Strongly-structured
Non-concurrent
Concurrent
Not strongly-structured
Non-concurrent
Scan path
LSSD
Partial scan
Boundary scan
BIST
BILBO
STUMP
Concurrent
Information redundancy based
Hardware redundancy based

Figure 3.4 Taxonomy of digital testing mechanisms adapted from [67]

the inputs of the unit. This requirement describes the basic functionality of the Built-In Self Test mechanism as will be further detailed.

A brief description of the digital testing strategies is presented in Fig. 3.4, where the conventional non-concurrent test mechanisms presented are:

- the Built-In Self Test [21], [68]
- the Scan Path approach [21], [68]
- the Boundary Scan method, together with the IEEE 1149.1 standard - Standard Test Access Port and Boundary-Scan Architecture [69]

For the case of strongly structured designs, such as storage modules (ROM, RAM memories) or programmable logic (PLA, FPGA), the non-concurrent test mechanism can intersect with the typical non-structured designs methodologies although they tend to employ specific test strategies [68], [20].

Yet another test strategy relies on verification using external mechanism: either hardware, such as the Automated Test Equipment [21], [23], [68] or software such as software diagnostic methods [70].

3.2.1 Built-In Self Test

The *Built-in Self Test* represents a typical *Design for Testability* measure aiming to insert into the structure of the digital circuits special modules capable of validating their functional correctness and integrity, with the general architecture depicted in Fig. 3.6. It is especially suitable for secure verification of a device, because it reduces the amount of information exchanged with device's environment to a minimum. Our early BIST research with respect to secure AES testable solutions was summarized by [71] in a presentation to the "The Claude Shannon Workshop".

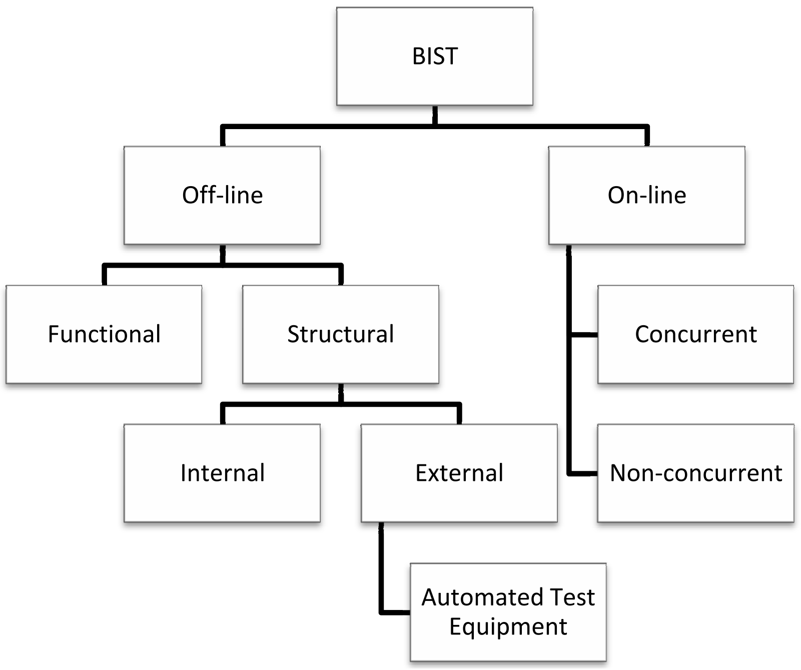


Figure 3.5 Built-In Self-Test taxonomy [20]

According to reference [20] the BIST testing approach can be classified as in Fig. 3.5. The non-concurrent labeling of an on-line test solution (as the figure presents) is rather non-standard as the literature confirms and the classification in Fig. 3.4 reveals. However, the terminology serves the intended description of the test solution as will be presented.

The concurrent on-line BIST performs the test function simultaneous with the circuit's normal operation. The behavior of the test scheme when a fault is encounter varies according to its design and properties:

- for a self-repairing design, the system's functionality is restored to a previous correct state
- in a fail-safe system the error could trigger a specific routine for preparing the unit to enter an erroneous state without compromising the current operation.

The "non-concurrent" online BIST operates in idle cycles of the system. However, if the system is facing a request while operating the test, its normal functionality must be restored as soon as possible. This behavior is usually implemented in a software approach, which, by use of an interrupt system, assures system's responsiveness and fast switching from test to normal operation. As presented in [20] and [22] the test is structured, in this case, as firmware or software routines.

The functional BIST in Fig. 3.5 ensures the system satisfies its design requirements. It is usually based on the design's specifications and is constructed from system's high level description (at the algorithm or RTL level). The structural BIST, on the other hand is concern with assuring that the system preserves its underlying structure. It usually targets the logic structure (at the gate level) because structural testing for lower levels is performed through parametric testing. The structural off-line BIST can be internal, as the solutions we propose, as well as external, when it is performed with Automated Test Equipment.

As already mentioned, the VLSI integration poses significant challenges to the test process, mainly due to the clock's frequency scale up, and increase in integration density. The cost of testing newly produced units using *Automated Test Equipment* is escalating firstly because of ATE's costs and secondly because of the high latency associated with the test application. The cost of such a tester increases with the maximum frequency at which it can execute the test programs and with the number of probes (pins) required for performing tests: in [21] is illustrated a case study for an *Automated Test Equipment* performing at a frequency of 1GHz with a cost per pin of $3000. Also in [21] is mentioned that typically, the *Automated Test Equipments* "remains" behind, in terms of the clock frequency, compared to the devices they are testing, this in turn, being the justification for the higher test process latency.

The on-line BIST class not only preserves characteristics of the on-line test, being able to check device's correctness during its normal operation, but, more important, it can detect transient or intermittent faults affecting the device during its functioning. This BIST strategy also implies some performance degradation due to the added circuitry responsible for integrity validation. On the contrary, off-line BIST has a lesser performance degradation because, as with any other off-line test mechanism, it is performed after device's normal operation was interrupted. Nevertheless, it cannot detect transient and intermittent faults, or, at least not the ones affecting the device during its normal operation, when the off-line test is not performed.

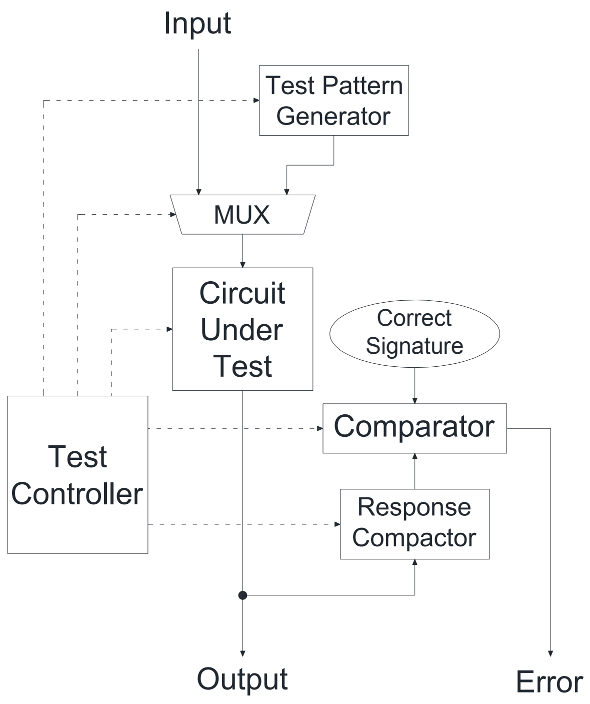


Figure 3.6 Off-line Built-In Self Test design from [72]

That being said, the design in Fig. 3.6 turns out to represent a typical off-line BIST structure. The test approach depicted in the figure is suitable for combinational networks capable of being separated into distinct sub-networks. Whenever a module to be tested contains storage elements, the test process is better managed if the system is partitioned into combinational sub-modules surrounded and linked by registers. This observation eliminates the elaborate issue of unknown values and unknown states [62]. The main components of an off-line BIST are presented in the followings.

Test controller supervises the test process. It receives control from device’s control unit, after the testing mode was installed. It enables the generation of stimulus vectors and directs them to the inputs of the module to be tested. Concurrently, it drives the compaction mechanism in accepting the responses generated at circuit’s outputs and pack them into a single signature. After the entire stimulation and signature compression was performed, the obtained signature is compared with the correct *golden value* in order to assess module’s integrity. Finally, the control is returned to the system’s control unit. The BIST method can be applied hierarchically to the system level, board level, chip level, and eventually module’s under test level, as described in [21], with each test controller being responsible for validating elements at its level and below. This solution might also benefit from test reuse (test vector sets or compaction schemes reuse).

Test vector generator has the role of stimulating the *Circuit Under Test* in order for the possible faults to manifest through erroneous circuit’s responses. The test generator can be implemented according to [21] as:

- Linear Feedback Shift Registers (LFSR): which is also the “preferred” solution both in academic and industrial designs

- Binary Counters
- Storage elements: ROMs containing a reduced test set specially crafted by an Automated Test Pattern Generator, or a combination between ROMs and LFSR in order to ameliorate the ROM's increased area requirements
- Cellular Automata
- Weighted generation logic (typical use for weighted LFSR) [68]

Dependent on the properties of the sequence produced by the test vector generation unit, the test is said to be exhaustive, pseudo-random or pseudo-exhaustive [21]. It is important to assure that generated test vector set is able to stimulate at least every single stuck-at-fault. Only an exhaustive approach would assure all single and multiple faults are stimulated.

When it comes to choosing a particular test pattern generator, it is important to analyze the costs of including it into design, both in terms of performance and area requirements, and evaluate the coverage in terms of detectable faults. Usually coverage is expressed in terms of the single stuck-at fault model. Moreover, the detection rate usually doesn't take into consideration the multiple faults conditions (the number of multiple stuck-at faults grow exponentially dependent on the number of single stuck-at faults as already discussed).

A comparison between the test pattern generators is presented in [68], where the test generation techniques presented in Table 3.1 were evaluated. It reveals the LFSR structure demand the most reduced circuit area. Moreover LFSR is able to offer 100% single stuck-at fault coverage because a careful selection of its generator polynomial (also referred to as *characteristic polynomial*) provides a maximal test set sequence (which can further be enhanced with generating also the all-zero vector) for generating exhaustively all Circuit's under Test inputs sequences. For obvious reasons this approach is appropriate for modules with reduced number of inputs: the test process latency depends exponentially on the number of inputs.

Test pattern generation	**Number of flip-flops**	**Number of EXOR gates**	**Number of gates**	**Number of gate inputs and outputs**
LFSR	8	3	8	33
Weighted LFSR	12	3	15	51
Cellular Automata (CA)	8	9	8	51
Counter	8	8	15	69
Finite State Machine (FSM)	9	9	19	94

Table 3.1 Comparison of test pattern generation mechanisms adapted from [68]

Test compactor: is expected to identify as many faulty conditions as possible, being known that the detection of all erroneous situations is almost impossible to obtain in the context of responses compaction. The detection relies on comparing the obtained signature with the correct one. For a detected error condition, the gathered signature is modified as a consequence of the erroneous responses generated by the faulty *Circuit Under Test*. However, the reason a compaction scheme cannot offer 100% detection is due to the *aliasing probability*. The *aliasing* is defined in this context as the situation in which a faulty module being stimulated with a set of test vectors generates at its output a set of response vectors compacted by the signature analyzer into a correct signature. In [21] are presented some of the most notable compaction mechanisms:

- Parity checking: a possible implementation would verify that the obtained signature has the expected parity. More complex parity detection mechanism can be devise as well.
- One counting: verify that the number of logic 1s in the response vector set is the expected one
- Transition counting: similar to the previous one but counts the transitions in the response stream
- Single Input Shift Register (SISR) or Multiple Input Shift Register (MISR) based on LFSR structures

There are also solutions for compaction using cellular automata. However, the LFSR-based approach, due to its reduced hardware overhead and low aliasing probability is preferred. It can be demonstrated that the aliasing depends on the length of the LFSR register: the larger the LFSR, the smaller the aliasing occurring.

The *gold signature* is computed with respect to the correct device. It is usually obtained through simulations and is hardcoded into the detection scheme. The comparator is only activated at the end of stimulation and compaction process, communicating module's integrity status to the control unit.

Upon detecting a defective situation, the system's control unit can be designed to halt system operation, set an error indicator, and wait for authorized personnel to evaluate system's condition. In the context of cryptographic units however, detected faults are handled differently, with emphasis on preserving the security of the sensitive data.

In the following, the security requirements for a cryptographic implementation will be presented, as they were prescribed by the Federal Information Processing Standards [73]. They can be summarized in the following rules:

- The system should never allow access to plaintext, key material or partially processed data
- The system should detect and indicate as soon as possible any system failure
- The system should detect immediately any unauthorized accesses and consequently erase all key material, and sensitive data while hinder normal operation
- The system should periodically reassess its reliability.

Once again it can be observed that the first three rules favor an on-line detection mechanism while the last recommendation best suits an off-line test solution.

3.2.2 Linear Feedback Shift Register

It is a typical structure employed both for BIST test pattern generation as well as for response compaction. Its generated sequence is pseudo-random [23]. A LFSR structure is generated by a characteristic polynomial. There are two distinct configurations that can be constructed for a particular generator polynomial: *internal* and *external* LFSR graphically described in Fig. 3.7 and Fig. 3.8 respectively.

Both LFSRs in Fig. 3.7 and Fig. 3.8 were generated using the x^4+x+1 primitive polynomial. The operation of internal LFSR is performed faster because

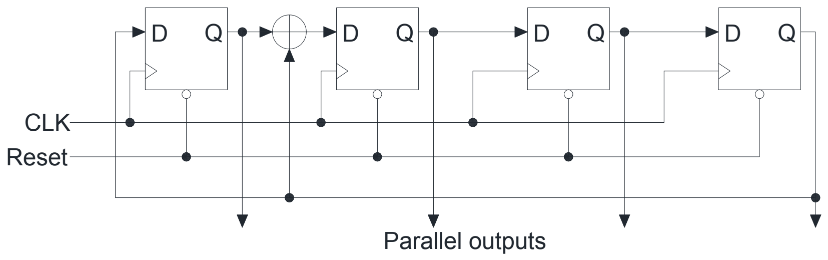


Figure 3.7 Internal Linear Feedback Shift Register architecture

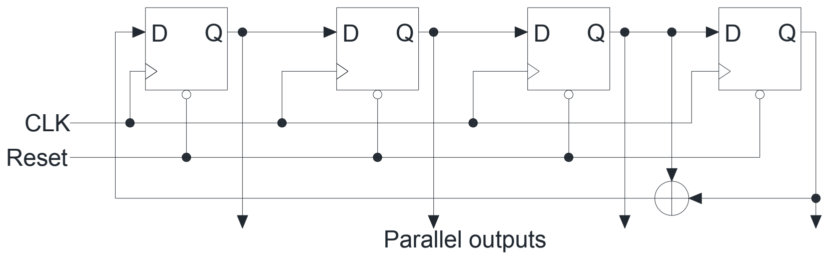


Figure 3.8 External Linear Feedback Shift Register architecture

any EXOR gate is placed between two storage elements, as opposed to the external solution in which the gates are placed on the feedback connection (from the last stage up to the input of the first stage). For the examples on Fig. 3.7 and Fig. 3.8 this observation is not relevant due to the reduced number of terms in the characteristic polynomial; however for a LFSR with a significant number of outputs (64, 128) and inner terms of the generator polynomial, the observation is pertinent and can save valuable time. Moreover, it can be shown that both structures share the same pattern generation properties.

The internal operation of LFSR in Fig. 3.7 can be interpreted as multiplication by monomial *x*, performed in the finite field constructed from LFSR's associated characteristic polynomial. That means the resulting value after one clock cycle execution will be the remainder obtained by dividing LFSR's previous value multiplied by *x*, with LFSR's characteristic polynomial. As a consequence, the LFSR structure can be analyzed and evaluated based on its characteristic polynomial. In its most generic form, the characteristic polynomial, for an n-stage LFSR is described in equation (3):

$$g(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \cdots + c_2x^2 + c_1x + 1,$$
$$c_i \in \{0,1\}, 0 < i < n \tag{3}$$

For each non-negative coefficient c_i an EXOR gate is inserted correspondingly in the LFSR structure (internal or external). The LFSR's behavior, with respect to the generated vector set, is dependent on the properties of the generator polynomial. It can be shown that the maximal sequence (or maximal periodicity) for the output set is obtained only for primitive characteristic polynomials [74].

The content of LFSR can be interpreted as a polynomial of degree *n-1* (this approach was already employed when detailing the functioning of the internal

LFSR). Considering the current content of an LFSR with characteristic polynomial (3), expressed by polynomial $a_i(x)$ as in (4), the content of the register after one clock cycle, namely $a_{i+1}(x)$, is described by equation (5):

$$a_i(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots a_2x^2 + a_1x + a_0 \tag{4}$$

$$\begin{aligned} a_{i+1}(x) = (a_{n-2} + a_{n-1} \cdot c_{n-1})x^{n-1} + \cdots + (a_1 + a_{n-1} \cdot c_2)x^2 + \\ (a_0 + a_{n-1} \cdot c_1)x + a_{n-1} \cdot c_0 \end{aligned} \tag{5}$$

The LFSR generates its periodic output sequence one pattern every clock cycle. The update mechanism, knowing that advancing the LFSR's content can be interpreted as multiplication in a finite field, is described in equation (6), with the LFSR's content after *i*th cycle being referred to by the S_i variable, and LFSR's initial state by S_0. S_i can be computed iteratively using (5).

$$S_i = (S_0x^i) \bmod g(x) \tag{6}$$

A graphical representation of the state transition for the internal LFSR in Fig. 3.7 is represented in Fig. 3.9. Firstly it is a maximal sequence of 2^4-1 elements due to the characteristic polynomial being primitive, the one missing element up to the maximum of 16 (for a 4 stage register) being the all-zero state. Secondly it can be understood that in order to properly generate the maximal sequence, the LFSR must first be initialized with any vector different than all-zero vector.

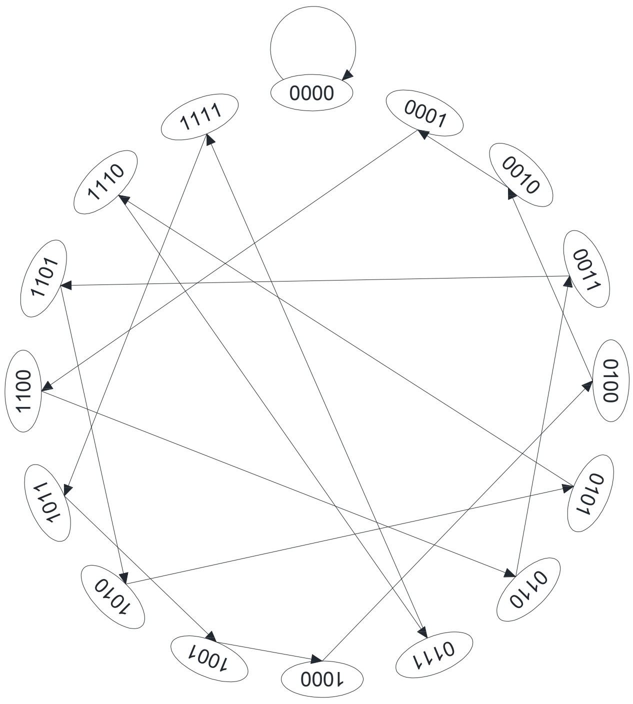


Figure 3.9 State transition diagram for the internal LFSR in Fig. 3.7

3.2.3 Signature Registers

Signature analysis is extensively used as a test result compaction method. It is based on the theory of cyclic redundancy checking as presented in [75]. A Single Input Signature Register (SISR) is built upon a typical LFSR structure. The architecture for a SISR constructed from the internal LFSR in Fig. 3.7 is presented in Fig. 3.10. Recalling from the LFSR description, the progress of an internal LFSR content, after each clock cycle, can be more easily understood as polynomial multiplication in the finite field of its characteristic polynomial. However, a LFSR structure by itself is a close system having no other input line than the clock. In order to use it as a compaction element, an additional EXOR gate is added at register's least significant stage, for adding the value of feedback connection with the input line. The input stream for compaction, advances each clock cycle. Although a LFSR cannot progress when loaded with all-zero patterns, the SISR is usually cleared before its use in order to correctly evaluate the resulted signature.

Starting from the all-zero value, the structure will capture the next bit each clock cycle. However, because the content of the register can be interpreted as a polynomial, the mathematical expression representing the state of the SISR at the next clock cycle is obtained by multiplying the mathematical representation of the current state by *x*, adding the current bit from the input stream to the result, and performing the modulo operation with respect to SISR's characteristic polynomial.

The SISR functioning can be described mathematical as in equations (7). We considered an approach similar to that used for LFSR, considering the current SISR's content to be described at the clock cycle *i* by the polynomial $a_i(x)$, and the input stream being referred to by the r_i variable, also at moment *i*.

$$\begin{aligned}
&a_0(x) = 0\\
&a_1(x) = r_0\\
&a_2(x) = (r_1 + r_0 x) \bmod g(x)\\
&\vdots\\
&a_i(x) = (r_{i-1} + a_{i-1}(x) \cdot x) \bmod g(x)
\end{aligned} \tag{7}$$

Because of the mathematical properties of modulo operator, $a_i(x)$ can be rewrite as in equation (8):

$$a_i(x) = (r_{i-1} + r_{i-2}x + r_{i-3}x^2 + \cdots + r_1 x^{i-2} + r_0 x^{i-1}) \bmod g(x) \tag{8}$$

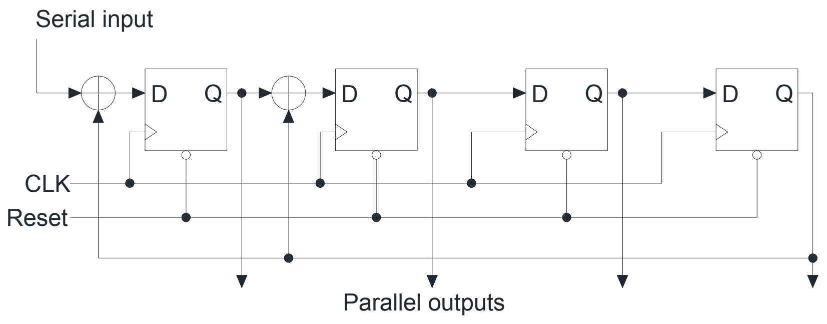


Figure 3.10 Single Input Signature Register

In other words, when initialized with the all-zero configuration, the SISR interprets the input stream as a polynomial delivered one coefficient every clock cycle, and performs division of the received polynomial by SISR's characteristic polynomial, with the remainder being stored as the SISR content. The remainder obtained from division is the final signature. This operational principle reveals an elegant compaction scheme. In order to obtain a fixed-size signature from a possibly very long stream, the SISR would contain the remainder of division (the quotient is variable-sized). Division is used because of its efficient implementation in terms of only shifts registers and EXOR gates (binary multiplication in a finite field has a higher complexity [76]).

With respect to the error detection capability, suppose the input polynomial *R(x)* described in (9) is applied to SISR's input. Correspondingly, after *R(x)*, have been serially entered into SISR, the obtained signature is *m(x)*, with $m(x) \equiv a_i(x)$. Considering the *R(x)* polynomial to be the sequence of correct responses generated by a Circuit Under Test (at most one line of the CUT's outputs) we define the "gold signature" to be *m(x)*. When dealing with possible faulty units, the test responses are expected to differ from the correct ones, this being expressed in terms of an error polynomial, *E(x)*, to be added to the *R(x)* polynomial.

$$R(x) = r_{i-1} + r_{i-2}x + r_{i-3}x^2 + \cdots + r_1 x^{i-2} + r_0 x^{i-1} \tag{9}$$

According to equation (8) the new signature will be different. As pointed out by (10), the difference between the correct and the faulty signature, appears due to the *e(x)* component, which confirms module's defectiveness.

$$\begin{aligned} a_i(x) &= (P(x) + E(x)) \bmod g(x) \\ &= P(x) \bmod g(x) + E(x) \bmod g(x) \\ &= m(x) + e(x) \end{aligned} \tag{10}$$

It is possible now to express the aliasing problem, mathematically, as the condition that determines *e(x)* to be 0, while *E(x)* is non-zero. In other words, SISR's characteristic polynomial divides the error polynomial. In the following, the aliasing probability will be detailed. Considering an input sequence of *m* bits in length, that is to be compacted by an SISR with *n* stages ($n<m$), each of the 2^m possible input streams will be compacted into one of the 2^n signatures. According to SISR's linearity all possible input streams are divided into 2^n classes of 2^{m-n} input stream configurations to which the same signature corresponds. Among the 2^m input configurations, only one will correspond to correct circuit all other 2^m-1 corresponding to erroneous circuits. However, there are $2^{m-n}-1$ other input configuration that will be compacted into the same signature. And thus the aliasing probability is computed as in (11):

$$P_{alias} = \frac{2^{m-n} - 1}{2^m - 1} \tag{11}$$

When *m* is much larger than *n*, the above probability can be fairly approximated to 2^{-n}, this being the reason for the aliasing probability to be strongly coupled with the number of stages for the SISR.

It must be noted that the computed probability doesn't take into account the structure of the SISR, or the architecture of the *Circuit Under Test*. It might turn out that not all of the 2^m-1 erroneous response can actually be produced by a faulty device, especially when considering permanent faults.

With respect to the two different SISR designs (corresponding to internal and external LFSRs), in [77] are presented methods for converting between the signature of the external solution into the signature generated by the internal one when the two share the same characteristic polynomial. In consequence both internal and external SISR provide the same detection capability and aliasing probability.

When trying to use SISR for *Circuit Under Test* with multiple outputs, one possible solution would be to provide a dedicated SISR for each output line. However the complexity and especially the area requirement are multiplied by the number of outputs. The same observation stands for the other single output compaction methods like one-counting and transition counting.

A more efficient approach would be to use Multiple Input Signature Registers (MISR) as a generalization of SISRs. MISR drives the concept of adding the input stream to the signature further: it constructs the signature by concurrently analyzing a number of streams equal to its number of stages. The input streams are added in the same way the single input was added for SISR by using EXOR gates for implementing modulo-2 addition.

The architecture of a MISR based on the same LFSR of Fig. 3.7 is depicted in Fig. 3.11. Each stage has an additional EXOR gate to allow the input stream corresponding to that stage to be compacted. When analyzing the MISR structure, one important observation is that a MISR with *m* stages can be interpreted as an *m*-stages SISR [30]. Considering the *m* input streams described by the polynomials $R_i(x)$, the equivalent SISR input polynomial is expressed in (12):

$$R(x) = R_0(x) + R_1(x)\cdot x + \cdots + R_{m-2}(x)\cdot x^{m-2} + R_{m-1}(x)\cdot x^{m-1},\quad 0 \le i < m \tag{12}$$

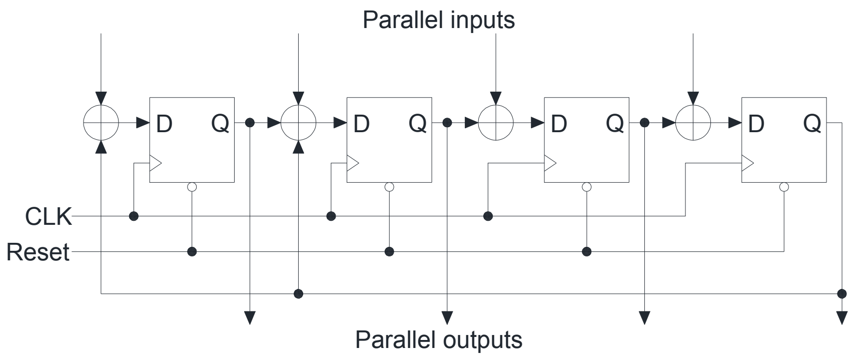


Figure 3.11 Multiple Input Signature Register

Another important aspect for MISR is the fact that it behaves very similar to SISR with respect to the aliasing probability. And this turns out to be an important advantage: being able to save a significant amount of chip's area without degrading the error detection accuracy.

3.2.4 Built-In Logic Block Observer

We have presented so far methods for test set generation and response compaction, in a typical BIST structure. A particular *Circuit Under Test*, be it a sub-module of an architecture or an entire unit, requires specific design adaptation in order for the BIST strategy in Fig. 3.6 to be added. The next step toward easy integration of BIST into a typical digital design is to imagine a solution that can be customized for a regular design, with smallest effort and minimum preconditions needed to be satisfied. One such BIST solution is the Built-In Logic Block Observer (BILBO) technique [78].

The method is applicable for designs capable of being partitioned into a network of combinational logic modules interlinked by storage elements. One can also observe that for some combinational structures, a much simpler testing solution would be to insert an additional register in order to allow BILBO procedure to operate the components rather than trying to construct a test vector set to collectively cover all components.

BILBO alters each register by supplementing them with additional functionality. We observed during the previous sections that LFSR and MISR are effective mechanism for generating the stimuli vector set and for response analysis respectively. However they required to be connected to the *Circuit Under Test* inputs and outputs respectively, similar to the solution in Fig. 3.6. BILBO's unique design goal is to transform, a regular register into a LFSR or MISR, whose functionality depends on two command lines. Additionally to the pattern generation and signature compaction modes and its normal parallel load operation, a BILBO register have a forth operation mode similar in functionality to the scan chain method, for shifting data serially.

A BILBO structure is more easily constructed from an external LFSR for characteristic polynomials with reduced number of inner terms (in order to avoid including 2 EXOR gates on the input logic of each stage), which due to its resemblance and possibility to be converted into an internal one, can successfully replace the internal LFSR. Among the many implementation solutions for a BILBO register, we adopted the one presented in [79] and described in Fig. 3.12. The BILBO register in Fig. 3.12 is constructed based on the external LFSR in Fig. 3.8.

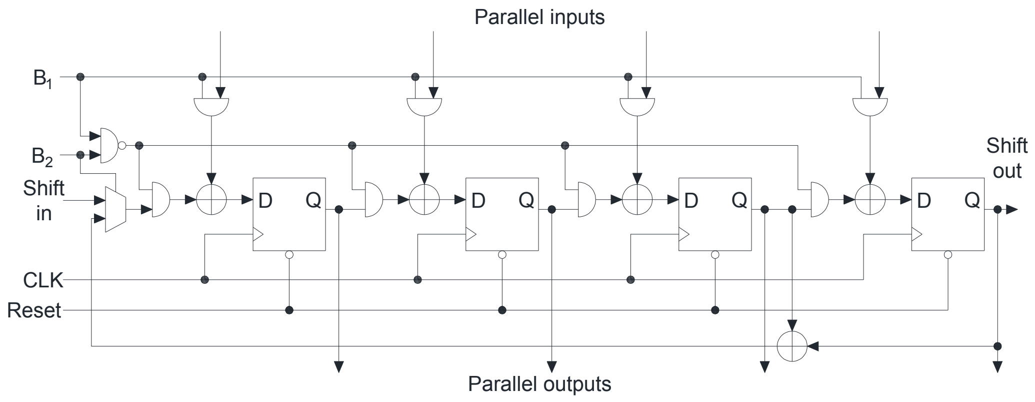


Figure 3.12 Built-In Logic Block Observer Register adapted from [79]

Control Lines		BILBO Behaviour
B1	**B2**	
0	0	LFSR-based pattern generator
0	1	Shift register, capturing in the stream on Shift In input
1	0	MISR mode, based on the same LFSR structure
1	1	Typical parallel buffer register (parallel input)

Table 3.2 BILBO register modes of operation adapted from [79]

The register will operate according to the description in Table 3.2, dependent on the two control lines B_1 and B_2.

BILBO technique can be used also as a scan chain approach provided that all BILBO registers in the design are connected through their *Shift In* and *Shift Out* lines correspondingly. However, it remains to be discussed the mechanism for signature comparison. A possible solution would be to hardcode the correct expected signature into each BILBO register such that at the end of stimulation, a simple network of *AND* and *OR* gates to evaluate signatures equality. In this scenario, however, the BILBO register can detect errors only for the combinational structure it was design to test. For example, suppose that to the input of the BILBO register is connected the output of a multiplexer, selecting the result of two different modules. In this case, it is more effectively in term of time and test length to treat each module separately, and test them at different intervals. Although it remains to be solved the testing process for the multiplexer, this approach is more effective than treating all three components as a whole and deriving test vectors for the complex construct. However, in this case, the gold signature cannot be hardcoded anymore. Instead, they are transmitted to the local test controller for test modules' integrity validation.

Discussion is also required concerning the length of the test sequence and the mechanisms to control it when using a BILBO approach. A possible solution would be to perform an exhaustive test procedure for each combinational node in which the design was partitioned. Another solution would be to hardcode into each BILBO's register LFSR mode the required length of the test sequence for the combinational node connected on its outputs, i.e. whenever the required coverage is reached, the LFSR will automatically reset itself to the sequence's initial vector. However, in this case the BILBO register connected on CUT's outputs will also require adaptation of its MISR mode for the new test sequence.

The testing process using BILBO strategy is described in Fig. 3.13. The test process comprise of two stages: in Fig. 3.13 the combinational networks A and C are first tested, that is the BILBO registers on their inputs, configured as LFSRs, generate the test sets while the BILBO registers on their outputs, configured as signature analyzers, verify circuits' integrity. In the second phase, the BILBO registers are reconfigured so that the combinational units B and D to be effectively tested. Moreover, as reference [79] presents, the entire process could be coordinated in a scan chain fashion, with all BILBO registers being serially linked. In this context, the process would require two scan procedures, extremely expensive in terms of test latency: a scan-out for retrieving the signature and a scan-in for initializing the BILBO registers. When shifting out the signatures, control unit can be concurrently verifying their correctness. Proper initialization can also be performed by a dedicated reset circuitry with all-zero for the MISRs and a specific initialization pattern for the LFSRs. The cost of BILBO's design is non-negligible as stated in [79]. The BILBO approach tends to double the size of the typical registers.

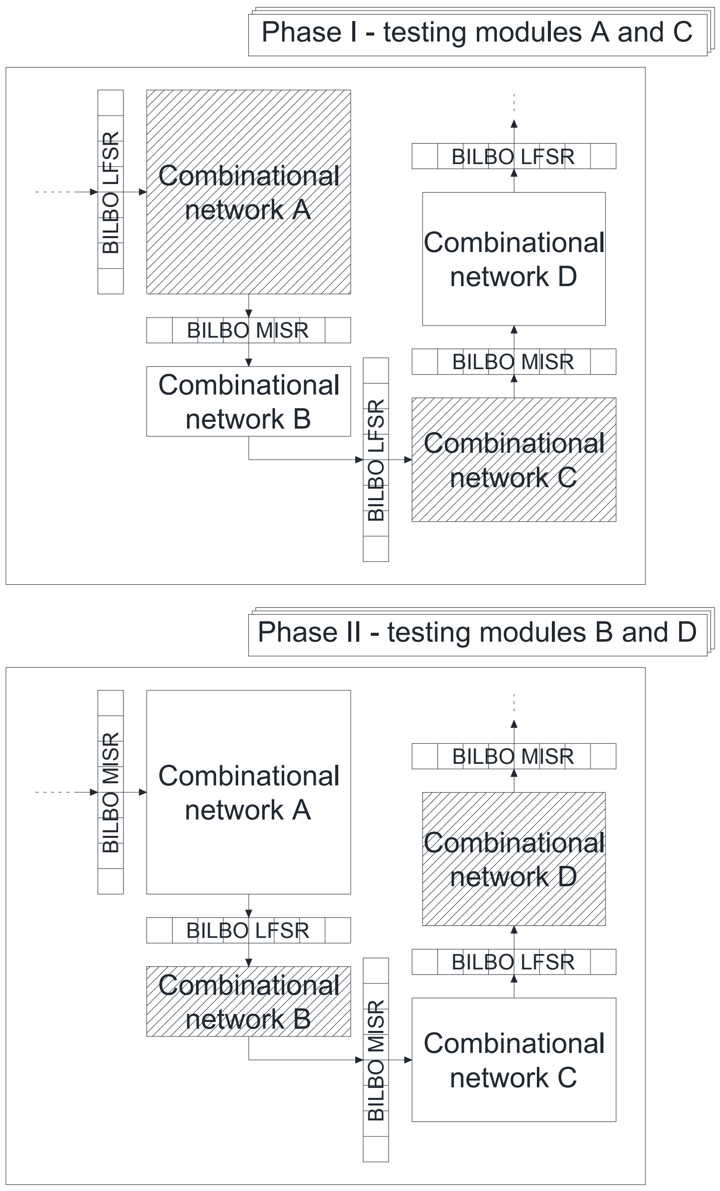


Figure 3.13 BILBO test strategy adapted from [68]

A particular adaptation of the BILBO technique, called Concurrent BILBO (CBILBO), increase the performance significantly, by allowing the test generation and signature compaction operations to be performed concurrently [62]. It achieves this by incorporating in each storage element two registers: one with a LFSR function and another one with a MISR behavior. The two operations (test generation and response compaction) are simultaneously selected by the same control lines configuration. The other operations of a CBILBO register are serially shift and parallel load.

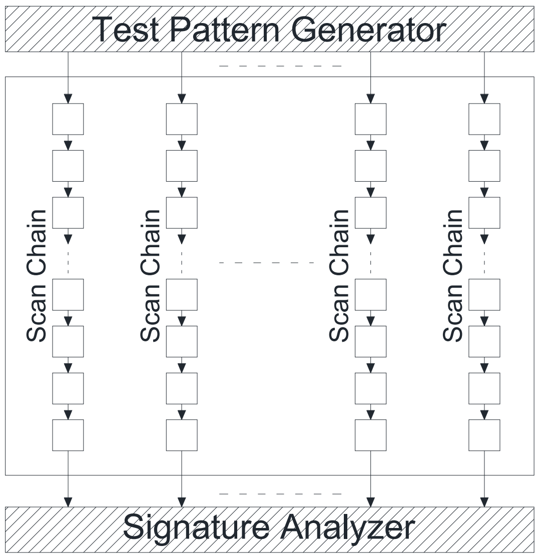


Figure 3.14 Self-testing using MISR and parallel SRSG architecture adapted from [62]

A particular adaptation of the BILBO technique, called Concurrent BILBO (CBILBO), increase the performance significantly, by allowing the test generation and signature compaction operations to be performed concurrently [62]. It achieves this by incorporating in each storage element two registers: one with a LFSR function and another one with a MISR behavior. The two operations (test generation and response compaction) are simultaneously selected by the same control lines configuration. The other operations of a CBILBO register are serially shift and parallel load.

Yet another similar approach is the so-called Self-Testing Using MISR and Parallel SRSG (STUMPS) architecture [62]. It is a BIST adaptation to be used in architectures designed in a scan-chain manner. In other words, the design is partitioned into easily testable scan chains, but in order to avoid using external modules or ATEs to generate test vectors and analyze response, the architecture include the Shift Register Sequence Generator (SRSG) structure and a MISR unit, servicing all the chains as described in Fig. 3.14.

3.3 On-line Testing

The previous chapter focused on defining the behavior of a digital system in the presence of *solid faults* [16]. However, as already presented in Chapter 1, digital circuits can be affected by *intermittent faults*. Field literature unanimously appreciate that intermittent faults, due to their temporarily nature, are not addressed by the test process targeting conventional fault models [80], [81], [82]. In fact, their presence can only be indicated by a permanent monitoring of the circuit's correct behavior. The test process directed toward stuck-at, stuck-on, stuck-open fault models is usually implemented in a *non-concurrent* manner, implying the circuit's normal operation is hindered in order for the prepared test activity to perform. Among the conventional non-concurrent, or *off-line*, test strategies, as already mentioned one can name the Built-In Self Test, the Scan

Chain approach and external testing relying on Automated Test Equipment. The test process, in correlation with the off-line testing techniques, implies the stimulation of the *Circuit Under Test* by applying a predetermined set of test vectors, acquiring the response vectors and comparing them to the expected correct response. As already discussed, the off-line test is usually applied after the circuit is manufactured, as part of the more thoroughly manufacturing test, and is also used periodically during system's lifetime, in maintenance tests.

The on-line test techniques is conventionally used in order to assure reliability in critical systems such as railway control, satellites, avionics, telecommunications, automotive systems, medical devices to name only a few [83]. As already presented throughout this work and observed in the field literature, the cryptosystems' vulnerability toward attacks and faults require a consistent on-line test discipline.

One reason for the lag between the development of on-line and off-line testing techniques concerns the lack of CAD tools support for automatic on-line test insertion, compared to the support offered for Design for Testability measures such as Scan Chain and off-line BIST. A possible reason relates to the design complexity an on-line solution amounts, making a difficult task for automated development tools to evaluate a good trade-off for the potential solutions.

The permanent validation of circuit integrity is referred to as *on-line testing*, *concurrent checking* or *concurrent error detection*. The main concern for the on-line test is to detect any modification in circuit's behavior as soon as possible. Although such a deviation can be the result of either a permanent or an intermittent fault affecting the device, the situation is more critical for intermittent faults, because they affect system's dependability: although the system fails because of the fault, the proximal off-line test will not detect this failure as its effect disappeared. In summary, the two characteristics of the transient faults which make this type of defects especially hard to detect are:

- The limited manifestation duration
- The unpredictability of their occurrence

The permanent faults are also unpredictable with respect to their apparition; however, concerning the off-line test process the moment of their emergence is not relevant, i.e. after the off-line test passed, a clear conclusion is drawn whether the fault was manifesting (so already emerged) or not. It is not the same with intermittent faults with respect to the on-line detection: because the goal of the

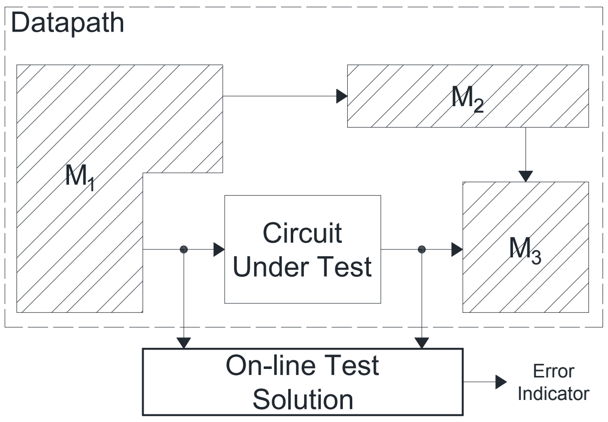


Figure 3.15 On-line test mechanism strategy

concurrent error detection mechanism is to determine the defective condition as soon as possible, the precise manifestation of the fault is crucial. Moreover, for the permanent faults their effect remains constant for a sufficiently large period of time, allowing the off-line test mechanism to identify them. The intermittent faults affecting a combinational network can have no effect over the correct behavior of the circuit if the latency for the signals propagated through an affected node toward the nearest storage element is greater than the remaining duration of the clock period. However, in a worst case scenario the apparition and duration of the intermittent defect determine a system failure.

The effect of intermittent fault can be described in terms of the gate-level or transistor-level fault models. However, the detection solution resorts to verifying circuit's correct behavior during its normal operation. Moreover, because of this requirement, a typical test sequence as constructed in order to detect faults at various design abstraction, cannot be used. The verification structure is expected to determine the correctness while the CUT is operational. As a consequence for each output response the correctness needs to be verified, by inspecting the CUT's inputs and outputs: the integrity of the circuit is evaluated and signaled to the control unit or to the system's user through an error indicator line as Fig. 3.15 depicts. According to Fig. 3.15 the CUT is a component element of the system's architecture; and it is individually protected by the on-line test mechanism, while performing its operations as part of the architecture's datapath.

The concurrent checking mechanism, as described in reference [84], conventionally relies on *hardware redundancy* or *time redundancy* principles. The typical hardware redundancy measures consist of predicting one property of the CUT's output and verifying that the respective property stands for each CUT output. In [85] the hardware redundancy implementation is dissected into a predictor and a checker similar to the structure in Fig. 3.16.

A distinction can be made regarding the application of the on-line test mechanism. A concurrent checking mechanism can be

- *Intrusive*: the CUT requires design modifications in order for the test strategy to be effectively introduced. The main disadvantage of this approach relates to the negative effect over the systems performance as a result of increasing CUT's latency. An example is the time redundancy approaches, which involves applying the same CUT operation at a successive moment in time, i.e. while the CUT is not used by the datapath (it is idle state) a previously applied input vector is re- applied

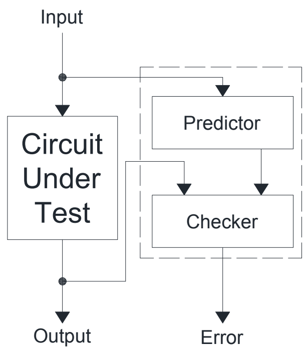


Figure 3.16 Hardware redundancy concurrent checking

to CUT's inputs and the result is compared against the value obtained previously. For this mechanism to be functional a distinct storage element and a multiplexer is required to be included in the system for allowing to store the result and select the normal or the test utilization of the CUT. The multiplexer however affects the datapath's performance. Another example consists in adapting a code-redundancy measure to the CUT. As will be presented, transforming the regular inputs of the CUT into specific codewords affects also the internal structure of the CUT or at least requires a codification and decodification stage assembled in the datapath before and after the CUT's inputs and outputs respectively, in order to transform codewords from and into non-codword vectors.

- *Non-intrusive*: the design in Fig. 3.15 and Fig. 3.16 depicts such a technique involving no modification of the CUT. It can be applied regardless of the specific implementation details of the CUT. No additional elements are required and it has no negative effect on datapath's performance. Moreover, a careful design of the error detection mechanism, involving buffering the error control line prevents the error indicator line to increase the overall system's latency.

The concurrent error detection techniques operate with a specific terminology regarding the ability of a detection scheme to protect the CUT against faults, especially intermittent faults. The following notions are presented in [17] as relevant for on-line testing:

- *Self-testing*: a CUT extended with a concurrent error detection mechanism is self-testing with respect to a set of faults if for every possible fault from the set there exist a CUT input for which the CUT's outputs are erroneous (a failure was triggered) [80].
- *Fault secureness*: a CUT extended with a concurrent error detection mechanism is fault-secure with respect to a set of faults if for every possible fault affecting the circuit the detection mechanism identifies the error at CUT's outputs [80].
- *Totally self-checking*: A self-testing, fault secure CUT is said to be totally self-checking (TSC). This property is crucial for reliable implementations.

With respect to self-testing property, as presented in the previous sections, there are fault conditions which cannot be detected (Fig. 2.7). We concluded that a circuit for which a defect remains undetected is redundant, a term also used in conjunction with designs whose structures can be further reduced by Boolean minimization, while the undetected fault is referred to as *redundant fault*. In order to preserve the self-testing and in consequence the TSC property, a redundant design need to be simplified.

An important aspect concerning the error detection capabilities of an on-line scheme relates to the correctness of CUT's inputs. Both Fig. 3.15 and Fig. 3.16 use the input vectors reaching the CUT directly, thus without verifying their correctness. In order to verify the validity of input vectors, either the datapath module supplying the CUT's input vectors (module M_1 in Fig. 3.15) will have to have the TSC attribute, or the vector exchanged between the two components will need to belong to an error detecting code [80]. The second solution relies on the property of the code-based mechanism to check codeword validity. However this alternative can also be interpreted as implementing the TSC attribute for the M_1 module. The conclusion is that a TSC system is composed of TSC components.

A general criterion for evaluating the on-line test mechanisms is offered in [80]: a proposed concurrent checking solution is "of interest" provided that the resulting design (the area for both the CUT and the on-line test mechanism) is "considerably smaller" than 220% of CUT's area and the single stuck-at detection is higher than 90%. This observation clearly evaluates concurrent test solutions with respect to the simplest on-line testing method: hardware duplication. Moreover, the important metric in evaluating on-line test solutions' detection rates remains the stuck-at fault model.

The self-testing, fault secureness and totally self-checking attributes previously defined, are evaluated with respect to a set of faults. Reference [80] alleges that the single stuck-at fault model "is used as a set of technical faults in almost all cases" with respect to the on-line testing. The rationales behind this decision are the following [80]:

- The majority of concurrent error detection mechanism are constructed based on the stuck at model and targets these faults.
- The "strong belief in the community" [80] that a fully covering stuck-at fault detection assures protection against most errors caused by faults not physically manifesting as stuck-at faults.
- The resemblance between the N-detect test vector set, targeting stuck-at faults and the totally self-checking attribute. There are literature references such as [86] and [87] whose experimental results confirms the ability of stuck-at fault model to detect not only different fault models but even unmodeled faults. Intuitively, an N-detect test vector set employs at most N, or the maximum number of test vectors, capable of detecting each stuck-at fault. Constructing such a test set reveals that more input vectors can be used to detect some of the stuck-at faults. While operating in the datapath, the CUT can receive any input configuration. Corresponding to the self-testing attribute, when affected by intermittent faults, erroneous outputs are produced by the CUT more than a single time allowing the detection scheme to detect the error multiple times.

Another motivation for stuck-at fault model's pertinent use in conjunction with concurrent checking mechanism can be justified by stuck-at fault's simple error manifestation. Regardless of the underlying cause of the fault (be it at the gate-level, transistor-level or layout-level related) the fault determines a failure, according to the terminology introduced in Chapter 1, provided that a deviation of system's behavior is detected. For the respective clock cycle, the circuit behaves as being stuck-at to an incorrect output vector. If more than a single line is affected, the cause could be an internal stuck-at line, whose effect manifests to more than a single output line, or the situation might as well be interpreted as a multiple-stuck-at fault. In other words, any variation from the correct output behavior, observed during a single clock cycle can be interpreted as a stuck-at fault, even if it is determined by a delay or bridging fault.

3.3.1 On-line Testing Techniques

The concurrent checking methods detailed in the literature are testing techniques proposed and "developed in the past" [83] and augmented by the current research in order to reflect the new realities of the VLSI technology. An authoritative reference with respect to the on-line testing mechanism is the survey performed by Michael Nicolaidis and Yervant Zorian, both of them being important figures in the testing community. The article [83] reviews an impressive number of 124 references in order to structure an on-line test compendium of solutions. Another, more recent reference, dealing with the on-line testing measures is [80] which doesn't presents the on-line testing solutions as a complete set of techniques, but instead focuses on code-based concurrent checking with emphasis on various error detection codes, and the associated circuit design strategies required for adapting the code-based on-line test mechanism to a particular architecture.

The different approaches to concurrent error detection are grouped together in classes, which are then briefly described in order to introduce the solutions proposed in chapter 5. In [83] the on-line techniques are differentiated into:

- "Self checking designs" [83]: encompassing error detecting code solutions, hardware duplication, time-redundancy approaches as well as all custom designs, providing concurrent verification mechanism and not covered by the next categories.
- "Signature monitoring" [83]: introduces the signature based on-line testing, with the test generation, response compression and response comparison. Moreover, in [83] the on-line signature mechanism is presented as an appropriate method for protecting the design of a Finite State Machines (FSM)
- Concurrent monitoring of circuit's physical parameters, such as the quiescent current, the operating temperature, the signal delays, the clock frequency, and the "radiation dose" [83].
- Specific BIST mechanism appropriate for the concurrent error detection [83]
- Specific Scan Chain techniques relaying on serializing the circuit's state for on-line verification of circuit's correct behavior using external test units [83]
- Fail-safe VLSI techniques [83]
- "Radiation hardened designs" [83] capable of performing correctly in the presence of ionizing radiation.

One reason for integrating on-line error detection techniques into current VLSI designs, as reference [80] asserts, relates to the increasing difficulty of attaining more than 99.0% fault coverage for non-stuck-at permanent faults in the current integration technology. The difficulties in achieving high coverage can be summed into the following:

- Increasing design dimensions: For designs with an ever increasing number of transistors (such as the Intel Prescot's 125 million transistors) the time required to test the design in order to fully cover all the fault models detailed in the previous chapters is impractically long, to such an extent that an on-line solution, permanently assessing chip's integrity is preferred [80].
- The decrease in transistor gate length: At smaller transistor dimensions, the parametric variance can easily develop into a defect. For example, high resistance shorts or high resistance opens, not affecting the logical behavior at lower clock frequencies, might provoke delay faults at the circuit's normal frequency. In order to detect these types of faults after fabrication, the test process is required to operate at the clock frequencies for which the delay faults provoke failures. The cost of Automated Test Equipment is increasing with the maximal frequency at which it operates. A possible alternative is the BIST strategy, which, for a fully covering test process would require an extensive period of time

as already detailed. The remaining solution is that of the on-line test paradigm [80].

- An increasing number of intermittent faults. The transient defects are usually considered to be provoked by "α-particle and cosmic radiations". Their effect over integrated circuits are more pronounced with the decreasing transistor size and lower voltage levels [80]. Moreover, the effect of "crosstalk error delay faults", marginally dependent on the particular environment and operating conditions, can only be interpreted as intermittent because of their non-repetitive behavior. These intermittent faults, as presented at the beginning of this chapter, can only be detected by means of on-line detection mechanisms [80].
- The unmodeled defects, for which the failure's cause or manifestation is not yet fully understood, cannot be targeted by off-line test process as no fault model was constructed for them. This category of defects is prevalent at smaller feature sizes, and can only be detected by on-line tests [80].

A description of the presented on-line test classes is described in the following sections together with relevant circuit design consideration for efficient implementing of the concurrent checking mechanism.

3.3.2 Self Checking Designs

The range of concurrent self-checking mechanisms conventionally encloses hardware duplication, code-based solutions and time-redundancy methods. Hardware duplication together with code-based mechanisms refers to hardware-redundancy methods, because the error checking uses additional hardware modules for correctness verification. For the case of hardware duplication, the CUT is copied as an additional instance and the correct behavior is verified by comparing the two instances' outputs while the two modules are driven by the same inputs. The second module, for obvious reasons is redundant. The secondary instance need not necessarily be a perfect copy of the CUT. It can also implement the same functionality by using a different approach as presented in [79] for the case of modulo 2^n+1 multiplication.

The code redundancy mechanism verifies that the output of the CUT preserves an *invariant property*. For a CUT for which the internal operation is known in terms of a truth table, for which no Boolean expression is provided for the outputs, and for which the space of inputs vectors is unrestricted, there is no evident invariant property. Trying to mathematically determine such a property or relation between inputs and outputs can become computationally intractable for circuits with numerous input and output lines. In consequence, the solution is to modify the input vector into a codeword for which an invariant property can be easily constructed, modifying the CUT accordingly, if necessary, and designing a checker for the respective invariant property. In this scenario, the predictor of Fig. 3.16 is expected to predict the invariable property of the output codeword based on the input codeword while the checker will compute the property for the current output vector and compared it with its predicted value.

The vector space for the codewords for which the invariant property is used is larger than the original vector space. However, the new codeword's vector space encodes the same amount of information: in other words, the error detecting codes have more bits than the original codes, but the number of possible correct input and

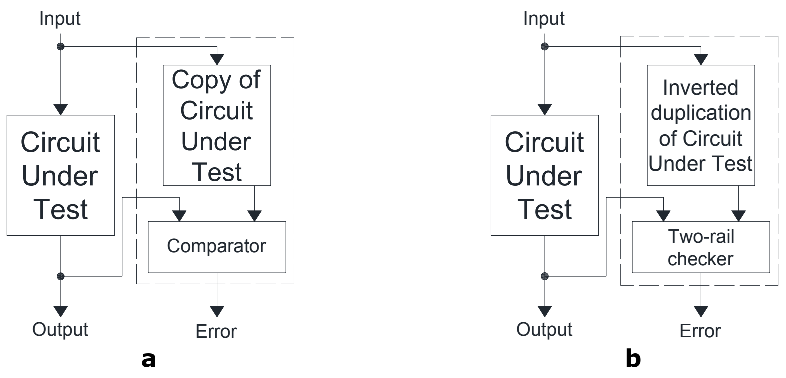


Figure 3.17 Duplication code (a) and Two-rail code (b) architectures adapted from [80]

output configurations remains the same. It follows that the supplementary bitsadded by the error detecting code are redundant: they carry no useful information. The code-based error detection thus supplements the output and the input vectors with redundant information bits and extends the CUT in order to accommodate the new code. The difference between the number of all possible configurations associated with the codeword length and the number of initial words (or the number of correct codewords) refers to the erroneous vectors. The validity of the output codewords assures CUT's correctness.

It must be noted that the error detecting code can be: *separable*, or *non-separable* [83]. Separable codes are also known as *systematic codes* and the non-separable codes as *non-systematic codes* [80]. A separable code maintains the initial code (non-redundant information bits) and forms the error detecting code by attaching a set of redundant bits. In this way, the bits of the initial code can be easily separated from the redundant data. In a non-separable code, the "information and the check bits are merged" [83]. This observation has a direct effect over the concurrent checking architecture: for separable code, the design is non-intrusive, while for non-separable the detection mechanism is intrusive.

For separable codes, the CUT remains unmodified because the new codeword can be separated into the initial code and redundant data. The CUT operates over the information bits while the predictor computes the redundant information for the output codeword from the input codeword. The checkers will compare the predicted redundant information with the actual redundant data obtained from CUT's output.

With respect to these definitions the hardware duplication can be interpreted as a separable error detection code, which doubles the initial code length. The redundant bits are exact copies of the information bits, the predictor is replaced by the second CUT instance and the checker is a simple comparator. This code is known as *duplication code* [80] depicted in Fig. 3.17a. A similar approach is the *two-rail code* for which the CUT is not duplicated but instead is redesigned in order to generate the complementary output vector when both modules are stimulated with the same input vector [80], as represented by Fig. 3.17b. The redundant information consists of the complemented information bits. There are sufficient literature references concerning the two-rail code circuit design, such as [88], [35], [37].

Another code for error detection is the *parity code*, which attach a parity bit to the information bits. The redundant bit can be computed using the *even* or *odd* parity: for even parity all codewords have an even number of ones in their configuration. For odd parity a similar rule is defined. However, in order to use a single parity bit for a CUT with multiple output line, the circuit is required to be developed so that "there is no sharing among the logic cones generating each output" [84]. For this scenario, a single stuck-at fault, as assumed in [84], will affect only one output line. If the logic cones for the outputs would share elements, a single output could affect an even number of erroneous outputs, resulting in aliasing situation for the parity code. The error is detected by recomputing the parity bit from the outputs and compared it with the predicted parity bit.

In order to alleviate the requirement of non-sharing output cones, *group parity codes* were introduced: all output whose logic cones share no elements are attributed to a distinct group. Each group is associated a parity bit. Output logic cones can share gates with cones belonging to other groups. If a single error affects the circuit, at least one parity bit will signal the error. Another parity related error detection code is the *checksum* computation [89], for which the summation of all words that are transmitted is performed. The different precision at which the checksum is computed and its various forms are described in [89].

Cyclic codes can also be used to determine faults. The codes are called cyclic because a rotation of the codeword (cyclic shift), results in another codeword. Their computation consists of multiplying the initial codeword by a primitive polynomial; however this approach results in a non-systematic code. A systematic cyclic code is obtained by multiplying the original codeword, represented as a polynomial, by the monic polynomial x^{n-k}, with n being the total number of bits of the codeword and k the number of information bits; the result of multiplication is then divided by the generator polynomial and the residue, which forms the redundant data, is attached to the initial information bits [89]. For error detection the systematic cyclic codes are preferred, and the detection consists in re-computing the redundant bits after the operation was performed and comparing it with the predicted value. The cyclic code having $x+1$ as its generator polynomial is the parity code: this observation makes the cyclic codes a generalization of the parity detection principle. The error detection using cyclic codes (also known as cyclic redundancy check) is used in [90].

Another class of error detection codes are the *unordered codes* [83]. For this category of codes one codeword does not cover other codeword, i.e. there is no codeword for which another codeword has a one in every position in which the first codeword has one. The invariant property for these codes is the non-covering property of codewords. It can be showed [83] that multiple unidirectional errors, that force the codeword bits in the same direction, for example, $1 \rightarrow 0$ can be detected because the affected vectors does not maintain the non-covering property [83]. Conventional unordered codes are *m-out-of-n* codes, *Berger* codes and *Bose-Lin* codes.

The m-out-of-n codes are non-systematic codes, for which each codeword has exactly m ones out of the total n number of bits. The Berger codes are systematic codes: the redundant information is constructed as the binary representation of the number of zeroes within the information bits. Another variant of the code represents the binary complement of the number of ones from the information bits into the redundant portion of the code. The number of data bits for Berger codes need to be less or equal to 2^N, with N being the number of redundant bits. Bose-Lin codes resolve this shortcoming allowing a smaller number of redundant bits to service a larger number of information bits. However, this comes at the price of being able to detect maximum N unidirectional errors using N

redundant bits as opposed to Berger codes which can detect unidirectional errors of any multiplicity [80], [84].

In order for the unordered codes to detect faults in a circuit, the CUT needs to be constructed so that for each node susceptible to faults, all paths from the node to the output will have the same inversion parity (either an even number of inversions or an odd number of inversions) [83]. There are references presented in [83] treating the transforming of a CUT into a design compliant with these restrictions.

The class of arithmetic codes is represented by the *residue codes* which can be both systematic and non-systematic. The separable code implementation constructs the redundant data as the residue of the information data computed modulo the code's *base* [83], [84]. The inverse residue code is defined similarly in [83]. Non-systematic arithmetic codes compute the codewords by multiplication of the original vectors by the base. The separable arithmetic codes are preferred provided that the base is carefully chosen in order to perform the residue operation efficiently. Usually the base is of form 2^n-1. A modulo 3 residue code can detect any single error, using only 2 redundant bits.

It must be noted that *error detection and correction* codes are also used, which, besides detecting errors also permit correcting a limited number of them. Examples are SEC/DEC Hamming codes, Reed-Solomon codes, BHC codes just to name the conventional solutions.

With respect to the checker, the first observation is that an error affecting the checker can mask any other defects in the CUT. Circumventing this issue is not an easy task: first of all, it raises the well-known problem of "who checks the checker". As reported in [83], for the proposed error detecting codes, the research literature have extensively developed self-testing checkers. The task of constructing such units is difficult because on one hand the checker is expected to generate an erroneous response for every erroneous output codeword, and a correct response for a correct output codeword. Moreover the self-testing principle requires that under the influence of a fault, the checkers have to signal an error for a correct output codeword.

Moreover, in order to avoid the critical effect of a stuck-at fault affecting the checker's error indicator, the checker is recommended to signal the error status using two lines. A two-rail code could be employed for which only two of the 4 configurations are valid. A detailed discussion regarding the particular implementation details for the CUT and the checker with respect to the previous discussed error detecting codes is offered in [83], [80] and [84].

The linear error detection codes, such as parity codes, cyclic codes, hamming codes, BCH codes, have the disadvantage of dividing the set of all possible errors into two disjoint sets: one from which all errors are detected and one for which none are detected [80]. Dependent on the dimension of redundant data, the covered errors prevails or in the worst case the two error sets have the same cardinality. The simple parity method, using a single parity bit can detect only odd errors and none of the even errors. In [80] is described a category of error detecting codes called, *non-linear split error detecting codes*. These codes divide the possible errors into fully covered, completely undetected, and covered with a ½ probability [80]. Usually there are only a few uncovered faults. The non-linearity is a consequence of using both the AND (or NAND) and OR (NOR) operators in constructing the code.

Concerning the time redundancy measures, as already stated, the simplest form involves enhancing the CUT's input with a multiplexer, storing the CUT's result

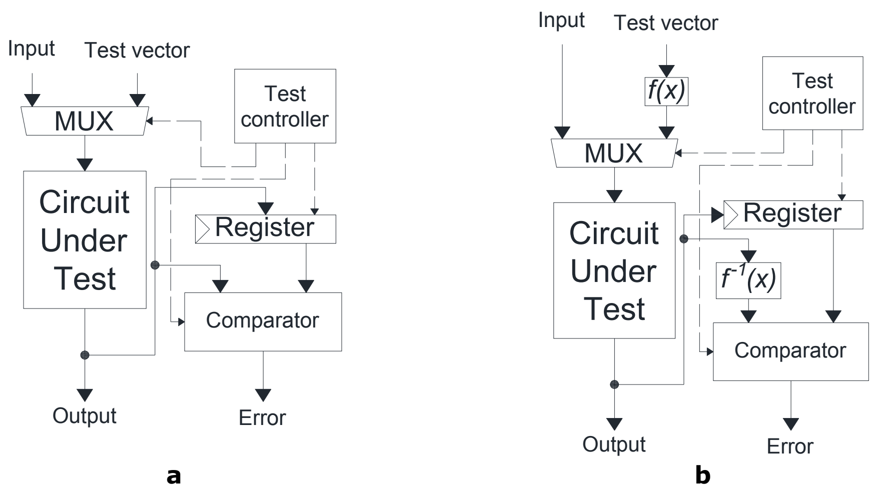


Figure 3.18 Simple time redundancy (a) and Re-computing with shifted operands (b) strategies

for particular input vector in a register and in idle cycles reapply the specific input to the CUT in order to compare the result with the previous one. Apart from affecting the CUT's latency this method requires monitoring the CUT's inputs for determining the specific input vector. This approach is depicted in Fig. 3.18a.

One disadvantage of the simple time redundancy check as observable also from Fig. 3.18a is that permanent faults are not detected. A variation of this method is *re-computation with shifted operands* as presented in [91]. It avoids the use of the same CUT inputs in order to be able to test for permanent faults also. This approach is depicted in Fig. 3.18b. As can be seen from the figures, whereas the simple time redundancy mechanisms, stimulates the CUT with the same input vector, the re-computation with shifted operands, modifies the input vector by applying an supplementary function *f(x)*, and after the CUT results are obtained, the inverse of *f(x)* is applied in order to compensate for the modified CUT input. When choosing the additional function and its inverse the designer need to assure the final result for the correct circuit is equal to the result of the CUT not altered by these two functions otherwise a compensation mechanism is required. For arithmetic units (adders, multipliers) an efficient *f(x)* function is the cyclic shift, for which the inverse function is easily constructed and devising the compensation mechanism allowing comparing the original result with the re-computed one is straightforward.

Another time redundancy mechanism, as presented in [84], is the *alternate-data retry*. It is a reliable error correcting mechanism, which depends on retransmitting the erroneous data (the temporal characteristic of the method) but in an alternative form. The alternative codification for the codewords is specifically constructed in order to compensate for various stuck-at faults affecting the communication lines.

The code-redundancy methods and hardware duplication can be combined as in [80]. The method requires that the CUT would consist of two separable sub-modules. One module (the one receiving the CUT inputs) is protected by means of code redundancy while the second module is entirely duplicated. Efficient protection mechanism can be obtained when the trade-off between code-redundancy fault coverage and hardware redundancy complexity is carefully selected.

3.3.3 Supplementary Concurrent Checking Mechanisms

Reference [80] describes other concurrent error detection mechanism used either as a complementary detection method (physical parameter monitoring) or as strategies at the system's architecture-level for performing the test process (BIST, Scan Chain).

The previous chapter presented the usefulness of quiescent current monitoring for detecting physical defects otherwise difficult to detect by the presented fault models. The *physical parameter monitoring* is concerned with non-logical properties of a circuit. As reference [83] underlines, these methods "include in a natural way the invariant property required for on-line monitoring".

The I_{DDQ} monitoring can be performed using external sensors or by incorporating Built-In Current Sensors (BICS) into a design [83]. These monitoring mechanism are operating at the circuit's normal clock frequency, being able to detect physical defects at-speed. However, there are some drawbacks associated with BICS, as [83] details:

- The integration process is the same for BICS and for the circuit containing it. However, the parameter variance for BICS is required to be much smaller in order to improve sensor's detection quality [83].
- The BICS might increase the circuit's latency [83].
- For I_{DDQ}, the current is measured at the quiescent regime, however, for fast integrated circuits, the logic levels are established before the circuit reaches the quiescent state. And because the sensors takes the logic levels as reference for determining the moment of current measuring, the clock signals are required to be slowed down for this measurement to take place.

In order to resolve these issues, the BICSs are not placed along the circuit's critical path. Another solution would be to perform the monitoring task periodically. However, the reliability of this monitoring approach suffers by not being able to detect the errors produced between two consecutive tests.

Other reliability indicators can be monitored [83], such as the *operating temperature*, the *dissipated energy*, *voltage levels*, the *outputs steadiness* and *the radiation total dose* [83]. More recently robust *low clock frequency detectors* were proposed in [92] for those specific cryptochip attacks which force the cryptographic system to operate at lower frequencies in order to efficiently identify the algorithm's steps and derive the key.

Another concurrent checking mechanism relies on *signature monitoring*. Apart from being an important component of a typical off-line BIST design, the signature monitoring as presented in [83] and [79], can be used also to test the finite state machines of a design. Typically, the control unit, for a hardwired implementation (as opposed to the microprogrammed solution) is implemented in terms of an automaton. In [79] the finite state machine implementing the control unit, and being designed as a Mealy or Moore solution is transformed into a Medvedev state machine, whose structure resemble a Linear Feedback Shift Register allowing to assign a signature corresponding to the automaton state transitions.

Fail safe designs are concerned with assuring the outputs of a circuit in the presence of a fault are either correct (fault unaffecting the outputs) or safe [79]. The quality of being safe is defined with respect to the systems and operations the circuit's outputs command. This design feature is a necessity in critical-oriented systems (avionics, railway control).

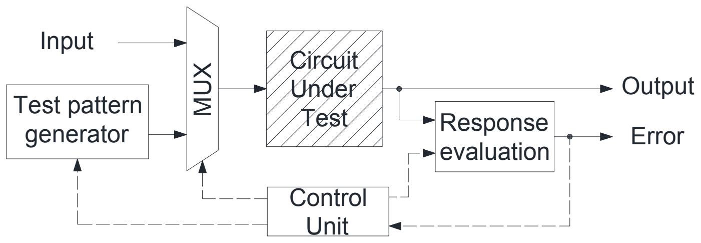


Figure 3.19 Conventional BIST architecture adapted from [68]

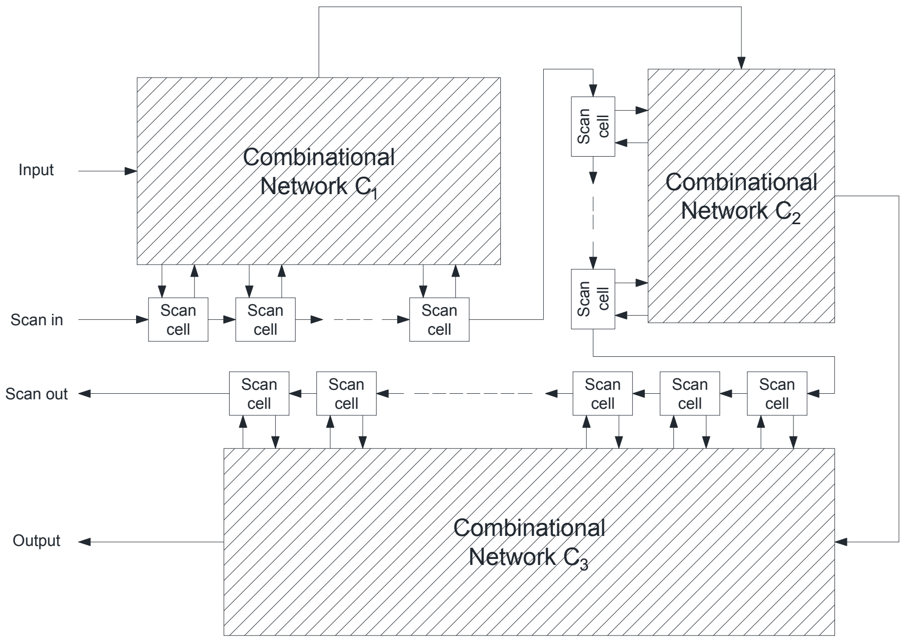


Figure 3.20 Conventional Scan Chain architecture

The *Built-In Self Test* methodology was already presented. As for the *Scan Chain* design, it relies on architecture extensions permitting to serialize the internal state of the device. The method typically connects all or the majority of the storing elements (registers, flip-flops) of a design into a single or multiple chains, addressable from the exterior, allowing to read or write the content of the registers for testing purposes. Scan Chain is conventionally an off-line test mechanism. The two paradigms: BIST or Scan Chain are evaluated and selected at the very beginning of the design phase as they usually embed all the testing activities for the design. The two architectures, namely the BIST and Scan Chain approach, are depicted in Fig. 3.19 and Fig. 3.20 respectively.

Chapter 4
Advanced Encryption Standard

For a long period of time, cryptographic systems were employed only for protecting communications between military and governmental officials. As a consequence, the information was protected through algorithm's secrecy due to the fact that very few individuals had the required competence in cryptosystem's field. Moreover, the cryptographic systems were operated only on few communication mediums such as teletype, telex, facsimile, voice, radio and data [93]. However, as our society pass into the so-called "*information era*", the information protection became an absolute necessity, triggering the rapid embracing of the cryptographic services at a large scale. The advent of large scale integration (LSI) and eventually the very large scale integration (VLSI) allowed implementing security algorithms in hardware for increased performance. The need for standardizing enciphering algorithms led to the selection of Data Encryption Standard as Federal Information Processing Standard in 1976. Another important step toward general cryptographic primitives' proliferation was the development of Secure Socket Layer allowing for secure communication over the Internet. One of the first promoters for cryptography development and its integration into everyday's life was businesses' need for effective protection. Cryptography's applicability was driven to such an extent that it covers Internet credit card payments, bank transfers, ticket reservation, medical assistance, pay-per-view television services, electronic mail, mobile phone communication to name only a few.

Cryptography is formally defined by the following security services [94]:

- *Confidentiality*: assures that only the entities authorized to access the protected information will have this right. It is typically achieved by using the encryption security mechanism.
- *Data integrity*: is a security service assuring the data is not accidentally or deliberately modified during its transfer from the source to its destination (by replacing, deleting or inserting new data into the message). Hash functions, as a mechanism of digital signature, are effective mechanisms for enforcing data integrity.
- *Authentication*: offers the receiver of the message the assurance that the received data is coming from the source it claims to come. It is automatically enforced by encryption (if the receiver possesses the correct encryption key used by the sender, it will be sure the message was sent by the entity using the same key). Digital signatures can also be used for authentication.
- *Non-repudiation*: none of the entities participating in a secure communication can repudiate its participation. This service protects against an individual denying receiving or sending data. It can be manifested as non-repudiation with proof of delivery or with proof of origin. Digital signatures can be used for this purpose.

A taxonomy of cryptographic primitives used for achieving these security services is presented in Fig. 4.1.

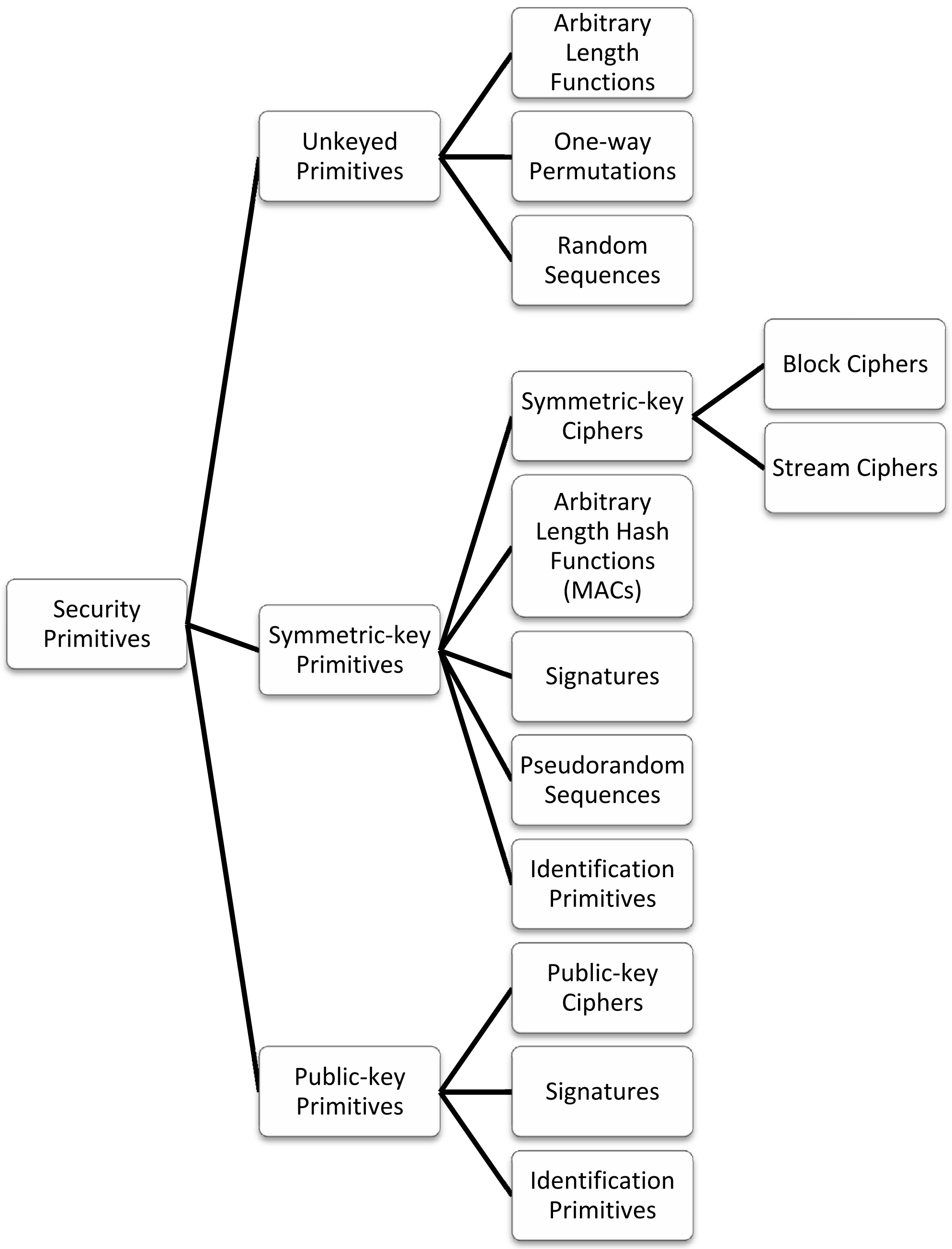


Figure 4.1 Cryptographic primitives taxonomy adapted from [94]

Advanced Encryption Standard became the new Federal Information Processing Standard in 2001 after being selected by National Institute of Standards and Technologies as the successor of Data Encryption Standard (DES). After a significant period of use, DES lost its security strength. Even close to DES's selection, it was argued that a machine capable of performing known plaintext attacks can be designed. At Crypto '93 Rump session, a detailed description for a DES key search design was delivered by Michael Wiener [95]. Although the proposed machine was never constructed, in 1998 the "*DES Cracker*" was built, capable of finding a DES key in 56 hours. Using today's computational power DES can be broken in a "few hours by launching a brute-force attack" [2]. The process of selecting a DES replacement was initiated in 1997. AES was required to operate with 128 bit data blocks, supporting 128, 192 and 256 bit keys and being royalty-free available. The selection process evaluated the candidates based on *security, cost* and *implementation* characteristics. The cost refers to resource requirements for implementing the algorithm on various platforms such as Application Specific Integrated Processors, Field Programmable Gate Arrays, Smart Cards and in software. The most important characteristic sought for AES was simplicity and algorithm flexibility. The Rijndael was eventually adopted as the new AES because of its implementation efficiency, flexibility and long-term foresaw security. National Institute of Standards and Technologies expects the security lifetime for 128-bit keys symmetric encryption algorithms (including AES) to last beyond 2030 [96].

When extending a cryptographic system with tamper-resistant features, the range of possible applications widens considerable. This combination first appeared in conjunction with securing military communication lines. The secure cryptosystems eventually opened its path into the commercial sector, being firstly adapted for Automated Teller Machines. In the following a brief description of the applications that can be served by a cryptosystem is presented [97]:

Automated Teller Machine: the main sector that still employs "high-volume use" of tamper-resistant security hardware. It also propelled the use of encryption hardware into civil services from its military-only use. Hardware Security Modules were first used as a mean to authenticate customers to ATMs. Currently the cryptoprocessors control all operations related to the PIN in a typical ATM (PIN acquisition from the keypad, PIN validation as well as PIN generation).

Electronic Payment Services: the natural tendency was to integrate security modules in financial services, similarly to the case of ATMs. In consequence, by using secure communication lines between banks or between bank and its costumer (individual merchant, supermarket, chain store) electronic payment systems have emerged. This new platform is required to be able to store verification keys specific to the bank in order to authenticate card payments. All-in-all, the general tendency with respect to this domain is to assure ubiquitous home banking (some attempts have already been made, by introducing stand-alone authorization devices).

Prepayment Electricity Meters: allows for a better budgeting with respect to electricity consumption. It is usually employed by electricity suppliers when the customer cannot sustain regular payments on their bills. In this case, it is evident the evolution: from the initial mechanical prepayment meters up to the reliable digital systems, that uses SmartCard technology. The meter recognizes authentic, encrypted information supplied by user through a prepayment key. Based on the current counter value stored inside and the payment information from the "credit token" it supplies energy. Once again, the Hardware Security Module is required to offer a trusted platform for storing the counter value and encryption keys. Thus, cryptography allows the power companyes to deliver inexpensive meters without being threatened by energy theft frauds.

Trusted Computing: is baked by the Trusted Platform Module objective to integrate cryptographic modules into PCs and mobile devices. Currently it is used in Lenovo ThinkPad Laptops (for disk encryption among other services). However, the new generation TPM chips aim to accommodate other security related tasks, such as authentication to other trusted machines, certifying a specific property (such as whether or not a particular program is executing), and assuring security in the Microsoft's trusted Windows OS. In order to offer a general approach for cryptographic interface construction, TPM will work in conjunction with a micro-kernel responsible for cryptographic primitives. The most important utilization would be for Digital Rights Management, sustaining the development of secure on-line services. A service provider would be confident that the information was sent to a valid customer rather than to a malicious one. It is easy to understand the vendors' benefits in a trusted computing platform when looking to the current DRM attained by software obfuscation, which eventually is tampered. DRM could also control the "flow of information within organizations" [97].

Military Equipment including devices to transport keys, tamper-resistant cryptographic devices using classified algorithms and even a command and control environment for secure management of the nuclear arsenal can also be mentioned as possible application of cryptographic processors.

4.1 AES Hardware Implementation

Advanced Encryption Standard as defined by Daemen and Rijmen [98] is a block cipher, operating encryption and decryption on data blocks using a secret key. It is a substitution permutation network, operating in an iterative manner, with a particular sequence of operations (collectively described as around) being repeated a certain number of times. The AES operations are byte-oriented, which is also the reason for its performance on resource-limited processors as well as on current 32 and 64-bit architectures. The design criterions were guided by Shannon's notions of *diffusion* and *confusion* as presented in his "Communication of Secrecy Systems". Diffusion will disperse the plaintext and key information into the ciphertext and is typically achieved in cryptographic algorithm by use of permutations. The confusion property achieves a complicated relation between the inputs (plaintext and the key) and the encrypted output, and is usually implemented by use of substitutions, or *SBox*-es.

The AES algorithm operates with 128-bit data blocks and accepts keys of 128, 192 and 256 bits. Dependent on the key length the algorithm is performed by iterating a different number of times the round transformations. The most used implementation, and the one analyzed in this thesis, as well as the one provisioned by NIST to maintain its security beyond 2030 [96] is based on 128-bit keys. AES belongs to the key-alternating block cipher class because its common round transformation is parameterized only by the key, and in consequence both encryption and decryption round depends on its correspondent round key.

The AES encryption algorithm is described in Fig. 4.2. The simplicity at the algorithm level manifests in its symmetry and the use of basic mathematical operations [98]. This property in turns allows for rigorous mathematical analysis of its behavior against common known attack strategies, also confers implementations higher performances (both software and hardware) and allows a higher degree of reusability by designing a single round which will iteratively performs the algorithm. From a software point of view, Daemen and Rijmen mentioned AES' implementation efficiency on 8, 32 and 128 bit processors [98]. The modularity of the algorithm strongly linked with its simplicity is responsible for its flexibility, being capable of implementations on limited platforms ("on a grain of sand" [99]) as well as targeting high speed applications. The modularity allows to add subsequent instances of the same operation for an increased performance, i.e. because SubBytes transformation operates on all the bytes of the state and it is also the most complex unit, on a SmartCard platform the AES unit would contain a single SubBytes instance with algorithm's control flow and datapath correspondingly modified in order to share the expensive module, while for a high speed Terabit network controller, the maximum number of SubBytes instances will be present for a high speed parallel processing. The same observation stands also for MixColumns transformation. The downside of this observation is the fact that instances' multiplicity has an important role over the power consumption.

Besides allowing constructing mathematical proofs for algorithm strength, the use of basic mathematical operations permits, in addition, simple, efficient and straightforward implementations for the most AES operations. All the mathematical operators involved are defined over the Galois Field - GF(2^8) - specific to the AES algorithm: the field generated by the irreducible polynomial *m(x)* represented in equation (13).

$$m(x) = x^8 + x^4 + x^3 + x + 1 \tag{13}$$

The intermediate results of applying AES' round to an input plaintext are denoted by the term *state*. The state can be interpreted as a byte matrix with 4 lines and 4 columns. During encryption the plain text represents the initial state and the final cipher text denotes the final state. The four transformations that builds AES' typical round are: SubBytes, ShiftRows, MixColumns and AddRoundKey. Correspondingly, for decryption, the inverse transformations are defined: InvSubBytes, InvShiftRows, InvMixColumns and InvAddRoundKey. The sequence of the four transformations within a round as well as the manner in which each of them affect the state is represented in Fig. 4.2.

4.1.1 SubBytes

SubBytes is a non-linear transformation transforming each input byte into an output byte and affecting all 16 bytes of the state [7]. For a high throughput implementation, 16 instances of the SubBytes transformation will concurrently process all state bytes. Literature reports that in a high-speed AES design the cumulated SubBytes instances together with the design's registers demands as much as 85% of chip's area [100] making SubBytes the most critical element of a hardware design as well as of a software implementation [98].

Literature presents various implementation methods, with the simplest one (conceived by the very creators of AES) using lookup tables in order to compute the SubBytes' result. However, by inserting the ROM-based lookup tables, the chip's area increases considerably, especially for the case of encryption-decryption AES designs, in which both SubBytes and InvSubBytes are required. Another alternative is to compute the two transformations using their mathematical definitions by

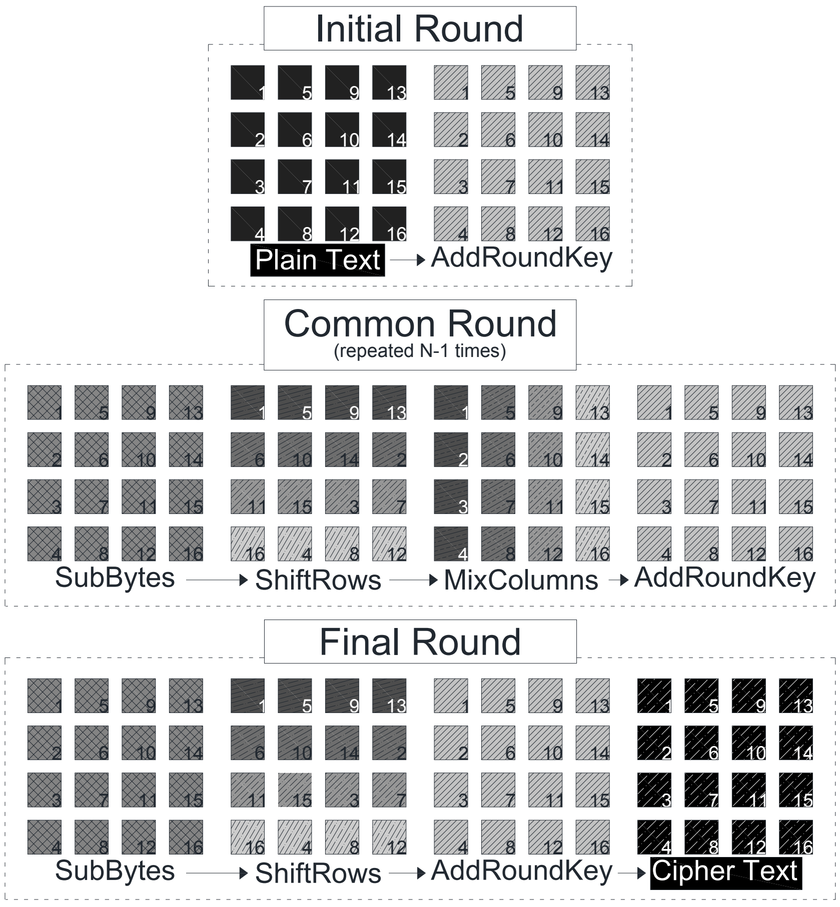


Figure 4.2 AES encryption algorithm

applying the affine transformation to the result of the multiplicative inversion performed in the finite field associated with AES.

As literature reports reveal, the inversion operation in finite fields is non-trivial, the AES' GF(2^8) making no exception. In consequence, in [101] is presented a hybrid solution that use look-up tables only for implementing the Galois Field inversion. SubBytes and InvSubBytes however will require the additional step of affine operation and inverse affine respectively. This solution is more efficient from hardware resources' point of view (both transformations share the same finite field inversion unit). For some designs (ASIC for example) the presence of ROMs can become inconvenient when designing for a higher throughput due to their well-known resistance against pipelining methods.

Yet another solution implements the field inversion as a combinational design. This class of implementations relies on Daemen and Rijmen observation regarding the composite fields and the ability to construct an isomorphism between AES' Galois Field and a composite field. The Galois Field GF(2^8) is isomorphic with the field GF(2^4)^2, that is, a mapping function can be found such that any element of the GF(2^8) to be associated to a pair of elements from the GF(2^4) (the elements pair is interpreted as a degree I polynomial with coefficients in the GF(2^4) field). There are literature references documenting the process of generating the isomorphism matrices starting from the polynomials used to construct the field GF(2^8) and the composite field GF(2^4)^2 [102]. However supplementary combinational logic is required for the initial map of one GF(2^8) element into the corresponding two GF(2^4) elements, and, after computing the inverse pair of GF(2^4) elements, for the final mapping in order to obtain the associated inversed GF(2^8) element. Although the solution is desirable when the unit's area or power consumption is an issue, the unit's performance is degraded by the two mapping stages. Other research, as proven in [103], drove the composite field applicability forward by reducing inversion in GF(2^4) to inversion in GF(2^2). Inversion in the latter finite field is even simpler to implement because it operates only on 2-bit values.

SubBytes transformation is also employed during the round key generation process requiring four instances of the operation. For a design computing the round keys on-the-fly, the 4 instances can be either shared with 4 of the 16 instances from the datapath or be separately implemented. It must be noted that a shared implementation requires additional control logic, as well as registers within the datapath in order to multiplex the inputs for the mutually used modules. Besides these, each round would require 2 clock cycles for completing in order to schedule two executions of the SubBytes transformation, leaving unresolved the problem of operations balancing for each cycle.

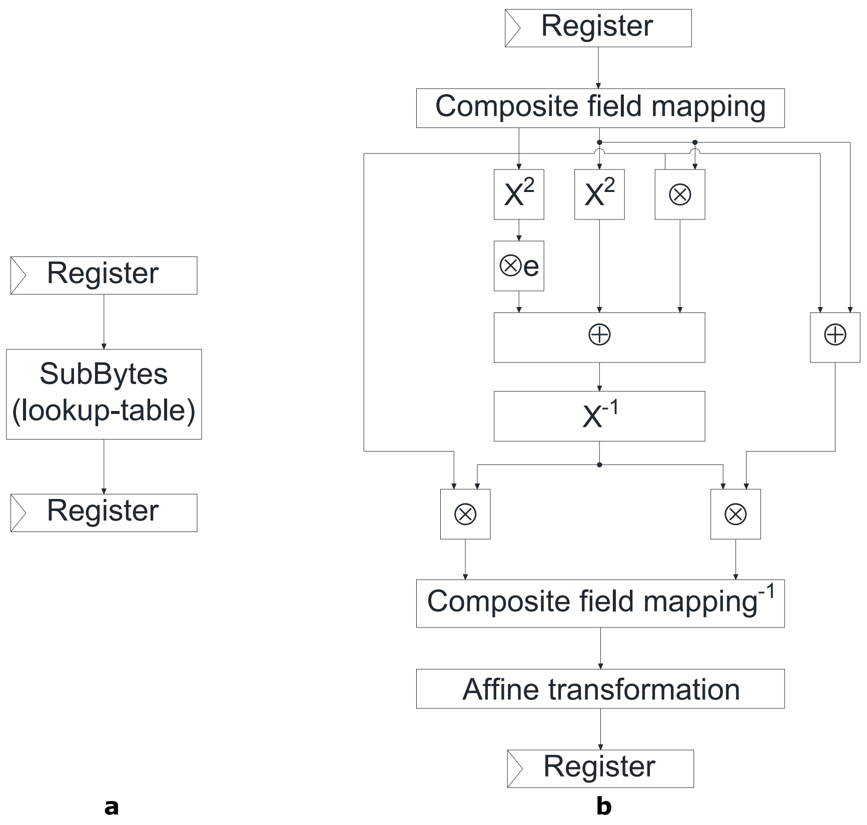


Figure 4.3 SubBytes implementation alternatives: lookup table (a) and composite field (b) adapted from [104]

A concise description for the most notable SubBytes implementations solutions including lookup table ones and composite fields approaches is presented in [104]. Experimental results are also documented regarding the speed-up for each particular design. For a complete analysis, the pipeline acceleration techniques were taken into account revealing that the most efficient implementation is the one that uses as many inner pipeline registers as possible.

Figure 4.3a depicts a typical lookup table SubBytes design while Fig. 4.3b illustrates a composite field combinational implementation. The mathematical relations in the latter case were derived by expressing one GF(2^8) element in terms of two elements of GF(2^4). The inversion unit uses addition, multiplication and squaring in the GF(2^4) field, operations which can be efficiently implemented using simple binary operators.

4.1.2 ShiftRows

ShiftRows is a row-wise transformation, modifying all four bytes of each state matrix row [7]. Each row is shifted to the left by a number of times dependent of the row's index. InvShiftRows, correspondingly shifts each row to the right in order to reverse the effect of ShiftRows. In a 128-bit architecture, for which each round transformation process the state matrix in a single pass, these two operations reduce to mere lines' re-routing. However, in a resource-constrained architecture, in which a reduced number of SubBytes instances are implemented (possible only one), the ShiftRows requires a dedicated stage.

4.1.3 MixColumns

MixColumns operates column-wise over the state matrix [7]. Each column is interpreted as an element of the GF(2^8)^4 composite field, i.e. each bytes of the column is a GF(2^8) coefficient of a degree-3 polynomial. The extension field is generated by the polynomial *M(x)*, represented in (14) and the new column is obtained by using a linear transformation which multiplies the initial column by the constant polynomial *c(x)* represented in (15).

$$M(x) = x^4 + 1 \tag{14}$$

$$c(x) = \{03\}x^3 + x^2 + x + \{02\} \tag{15}$$

In the mathematical equations during this thesis, elements of the GF(2^8) field are written between curly brackets. Because all the GF(2^8) coefficients involved in this operation are small the multiplication can be realized efficiently: multiplication by 2 is implemented by the *xtime()* operator and multiplication by 3 reduces to a multiplication by 2 and one addition in the finite field. The inverse operation, InvMixColumns requires multiplication with the inverse polynomial *d(x)* from equation (16)

$$d(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0e\} \tag{16}$$

In order to implement multiplication by 11, 13, 9 and 14, the same method, used for multiplication by 3, is applied, but with evident lower performances due to the high coefficients involved. This can become an important issue when implementing AES designs capable of performing both encryption and decryption. A design in which both the MixColumns and the InvMixColumns instances are present will unnecessarily increase design's area. Moreover, although MixColumns performs faster compared with InvMixColumns, the overall performance will be given by the largest latency of the two operations, which also conditions the fastest clock that can correctly drive the design. In consequence, a design that exhibits hardware reuse between the two operations is desirable.

To solve this issue, Rijmen and Daemen presented an efficient realization method for InvMixColumns based on the mathematical property between *d(x)* and *c(x)* described in (17).

$$d(x) = (\{04\}x^2 + \{05\})c(x) \tag{17}$$

Corresponding to relation (17) InvMixColumns can be obtained from the MixColumns transformation result by using a subsequent multiplication. Another alternative for integration of the two transformations would implement a single module for reduced area requirements. To the authors' knowledge, this method was first introduced in [101]. The innovation introduced by this approach relies on the mathematical similarities between the common factors that can be shared between the two transformations. The driving concept of this alternative is to obtain a set of mathematical relations to describe both transformations, parameterized by a single variable that would select the appropriate operation.

However, the mathematical relations in [101] are subject to a mathematical incoherence for the InvMixColumns implementation. Starting from this solution's

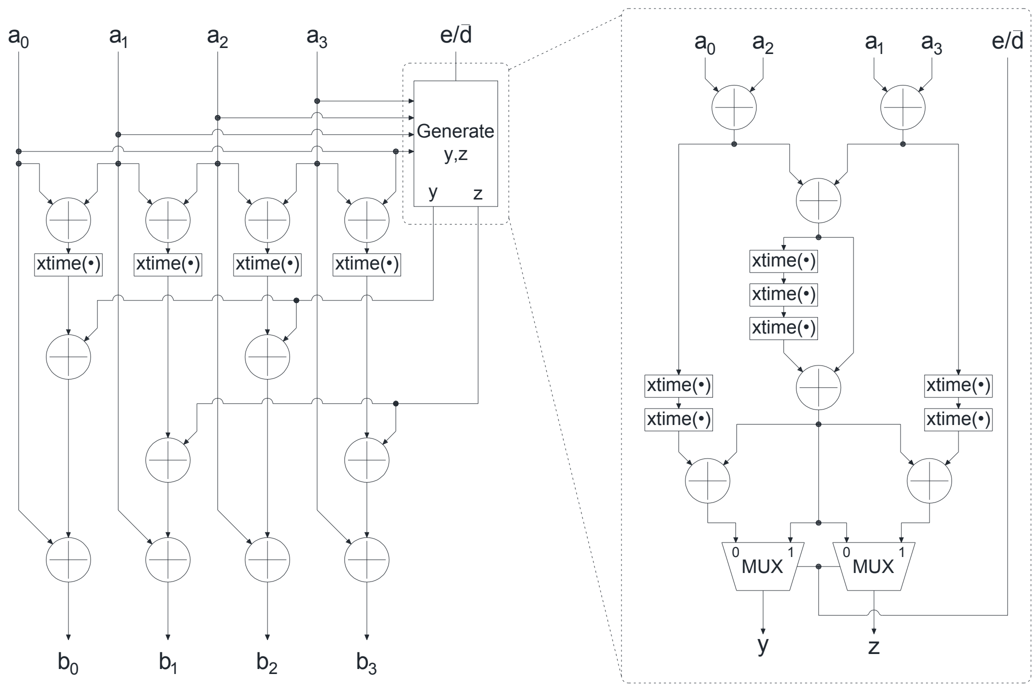


Figure 4.4 MixColumns–InvMixColumns unit adapted from [101]

main goal (to share as many elements as possible in order to save up silicon area) we constructed the correct mathematical expression for obtaining the desired implementation. The design is described in Fig. 4.4 (the module operates over a single state column) in which the *encrypt/not decrypt* line selects one of the two operations. The entity that generates *Y* and *Z* parameters required in computing the final result is detailed in the left side of Fig. 4.4.

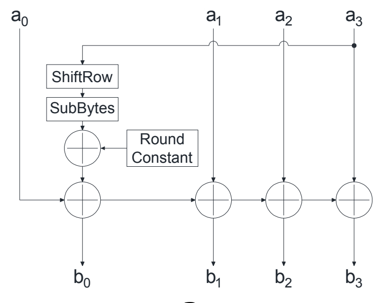
4.1.4 AddRoundKey

AddRoundKey adds every round the current round key to the state [7]. In the GF(2^8) field, addition is implemented as bit-wise EXOR between the two elements. And because EXOR function is its own inverse, AddRoundKey implements also its inverse round transformation.

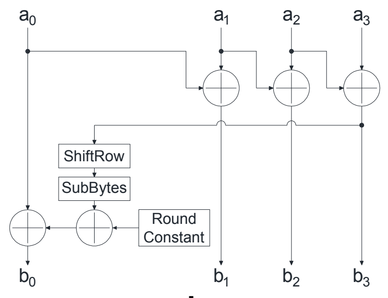
4.1.5 Key Scheduler

The *key schedule* comprises of two processes [98]: one for generating the key material and the other for selecting the current round key required by the AddRoundKey transformation [7]. AES-128 consists of 11 round keys: in encryption the algorithm's initial key is also the first round key. Every round key has the same length with the state (and also the data) block. Because AES standard can operate with different initial key lengths: 128, 192 or 256, the process of computing the key material and the selection of each round key differs dependent on the key different lengths. As already stated, the most common implementation operates with 128-bit keys, which is also the solution adopted in our designs.

Two mechanisms for generating the round keys are taken into consideration. The first method is to pre-compute and store the entire key material



a



b

Figure 4.5 AES round key generation for encryption (a) and decryption (b)

in a dedicated RAM, from where the selection process will access each round key on demand. This approach is advantageous for situations in which the algorithm's key do not change too often (operating with the same initial key for subsequent data blocks), and in consequence the computation of the same key material, during each encryption is redundant. The other alternative is to compute keys on-the-fly. This is possible because, hardware implementations allow for concurrent execution of different modules. In this case one unit would be the AES' datapath performing the enciphering algorithm while the second unit would generate the round keys, the only observation is that the 2 modules require appropriate control logic in order to synchronized key scheduler with respect to the moment the AES datapath requires a round key. The round key generation process is facilitated by its iterative nature. On-the-fly computation of the key material is efficient when round keys change regularly, because the initial latency for computing the keys is absorbed into the AES execution. However, this solution might exhibit higher power consumption for cases where RAM-based design are advantageous because the power consumption of the key schedule is expected to be higher than that of the round keys RAM. Structurally, the key schedule unit contains 4 SubBytes instances, a simple shift operator and a module providing the round constant. The round constant is introduced in order to eliminate the key symmetry [98].

In Fig. 4.5a and Fig. 4.5b are described the round key generation process for encryption and decryption respectively illustrating the iterative property of the key scheduler allowing on-the-fly generation as well as its invertible property with respect to encryption and decryption.

4.1.6 AES Decryption

The straightforward decryption algorithm, executes the inverse operations associated with each transformation in reverse order compared with AES encryption [7], this process being similar in nature to a composition function inversion. The operation sequence for this approach is described in Fig. 4.6. Although it can be directly constructed, implementing an encryption-decryption AES design that makes use of the encryption algorithm described in Fig. 4.2 and the decryption sequence from Fig. 4.6, poses hardware implementation challenges. More precisely, although the two algorithms have the potential for significant reuse within the operations involved (the high complexity inversion module for both SubBytes and InvSubBytes; MisColumns and its inverse as previously detailed), the corresponding operations are not executed at the same stage in encryption's and decryption's round respectively. For example, during encryption SubBytes is at the beginning of each common round while in Fig. 4.6 it is at the end of the round. In order to accommodate a common design with both these distinct processes, is required a significant overhead for control logic as well as routing elements (multiplexers).

Daemen and Rijmen proposed reordering the typical decryption round swapping the position of InvShiftRows and InvSubBytes because of their non-interfering algebraic definition (InvShiftRows transpose each byte having no effect over their values). Analyzing the reordering of AddRoundKey and InvMixColumns (seeking to construct the same sequence as for encryption), in order for the AddRoundKey to succeed the mixing operation an additional InvMixColumns application to each round key is required [98]. But, this additional InvMixColumns not only increase the hardware cost but also increase the latency of the key generation unit. This can be avoided by allowing the AddRoundKey and

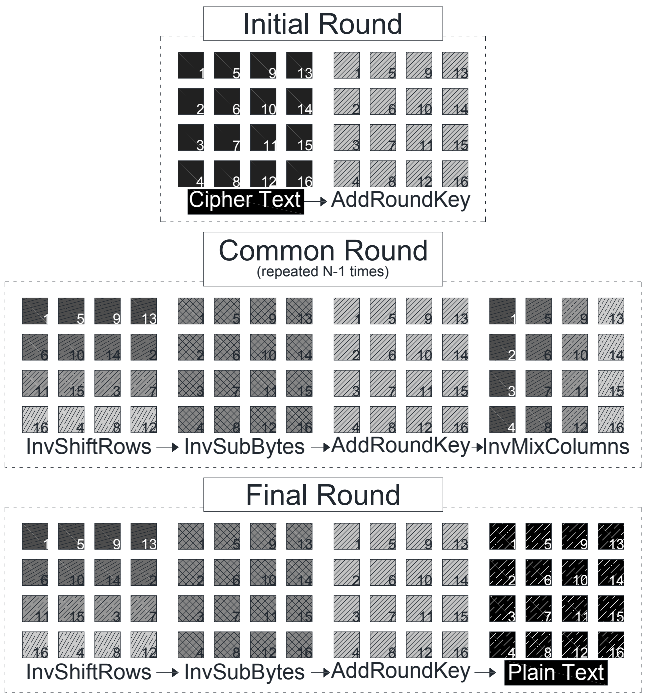


Figure 4.6 Straightforward AES decryption algorithm

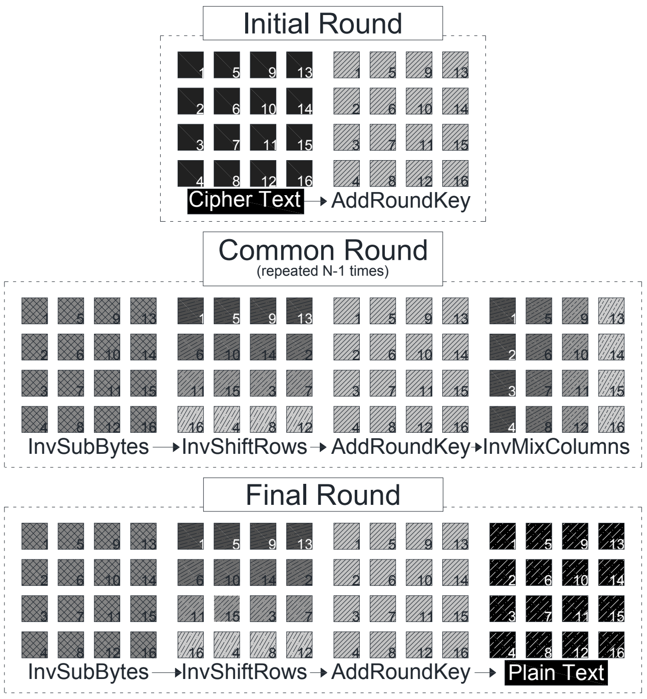


Figure 4.7 Modified AES decryption algorithm

InvMixColumns to be executed at different stages in encryption and decryption. The generic decryption algorithm that takes into account these observations, as well as the algorithm used in our AES implementations throughout this thesis, is depicted in Fig. 4.7.

When considering the natural definition of decryption as the inverse sequence of encryption's transformations it is obvious that the decryption process requires the set of round keys provided in reverse sequence compared with encryption. Except from the case of RAM-stored key material, this task is also non-trivial. One reason for key generation complexity is computing the initial decryption key. Considering that the initial encryption key has N_K bytes (in our design N_K=16, however, for larger keys the next observation still remains valid), the initial decryption key is formed by the last N_K bytes of the key material computed during encryption. In consequence the initial decryption key can be obtained either by executing the typical encryption key schedule and selecting the last N_K bytes, or by being supplied from the user as decryption's initial key, similar to the case of encryption. The pre-computation solution implies significant algorithm latency for the initial key derivation phase, apart from the additional power cnsumption, and in consequence the decryption would require supplementary clock cycles for execution. If all the round keys are pre-computed and stored in a RAM, the decryption is greatly improved only if the same initial encryption key is used (all the round keys are already computed and need only be correctly selected); in all the other cases no speedup is gained, on the contrary.

4.2 AES Architecture

Synthesizing the observations presented in this thesis as well as other information provided by literature with respect to AES hardware realization, we developed an AES architecture based on the observations presented in the plenum of the "The Claude Shannon Workshop for Coding & Cryptography" [71]. The datapath's architecture is described in Fig. 4.8 and is capable of performing both encryption and decryption. With respect to AES's hardware design the datapath structure presented in Fig. 4.8 is our contribution regarding high speed cryptochip's implementation. The structure in Fig. 4.8, however, do not uses the combined MixColumns/InvMixColumns unit for reduced overhead as already presented, in order to simplify the control logic. Furthermore as it can be observed, the SubBytes and InvSubBytes modules share the multiplicative inversion modules, but in order to avoid the two multiplexers required for implementing both transformation in a single block and using a single inversion unit (as detailed in [105]) the affine transformation and the inverse affine transformation were separated out and duplicated into the datapath in order to obtain smaller latency both for encryption and decryption. This was achieved by balancing the amount of activity on both cipher operations. This can benefit the cipher's resistance against side-channel attacks by exhibiting similar power consumption for both operations. Additionally, the operations succeeding the multiplicative inversion stage are separated into two distinct sub-datapaths because of the modified decryption algorithm, presented in Fig. 4.7, which performs the mixing transformation and key addition differently in encryption and decryption.

Another observation pertains to the selection of final result: in order to avoid the multiplexer on the AES' encryption datapath branch required to prevent

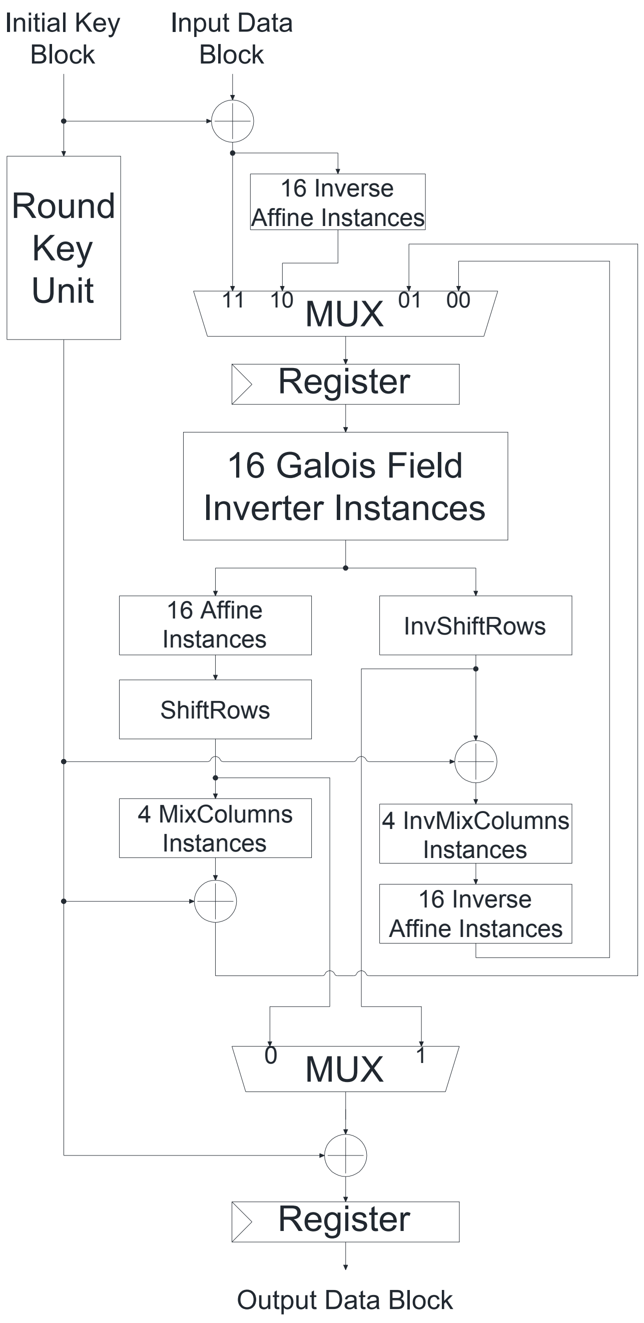


Figure 4.8 High speed AES datapath

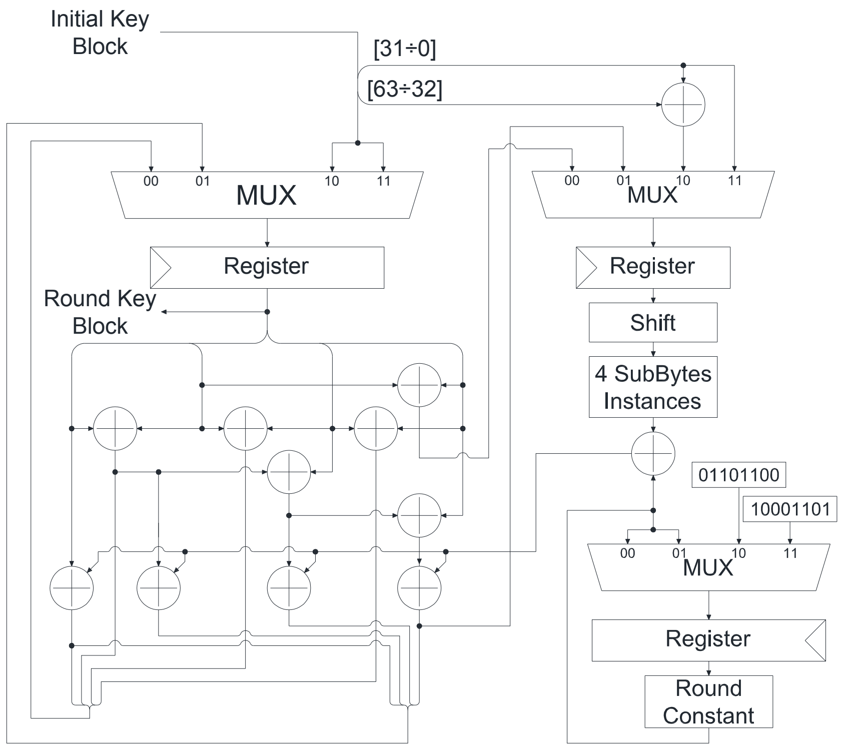


Figure 4.9 High speed AES key unit

the execution of MixColumns in the final round, a supplementary addition unit was inserted. The key scheduler design is graphically represented in Fig. 4.9. The modules performing finite field addition on 32-bit vectors, as defined in [98] were duplicated in order to avoid the high latency and area penalties incurred by typical multiplexers, selecting the appropriate addition terms for encryption and decryption respectively.

An less customized AES architecture, similar to the structure presented in Fig. 4.8 and Fig. 4.9, targeting the Cyclone II FPGA platform was detailed in the paper "*A High-Speed AES Architecture Implementation*" presented at the *ACM International Conference on Computing Frontiers*, 2010 [106].

Chapter 5
On the testable design solutions applied to Advanced Encryption Standard

This chapter presents our proposed testable design methods applicable to a high speed AES hardware design, as well as a particular on-line and off-line test strategies targeting the AES' multiplicative inversion module. A brief description of the related work concerning AES design with test facilities is also presented. The existing approaches serve as a motivation and justification for the importance of adding testability measures to the AES: both on-line and off-line. The chapter continues with detailing the proposed test mechanisms: one regarding an on-line Built-In Self Test method to detect faults in AES' datapath unit, another one analyzing the applicability of the off-line BIST architecture to the multiplicative inversion unit and the last one constructing an on-line error detection strategy for protecting the AES round. The chapter will finally draw a conclusion regarding the applicability of the presented methods for the on-line and off-line testing of AES.

5.1 Related Work on AES Testability

When reviewing the literature, one can observe a vivid interest for enhancing an AES design with testability solutions. The majority, as will be presented, employs on-line error detection techniques, however, there are approaches targeting the off-line domain as well. With respect to the on-line testability, there are references tackling the problem of concurrent checking for the AES's finite field inversion module. The presented references also emphasize the importance granted to testability in the context of AES implementations.

The mathematical definitions of SubBytes and InvSubBytes reveal the inversion module to be a component of the two transformations. As a consequence a test mechanism applicable to any of the three operations can usually be adapted for the other two [72]. With respect to the three non-linear operations of the AES, to the best of our knowledge, the majority of literature references treat the protection of the SubBytes transformation. However the mechanisms are expected to be adaptable for the inversion module also.

References explicitly considering test solutions for the inversion module are [107], [108], [109] and [110]. In [107] and [108] arithmetic relations are constructed linking the input and output of the inversion module. A similar relation is proposed for the SubBytes operation as well. The relations are described in terms of matrix operations, and permits predicting the parity bit of the output [107]. Both references construct a test mechanism independent on the particular implementation of the protected operations. Moreover the solutions are non-intrusive with respect to the AES round transformations. As already remarked, the solution in [107] is a parity based one, which groups the SubBytes with ShiftRows and the MixColumns with the AddRoundKeys, in order to make the test solution more competitive with respect to its area complexity. The parity detection mechanism for the SubBytes and InvSubBytes is generalized from the inversion module's parity protection solution [107]. The code-based parity solution of [107] was detailed by the same authors in [111] also, where more specific details of the architecture are presented. Moreover the approach in [111] considers the Galois field multiplicative inversion to be implemented using a composite field approach [104].

The parity-based concurrent error detection was also considered in [112] and [113]. The solutions presented in these two references extend the SubBytes with a single parity bit for error detection. In [112] the reconfigurable design principle is employed: a "data cell" detected to be faulty is avoided and "replaced" by a backup one. It worth noting that the reconfigurable and parity based solutions were applied to the highly modular AES design, presented in [114]. Moreover the effort of the authors came as an extension to their analysis of parity-based error detection procedures for the AES, presented previously in [58]. The parity-oriented error detection described in [58], was further developed in reference [113] by taking into account the observations from [115] regarding the error propagation properties in the context of AES implementations.

Yet another parity concurrent checking solution is described in [116] in which both the parity of the SubBytes' output is predicted from its input as well as input's parity is predicted from module's output. The solution is shown to offer a better fault coverage than the approaches employing only output parity prediction. The FPGA implementation of a single parity bit protection for SubBytes is detailed in [117].

The reference [109] proposes two scenarios for non-linear operations' protection. In the first approach the InvSubBytes and the SubBytes modules are considered to be present in the architecture, and while one of them is actively used by the datapath, the second one is idle and can be used for error detection. The result of the SubBytes is subsequently processed by the InvSubBytes and the final result compared against SubBytes' input. This is a form of hardware duplication necessitating specific control logic and multiplexers for directing the SubBytes result to the InvSubBytes module. However this solution can also be implemented as a form of time-redundancy concurrent checking as previously presented. The second approach presented in [109] considers the specific protection of multiplicative inversion modules. The operation is implemented using lookup tables: the ROM stores the sum between inversion's output and input. The second solution can be showed to cover only partially the possible faults.

In [110], the first protection mechanism targets the finite field multiplicative inversion, relying on the property of inversion that multiplying the module's result by the module's input yield the multiplicative neutral element of the field. If the inversion's input is the neutral element of field's addition operator (the so-called field's zero) the multiplication of module's input and output yield zero. In order to avoid the inherent area complexity of a complete field multiplier, the authors considered computing a limited number of the result bits. They observed that only 2 bits are sufficient for their intended fault coverage. This approach does not follow the conventional on-line design strategies as presented previously, however it belongs to the hardware redundancy measures. The protection mechanism is differentiated for the linear and non-linear blocks: the non-linear SubBytes being protected by the multiplier solution already detailed and the linear operators being protected by linear codes constructed as systematic codes for which the redundant data is computed by summing the 4 bytes of each column (a parity-based detection mechanism). Yet another solution is presented in [110] pertaining to the code-based approaches, which uses “non-linear (cubic) robust error detecting codes”. It collectively protect an entire AES round, however it requires the predictor to be constructed, which in the case of AES round implies considerable area complexity due to SubBytes’ non-linearity. The predictor also implements signature compression by means of addition in the finite field. Consequently a supplementary module is required at the round’s output to compute the same signature. Both the predicted and the actual signatures are raised to the power of 3 in the finite field (cubic code), and the checkers compares the two results.

Another hardware duplication solution is presented in [100], in which every group of 4 SubBytes is extended by an additional one, used for correctness verification. Only one of the 4 SubBytes can be tested each round by properly selecting the routes through the input multiplexers. Different testing scenarios can be constructed by properly controlling the multiplexers.

Another hardware redundancy method is presented in [118] in which distinct encryption and decryption datapaths were designed within the AES chip. While the chip performs an encryption, the decryption datapath is used to subsequently process the result from encryption’s flow, and compare the obtained result with encryption’s input. It is evident that this solution requires a careful control mechanism in order to accommodate the normal operation and the test. In the same article various error detection levels are discussed together with their associated implementation overhead and latency.

Moreover, the reference [119] presents a hybrid protection scheme, consisting of a parity-based detection mechanism for the linear elements and a hardware redundancy approach for the non-linear (SubBytes, in essence) modules. The approach, as the authors admit, can be interpreted as a partial circuit duplication [80].

5.2 Concurrent Error Detection for AES Round

The error detection method to be presented is based on reference [90]. It represents a typical on-line BIST solution for enhancing AES architecture with non-intrusive error-detection mechanisms based on parity prediction of AES’ round output. The fault detection is performed at the round level in order to reduce the complexity of the added BIST circuitry. The parity control mechanism relies on cyclic redundancy codes (5,4) associating a parity byte to every 4 bytes of AES’ matrix. The proposed solution is compared with respect to performance, area requirements and power consumption with the basic architecture which synthesizes parity control solutions proposed by field’s literature, such as [109] and [58]. The two architectures differ mainly in the way parity bytes are computed from the state matrix, distinction from which derives the other differences. Our proposed solution exhibit a parity prediction channel completely decoupled from AES’ datapath, which correspond to the classical definition of a parity prediction checker as found in [120].

The *basic structure* depicted in Fig. 5.1, associates a parity byte to each column of the AES state matrix. It makes use of the solution already presented in other references for implementing an efficient parity control mechanism for the AES. As outlined in [109], AES’ datapath can accommodate cyclic redundancy codes of various redundancy levels: from CRC(17,16) up to CRC(5,4). However, apart from the higher detection capability of the CRC(5,4), both the MixColumns and the round key generation unit favors CRC(5,4) codes.

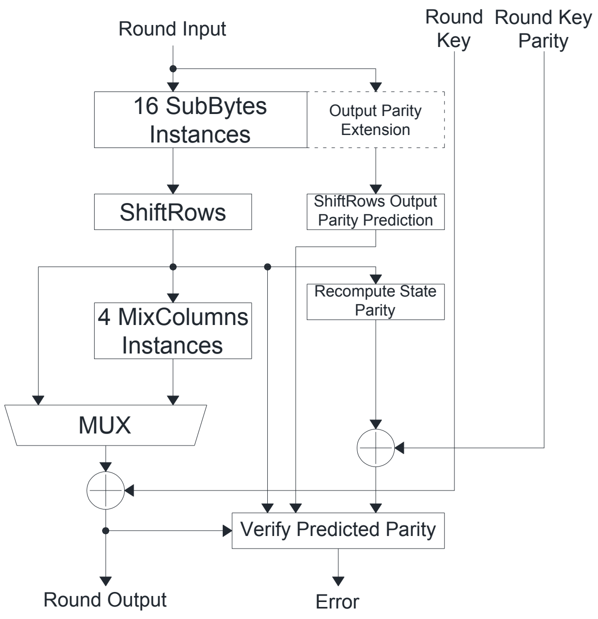


Figure 5.1 Basic on-line error detection strategy for AES round

It is also important to note that due to the AddRoundKey transformation being a binary operator, the same prediction scheme needs to be implemented for key unit in order to correctly predict the parity for round's output. The similar column-wise parity computation is easier to obtain due to key scheduler's compact structure. Moreover, because the key unit contains only a SubBytes instance, a shifting operation and addition in the finite field, the parity prediction for the key unit uses parity prediction modules already constructed for the operations on AES's datapath.

The efficiency goal sought when designing this architecture was to include a BIST mechanism as small as possible. We refer to the parity prediction channel as the sequence of parity module designed to computed the predicted parity of the output for each AES transformation. The typical solution is to cascade all prediction modules in order to obtain the final round parity. Whenever the parity prediction channel is hindered by operations in AES datapath for which the parity prediction cannot be integrated into the channel (due to its different computation), the extra control logic and combinational logic required to reconcile the two different parity predictions will in fact increase size and complexity of the error detection mechanism.

Analyzing the four AES transformations, it is observed that for column-wise parity, ShiftRows transformation requires a totally different prediction method, because it operates on state matrix lines. The simplest solution is to add all predicted parity bytes before executing ShiftRows and comparing the result with the added parity obtained from all 16 bytes of the state after the operation was performed. In addition, the column-wise parity bytes need to be re-computed from the state matrix after ShiftRows' application. The inability to construct a completely decoupled parity control channel differentiates the basic solution from the non-intrusive detection principle. This observation has impact over the effective error detection capability of the scheme: when verifying correctness for ShiftRows, a comparison between the predicted parity byte and the current parity byte is performed. In consequence the level of redundancy for ShiftRows verification is one fourth of the redundancy employed for the other transformation.

One advantage of the basic architecture is a mathematical relation that permits trivial computation of parity prediction for the MixColumns and its inverse. More precisely, in [109] is demonstrated that MixColumns do not affect column-wise parity bytes. This is the reason for no parity prediction module in the parity channel associated to the transformation as observed in Fig. 5.1.

Regarding the error resilience policy, when detecting an erroneous condition, the control logic halts algorithm's execution, clears the storage elements and sets an error indicator.

The issue imputed to any parity-based error detection relates to its inability in detecting an even number of errors. However, in this case the function to be protected using parity prediction, namely the AES round, is constructed based on the diffusion property as already remarked. This property manifests in that the influence of each input byte is spread across as many state matrix bytes as possible. Even when taking into consideration only a single round, the effect is dispersed over entire matrix as a result of the ShiftRows and MixColumns operations. As a result the effect of a single fault is distributed within the state matrix. In consequence for multiple faults, the error diffusion creates the premises for error detection. In fact reference [109] computed the fault coverage for an architecture similar to the one in Fig. 5.1 obtaining a detection rate higher than 98%.

The *proposed structure*, graphically described in Fig. 5.2, employs the same amount of redundancy as the previous architecture, organized in a different manner: a parity byte is associated to each state matrix' row.

The first observation would be that ShiftRows does not affect the parity channel anymore as the parity bytes are associated row-wise which is also how the transformation affect the state matrix. The second observation is that in this configuration, the ShiftRows' prediction is trivial and requires no dedicated module.

The MixColumns benefits no more from the mathematical property of not interfering with the parity bytes. However, it can be showed that the parity bytes can be predicted by applying the MixColumns transformation to the input parity bytes.

Considering the state matrix defined as in (18), the parity bytes computation for the proposed architecture is expressed in (19).

$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \sum_{i=0}^{3} \begin{bmatrix} b_{4i} \\ b_{4i+1} \\ b_{4i+2} \\ b_{4i+3} \end{bmatrix} \tag{19}$$

In the equations to follow a prime symbol represents the new value obtained after applying an AES transformation, both for regular bytes and for the parity bytes.

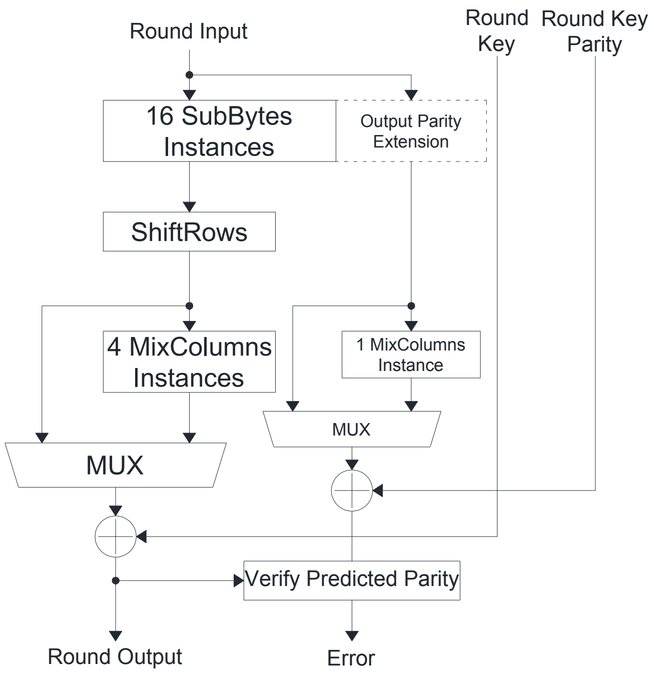


Figure 5.2 Proposed on-line error detection strategy for AES round

The parity bytes of the state matrix, after the MixColumns has been applied are expressed in (20). The effect of the MixColumn transformation over a column of the state, as described in [98], can be expressed as in (21).

$$\begin{bmatrix} p'_0 \\ p'_1 \\ p'_2 \\ p'_3 \end{bmatrix} \equiv \sum_{k=0}^{3} \begin{bmatrix} b'_{4i} \\ b'_{4i+1} \\ b'_{4i+2} \\ b'_{4i+3} \end{bmatrix} \tag{20}$$

$$\begin{bmatrix} b'_{4i} \\ b'_{4i+1} \\ b'_{4i+2} \\ b'_{4i+3} \end{bmatrix} \equiv \begin{bmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{bmatrix} \times \begin{bmatrix} b_{4i} \\ b_{4i+1} \\ b_{4i+2} \\ b_{4i+3} \end{bmatrix} \tag{21}$$

From (20) and (21), after reduction, we compute the parity bytes associated to the transformation's output in equation (22)-(23)

$$\begin{bmatrix} p'_0 \\ p'_1 \\ p'_2 \\ p'_3 \end{bmatrix} \equiv \sum_{k=0}^{3} \left(\begin{bmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{bmatrix} \times \begin{bmatrix} b_{4i} \\ b_{4i+1} \\ b_{4i+2} \\ b_{4i+3} \end{bmatrix} \right) \tag{21}$$

$$\begin{bmatrix} p'_0 \\ p'_1 \\ p'_2 \\ p'_3 \end{bmatrix} = \begin{bmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{bmatrix} \times \sum_{k=0}^{3} \begin{bmatrix} b_{4i} \\ b_{4i+1} \\ b_{4i+2} \\ b_{4i+3} \end{bmatrix} \tag{22}$$

$$\begin{bmatrix} p'_0 \\ p'_1 \\ p'_2 \\ p'_3 \end{bmatrix} = \begin{bmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{bmatrix} \times \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{23}$$

The result in (23) support our initial assessment that the parity of the MixColumns output can be predicted by applying the transformation to the parity bytes of the inputs. The same property can be shown to stands for InvMixColumns also.

Additionally, the parity prediction for key will be different for the proposed solution compared to the basic one. For computing the parity prediction mechanism for the key unit, we consider the bytes of the round key organized as in (24) and the associated parity bytes expressed as in (25).

$$\begin{bmatrix} rk_0 & rk_4 & rk_8 & rk_{12} \\ rk_1 & rk_5 & rk_9 & rk_{13} \\ rk_2 & rk_6 & rk_{10} & rk_{14} \\ rk_3 & rk_7 & rk_{11} & rk_{15} \end{bmatrix} \tag{24}$$

$$\begin{bmatrix} p_{rk_0} \\ p_{rk_0} \\ p_{rk_0} \\ p_{rk_0} \end{bmatrix} = \sum_{i=0}^{3} \begin{bmatrix} rk_{4i} \\ rk_{4i+1} \\ rk_{4i+2} \\ rk_{4i+3} \end{bmatrix} \tag{25}$$

Taking into consideration that the parity prediction for the operations of the key generation process were already treated and by following an approach similar to the one presented above, the predicted parity bytes for the round key in encryption and decryption are obtained as in (26) and (27) respectively.

$$\begin{bmatrix} p'_{rk_0} \\ p'_{rk_1} \\ p'_{rk_2} \\ p'_{rk_3} \end{bmatrix} = \begin{bmatrix} rk_4 \\ rk_5 \\ rk_6 \\ rk_7 \end{bmatrix} + \begin{bmatrix} rk_{12} \\ rk_{13} \\ rk_{14} \\ rk_{15} \end{bmatrix} \tag{26}$$

$$\begin{bmatrix} p'_{rk_0} \\ p'_{rk_1} \\ p'_{rk_2} \\ p'_{rk_3} \end{bmatrix} = \begin{bmatrix} rk_{12} \\ rk_{13} \\ rk_{14} \\ rk_{15} \end{bmatrix} + \begin{bmatrix} SubBytes(rk_{13}) \\ SubBytes(rk_{14}) \\ SubBytes(rk_{15}) \\ SubBytes(rk_{12}) \end{bmatrix} + \begin{bmatrix} Rcon(i) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{27}$$

By analyzing (26) and (27) it follows that the size of the parity prediction modules for the key unit in the proposed scheme is smaller and much simpler to implement compared to the one used by the basic architecture, especially for the encryption.

Although the parity control mechanism for the proposed architecture incorporates the complex MixColumns transformation, it requires no parity bytes re-computation and also requires no prediction for the ShiftRows transformation. In addition the complexity of the key unit error detection scheme is significantly reduced as the experimental results reveal. The error response policy is identical to the one employed in the basic structure. It is worth noting that because of the same level of redundancy and the diffusion property of the AES round which manifest both column-wise (in MixColumns) and row-wise (in ShiftRows), the observations for parity error detection efficiency with respect to detecting an even number of errors, mentioned for the basic structure, are also valid for the proposed architecture.

Experimental results are offered in [90] regarding the synthesis of the basic and proposed architecture. Both on-line BIST solutions were applied to AES encryption, AES decryption and encryption-decryption capable AES designs.

	AES Design	**Basic CED Design Overhead**	**Proposed CED Design Overhead**
AES Round Unit			
Combinational	305056	29412	25380
Registers	22528	0	0
AES Key Unit			
Combinational	76924	10024	2856
Registers	23936	5632	5632
Total	428444	45068	33868

Table 5.1 Area overhead entailed by the on-line parity based error detection architectures [μm^2] from [90]

	Non-CED Design	**Basic CED**	**Proposed CED**
AES Encryption	35.3	41.3	39.1
AES Decryption	39.4	45.1	45.2
AES Encryption-Decryption	55.5	67.3	65.7

Table 5.2 Power consumption associated with the on-line parity based error detection architectures [mW] from [90]

		Non-CED Design	**Basic CED**	**Proposed CED**
AES Encryption	Round Unit	4.55	5.69	5.93
	Key Unit	3.80	4.34	3.88
AES Decryption	Round Unit	5.04	6.13	5.96
	Key Unit	3.5	3.95	3.60
AES Encryption-Decryption	Round Unit	4.53	6.51	6.42
	Key Unit	4.97	5.40	5.02

Table 5.3 Latency entailed by the on-line parity based error detection architectures [ns] from [90]

Table 5.1 presents the area investment for including the two BIST solutions into an encryption AES design. The incurred overhead is provided with respect to the size of the encryption implementation. The overhead of the basic architecture, as revealed by Table 5.1 is 10.52%, while for the proposed solution it is of only 7.90%.

Table 5.2 summarizes the power consumption obtained by including error detection mechanisms into the three AES designs. The power consumption was estimated for a 100MHz clock frequency. The influence over performance of the BIST strategies is evaluated in Table 5.3 by computing the delay on the critical path.

5.3 Analysis of Built-In Self Test Applicability for AES Non-linear Transformations

This section deals with the error detection analysis and testability measures presented in [72]. One of the advantages offered by an off-line BIST mechanism, as mentioned in [72], is its ability to detect unmodeled faults and timing-related defects [33], [86]. The tree non-linear operations targeted by this solution are: SubBytes, InvSubBytes and multiplicative inversion in the Galois Field associated to AES algorithm. Although neither the AES encryption nor decryption explicitly use the inversion operator, the reference details the reason for include it in the list, as related to efficient hardware implementation for encryption-decryption AES designs as already suggested in this thesis. The typical BIST architecture used in this approach is the off-line design presented in Fig. 3.6.

Analyzing the related work on AES testability, and the special literature, lead us to the conclusion that pseudo-random Built-In Self Test was not yet thoroughly analyzed in the specific context of AES. To the best of our knowledge, the three non-linear transformations were not addressed by pseudo-random stimulation coupled with signature-based error detection. The reduced size of the added BIST mechanism linked with its simple control logic makes this solution a perfect candidate for secure testing.

The main objectives during designing and analyzing the proposed solution were to obtain complete single stuck-at fault coverage and error detection rate as high as possible for multiple stuck-at faults. Moreover, due to the fact that a deterministic selection of the most appropriate test pattern generator and signature analyzer is considered by the literature to be a rather very complex problem (to the best of our knowledge no generic approach exists towards this direction), we guided our selection process by simulations.

Another important issue to be mentioned is the applicability of the proposed solution to any of the three non-linear functions, regardless of their particular implementation details. This, in fact, is generally true for BIST solutions, unlike the error detection solutions which are subjected by the particular structural details of the design to be protected (intrusive test mechanisms).

When dealing with protection strategies for testing devices' integrity, as already discussed, one simple question can arise, as to "who checks the checker". This problem, apart from its pragmatic context, lead to the development of the so-called self-checking checkers [120], [121]. In fact, in [68], the author graphically details the self-checking capabilities for the LFSR and MISR solutions in an attempt to address the above inquiry.

Another important observation reveals that the probability of a fault affecting a module in a design depends on the complexity (the size) of the module relatively to the entire architecture. Firstly, the above observation represents the driving mobile for addressing testability on the context of the non-linear operations, because, as detailed in [100], in a high speed AES encryption design, the 20 SubBytes instances together with the registers claim as much as 85% of the circuit's area. Although for InvSubBytes and inversion the percentage would be less (no more than 16 instances of the two are required in any design), the initial observation stands as the two operations together with SubBytes are the top three most complex operations, being in consequence the most likely to be affected by a fault.

A second consequence of the fact that the fault probability of being affected is higher for complex modules is that by using simple testability mechanism with reduced area, the probability of a fault affecting the checker is significantly diminished. This was another criterion for selecting the testability mechanism in [72].

The test generation mechanism we took into consideration was the *Linear Feedback Shift Register* mechanism because of its simple structure and reduced size. Moreover, due to the off-line nature of the test, there are no necessarily imposed conditions on its speed, so that, because of the reduced number of inputs in the modules to be tested (8 bits for all three operations), the entire LFSR-generated sequence was taken into consideration. However, we also compared the performances of LFSR-based test generators with respect to other alternatives. A similar simple structure, but with very limited pseudo-randomness, having a test sequence only one element longer than LFSR's is a binary counter.

As opposed to the above mentioned test generation solutions, a more complex approach is to stimulate the modules with a specially built pattern generator that would offer at its output a test vector sequence deterministically constructed by an Automatic Test Pattern Generator. Such a design could include a ROM for storing the patterns or a custom built *Finite State Machine* sequencing at its output the respective test vector set. The deterministic test set was obtained using ATALANTA tool [45]: the ATPG program was applied to synthesized designs of the three non-linear operations to be tested. The difference in the test process between the LFSR-based stimulation and the deterministic approach are evident when comparing the lengths of 255 for the LFSR sequence with the length of 152 and 154 of ATALANTA's generated test sets for the inversion module and the other two transformations respectively.

Another aspect worth noting and mentioned in [72] relates to the N-detect single stuck-at fault property of the test set generated by LFSR and the binary counter. A N-detect single stuck-at fault test set is considered in [33] "a better metric for defect coverage compared to the traditional stuck-at fault coverage". In [72] is formulated a justification regarding the LFSR and the binary counter sequences being N-detect test sets.

The next problem faced when adopting a LFSR test method is choosing the appropriate characteristic polynomial. At first, this has little importance, as for any degree-8 primitive polynomial, the entire 255-element sequence will be generated. However, a thoroughly analysis and experimental results revealed, that the generation polynomial has an important impact over the compaction scheme. The quest for analytically determining the most appropriate characteristic polynomial, for test generation and signature analysis as already state is beyond the scope of this work and the work performed in [72]. We consider the simulation approach to be a more manageable solution in this context.

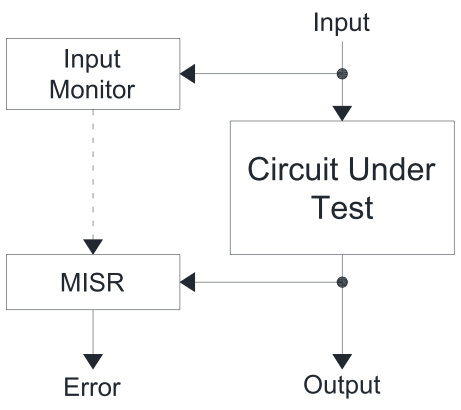


Figure 5.3 On-line adaptation for a typical off-line BIST architecture

As already anticipated, the compaction scheme adopted in [72] is the *Multiple Input Signature Register* because of its reduced area requirement, making it less probable to be affected by faults. The selection of the characteristic polynomial was directed by simulations aiming toward the objective of 100% single stuck-at fault coverage and multiple stuck-at faults error detection higher than 90%. As it turned out, there is no 8-stage MISR capable of completely detecting all single stuck-at faults conditions when stimulating the circuit with the test sequence of a LFSR, a binary counter or ATALANTA's deterministic set. However there are 9-stage MISRs capable of accomplishing this requirement.

Reference [72] also presents a particular on-line BIST solution. The approach is different from the typical error detecting code solutions, in that it uses signature compaction as a mechanism to verify the integrity of *Circuit Under Test*. Such an approach is described in Fig. 5.3. In order to adapt the off-line BIST principle already presented, into a concurrent process, the test pattern generation unit is no more required. As part of the AES datapath, the three transformations receive inputs during circuit's normal operation. In consequence an *Input Sequence Monitor* is replacing the test generation, in order to track the CUT's inputs and detect those vectors relevant for the compaction scheme.

However the detection process for the on-line BIST has a probabilistic nature in that, for a complete signature to be obtained the number of cycles required are non-deterministic because it depends on the specific input vectors received from AES's datapath. This is due to the statistical independence of a cryptographic algorithm's outputs with respect to its inputs (data block and key). In other words, although knowing the inputs of the algorithm, its outputs can be regarded as pseudo-randomly generated. At a smaller scale this is also the case for a single round's execution.

The error control consists in comparing the obtained signature with the expected one for both the off-line and the on-line structure. A special observation targets the on-line solution: when trying to avoid the concurrent test process, the attacker might attempt to inject a fault on the error detection line. However, due to the already discussed probabilistic nature for test completion and the fact that the precise internal structure remains unknown, the attack does not guarantee that the test module was in fact disabled. Moreover, by using two-rail encoded error indicator, the probability of the verifier to be disabled is greatly reduced.

The experimental result in [72] are relevant for the process of LFSR and MISR characteristic polynomial selection. An in-house fault simulator was constructed in order to effectively evaluate the fault coverage for single stuck-at faults situations and the aliasing probability for multiple stuck-at faults. The three non-linear transformations were described using VHDL and implemented using a sum-of-products approach.

	Gate Count	**Logic Levels**	**Collapsed Faults Number**
SubBytes	651	17	1798
InvSubBytes	680	15	1856
Inversion Module	657	16	1765

Table 5.4 Implementation details for the three non-linear AES operations from [72]

In Table 5.4 is presented a summarized description of the three designs regarding the complexity expressed in number of gates and logic levels, as well as the number of collapsed faults determined by ATALANTA tool.

As already mentioned, the MISR configuration capable of detecting all single stuck-at faults has 9 stages, i.e. only 8 stages will receive input from the modules to be tested. In consequence for each characteristic polynomial, there are 9 possible MISR architectures. The total number of the 9-stages MISRs as revealed in [72] is 432. During the simulations, the connection between the LFSR structure and the MISR compactor became even more evident, justifying once again the use of simulation for BIST design.

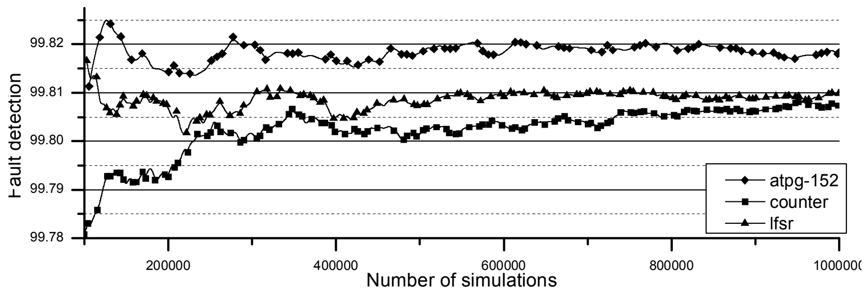


Figure 5.4 Error detection rate for 2 stuck-at faults injected into inversion unit from [72]

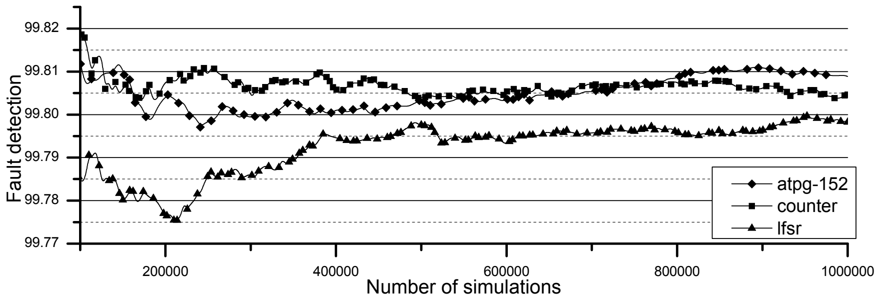


Figure 5.5 Error detection rate for 5 stuck-at faults injected into inversion unit from [72]

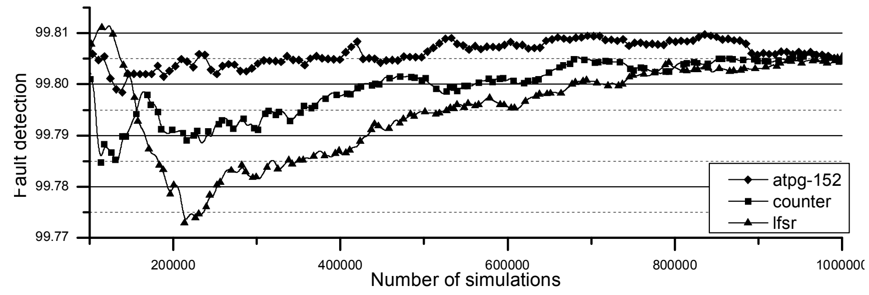


Figure 5.6 Error detection rate for 10 stuck-at faults injected into inversion unit from [72]

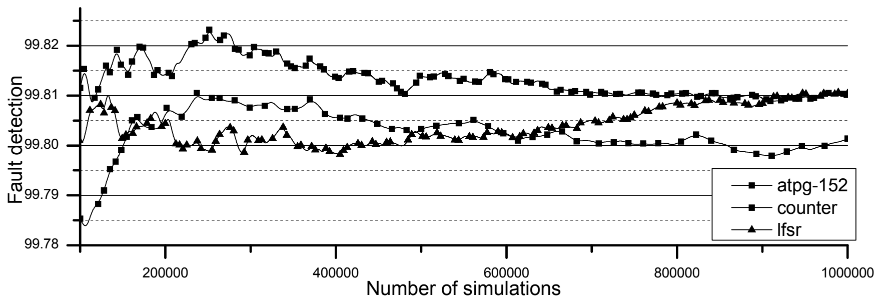


Figure 5.7 Error detection rate for 50 stuck-at faults injected into inversion unit from [72]

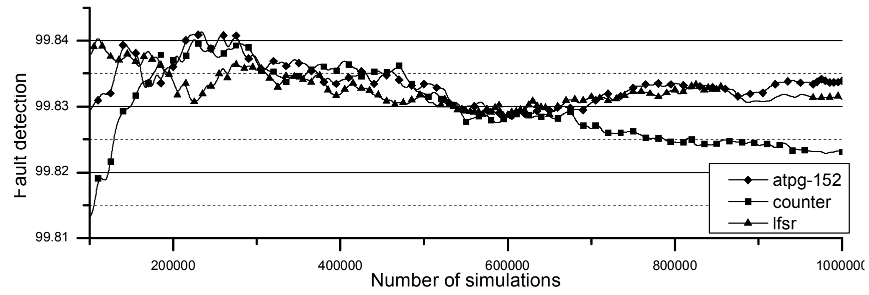


Figure 5.8 Error detection rate for 100 stuck-at faults injected into inversion unit from [72]

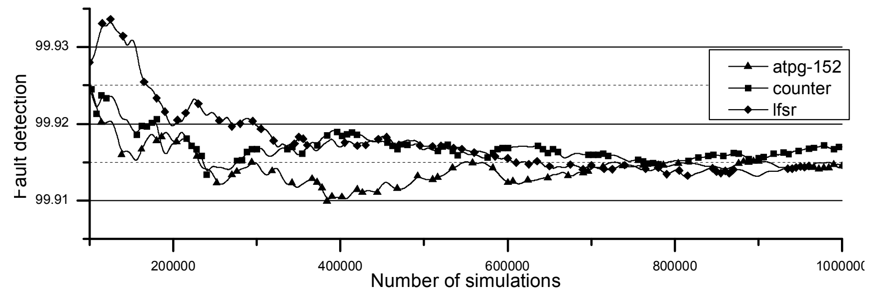


Figure 5.9 Error detection rate for 250 stuck-at faults injected into inversion unit from [72]

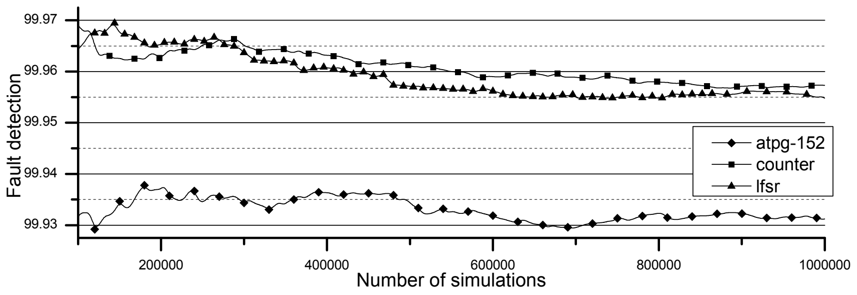


Figure 5.10 Error detection rate for 500 stuck-at faults injected into inversion unit from [72]

Reference [72] presents also the simulation results for multiple stuck-at faults affecting the inversion module which reveal an error detection rate higher than 99.78%. In Fig. 5.4, Fig. 5.5, Fig. 5.6, Fig. 5.7, Fig. 5.8, Fig. 5.9 and Fig. 5.10 are presented the detection rates for a particular LFSR-MISR pair when injecting 2, 5, 10, 50, 100, 250 and 500 stuck-at faults into the inversion unit.

5.4 Concurrent Checking for AES' multiplicative inversion

The concurrent error detection mechanism we propose in this section targeting the AES multiplicative inversion operation is detailed in [122]. For the experimental results we used the AES architecture from [106].

In a dual encryption-decryption AES design, the sharing of the field inversion element is an important area saving design decision. In a high speed design, allowing to concurrently processing all 16 bytes of the data block this decision allows alleviating the complexity of having 16 InvSubBytes and 16 SubBytes modules embedded in the architecture. However, this decision will have an effect over the implementation latency because of the cumulated delay associated to the affine transformation and the inversion module. The effect of the multiplexers selecting one of the two operations can be diminished by using an AES datapath design similar to the one in Fig. 4.8 for which the AES datapath is split, after the SubBytes or InvSubBytes operations are executed, into two distinct sub-paths.

The testing mechanism relies on the following mathematical property of the multiplicative inversion in the finite field:

Proposition: Let $\alpha,\beta \in GF(2^m)$, with $\alpha \neq 0$, $\beta \neq 0$, $\alpha \neq \beta$ and $\alpha \neq \beta^{-1}$. The following expression stands:

$$\alpha + \alpha^{-1} \neq \beta + \beta^{-1} \tag{28}$$

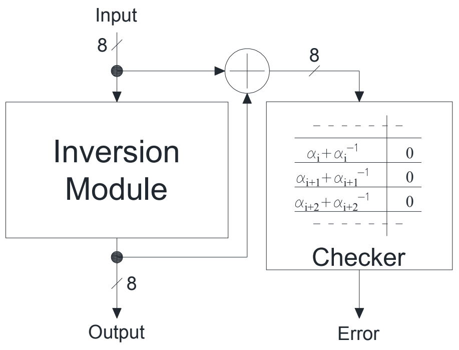
Proof: The conditions $\alpha \neq 0$ and $\beta \neq 0$ assures that α and β are invertible in the Galois Field [123]. The demonstration is constructed as a proof by contradiction. Considering (28) to be false and multiplying both equation members by the factor $\alpha\beta$, the following equalities are obtained:

$$\begin{aligned}
&\alpha^2\beta + \beta = \alpha\beta^2 + \alpha \\
&\alpha^2\beta + \beta + \alpha\beta^2 + \alpha = 0 \\
&\alpha\beta(\alpha + \beta) + (\alpha + \beta) = 0 \\
&(\alpha\beta + 1)(\alpha + \beta) = 0 \qquad (29)
\end{aligned}$$

The equation (29) implies that either one of the $\alpha\beta + 1$ and $\alpha + \beta$ factors or both are zero. This however contradicts the proposition's hypothesis because $\alpha + \beta = 0$ implies $\alpha = \beta$ and $\alpha\beta + 1 = 0$ implies $\alpha = \beta^{-1}$. Thus the demonstration's assumption is false making the proposition true. □

The consequence of the above proposition, is that the elements of the abelian multiplicative group [123] consisting of all non-zero elements of the finite field can be grouped accordingly into 128 classes with respect to the relation $\alpha + \alpha^{-1}$ over the finite field. The inversion function is also defined for the addition's neutral element: the field element zero [98]. As a consequence for both field's element zero and one, the sum with their own inverse is zero.

The first verification architecture is represented in Fig. 5.11. It is a non-typical hardware redundancy concurrent checking mechanism, similar to the one presented in [110] in the sense that both methods rely on a mathematical property to evaluate correctness. Moreover the architecture in Fig. 5.11 pertains to the non-intrusive concurrent checking principle being able to protect the inversion module



Input
8
8
Inversion Module
$\alpha_i + \alpha_i^{-1}$ 0
$\alpha_{i+1} + \alpha_{i+1}^{-1}$ 0
$\alpha_{i+2} + \alpha_{i+2}^{-1}$ 0
Checker
8
Output
Error

Figure 5.11 First AES inversion module concurrent checking architecture

regardless of its implementation details. The truth table for the verifier (checker) is constructed by computing, for each field element, the sum with its own inverse, and considering only these sums to be correct when delivered at the checker's inputs.

The concurrent error detection architecture can also be embedded into a BIST structure, as Fig. 5.12 depicts. As opposed to the conventional BIST architecture the solution in Fig. 5.12 requires no response compaction module because the proposed on-line protection mechanism from Fig. 5.11 already generates a single GO/NOGO line. The BIST structure is only required to store the error indicator and transmit it to the control unit. In fact the error indicator buffering is optional: a correspondingly designed control unit would test the line at the appropriate moment, or even buffer the signal itself. A conventional BIST architecture, as the one presented in the previous section and in [72] attains a high detection probability only after the entire test input sequence was applied to the Circuit Under Test, which in this case is the inversion module. The smallest BIST latency, as already reported in the previous section is obtained when generating the input test vectors deterministically using ATALANTA [45]. The test latency is of 152 clock cycles in this case and 255 clock cycles when the test input vectors are generated using an 8-bit LFSR. As opposed to these latencies, the BIST architecture constructed around the solution in Fig. 5.11 offers a test latency of 4 clock cycles, because the structure detects permanent defects with multiplicity higher than 2 within 4 clock cycles, and single and double defects within only 1 clock cycle. The reduction in test latency is significant: from 152 to only 4 clock cycles.

In order to detect intermittent faults with higher multiplicity, such as the one injected in attack situations conducted using exposure to laser [15], a concurrent checking mechanism with a higher detection probability was constructed. The design principle relies on increasing the redundancy at the checker's level. Because the checker in the first solution uses no information pertaining to the inversion's input the aliasing problem can mask errors. However, by including information regarding the current inversion's inputs, the effect of aliasing can be greatly diminished as the experimental results confirm.

The checker will analyze the input vector information and the sum between this input and the inversion's output and decides for the correctness of the operation. In other words, compared to the first solution, the checker's truth table will contain also information from the input vector. The area of the checker needs to be smaller than that of the inversion module itself, in order to obtain an efficient concurrent checking mechanism, as already observed in the previous chapter. If all 8 bits of inversion's input are considered as redundant information, the resulting checker is similar in complexity with the multiplicative inversion module. As a consequence a condensed form of the input will be used by the checker. A signature is computed for each input vector and based on all the input vectors associated to each signature, the correct sums between those inputs and the corresponding inversion's outputs are evaluated. The signatures are then associated with all possible correct sums and based on this the checker's operation mode is defined. The signature applied to inversion's input is computed by spatial compaction, not by temporal compaction, because the signature is required in the same clock cycle. As a result, an EXOR tree was used compacting the 8 input bits into a smaller signature. We adopted in our experiments a signature length of 4 bits. Spatial compaction methods with the same signature size are expected to behave similarly. Moreover the signature size can be modified in order to adapt for the required fault detection. The second proposed concurrent error detection mechanism is presented

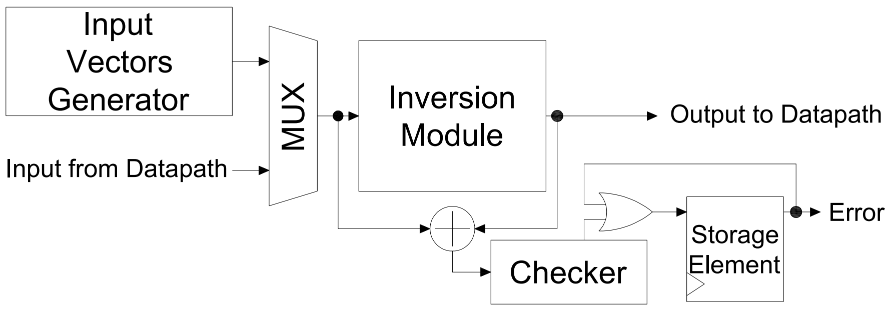


Figure 5.12 BIST constructed around the proposed concurrent checking architectures

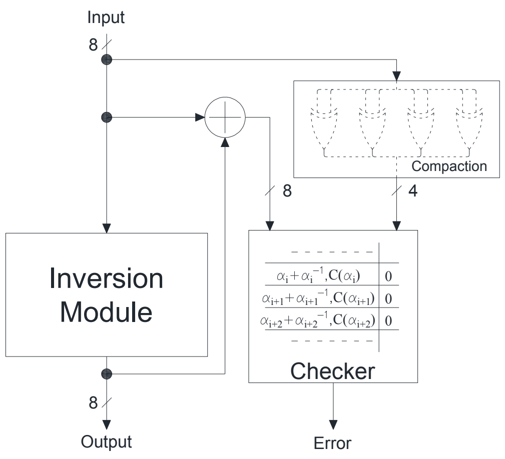

Input
8
Compaction
8
4
Inversion Module
$\alpha_i + \alpha_i^{-1}, C(\alpha_i)$ 0
$\alpha_{i+1} + \alpha_{i+1}^{-1}, C(\alpha_{i+1})$ 0
$\alpha_{i+2} + \alpha_{i+2}^{-1}, C(\alpha_{i+2})$ 0
Checker
8
Output
Error

Figure 5.13 Second AES inversion module concurrent checking architecture

in Fig. 5.13. The same off-line BIST mechanism can be constructed around the second architecture although the on-line mechanism detects with high confidence any defects intermittent or permanent.

When evaluating the error detection of the second concurrent checking mechanisms we simulated the injection of stuck-at faults of multiplicity 1, 2, 5, 10 up to 500 into a single AES round. We also simulated the injection of single stuck-open defects into a single AES round. The simulations of defects manifesting over a single round were performed using an in-house simulation framework. Each faulty condition was injected 10^6 times in order to obtain reliable detection coverage results. When evaluating the fault detection for defects manifesting for more than a single clock cycle (the base AES design we used in this experiment, described in [106], executes one round in one clock cycle) we used probability calculation. The detection rate can be interpreted as the probability that a fault is detected by the concurrent checking scheme. The undetected probability is defined in terms of the complementary probability. We computed the detection probability associated to the proposed architecture for faults affecting multiple clock cycles, using complementary probability calculation: we compute the probability for the fault to be undetected when manifesting over multiple cycles and by subtracting the obtained probability from 1 the probability of detecting that fault when it influences the module for multiple cycles is obtained.

Because of the AES' round transformations and the fact that each AES round is parameterize by a different round key, the input for the inversion module in each round can be considered pseudorandom. In fact experimental results show the probability distribution for each field element, for each of the state matrix' 16 bytes, at the start of a round is uniform making each element of the field equiprobable at the inversion's input. In consequence, the undetected probability for a fault affecting a single cycle is computed as the number of the undetected erroneous inversion's outputs obtained when considering all field elements as possible inputs in the presence of that fault. For the same fault, manifesting over 2 clock cycles, the events in which the defect remains undetected are those for which the error generated at the output of inversion module, in the first round and the erroneous output of the inversion in the second round are belonging both to the category of

	Without concurrent checking	**Hardware duplication**	**First concurrent checking architecture**	**Second concurrent checking architecture**
AES Encryption	506912.00	782452.00	563599.00	707171.00
AES Decryption	541400.00	815423.00	597322.00	741112.00
AES Encryption/ Decryption	620971.00	896083.00	679743.00	830283.00

Table 5.5 Area overhead for the proposed concurrent checking methods in comparison with the hardware duplication solution [μm^2]

undetected erroneous outputs. In other words the undetected probability is, in this case, the undetected probability of the fault affecting a single clock cycle raised to the power of 2. By induction, the probability of a fault to remain undetected when manifesting over *n* clock cycles is expressed as in equation (30) to be the probability of that fault to remain undetected when affecting a single cycle, raised to the power of *n*. From equation (30) the probability of a faulty condition to be detected when manifesting over *n* clock cycles is computed as in equation (31).

$$P^{n}_{undetected} = (P_{undetected})^{n} \tag{30}$$

$$P^{n}_{detection} = 1 - (P_{undetected})^{n} \tag{31}$$

The detection probability for the first solution when considering permanent stuck-at faults of multiplicity 1, 2, 10 and 100 is represented in Fig. 5.14, Fig. 5.15, Fig. 5.16 and Fig. 5.17. The behavior of the first architecture with respect to stuck-open faults is the same as for stuck-at faults: the detection rate of the design when a single permanent stuck-open affects the inversion module is depicted in Fig. 5.18. The performance of the second detection architecture with respect to the detection of intermittent faults manifesting during a single AES round is depicted in Fig. 5.19.

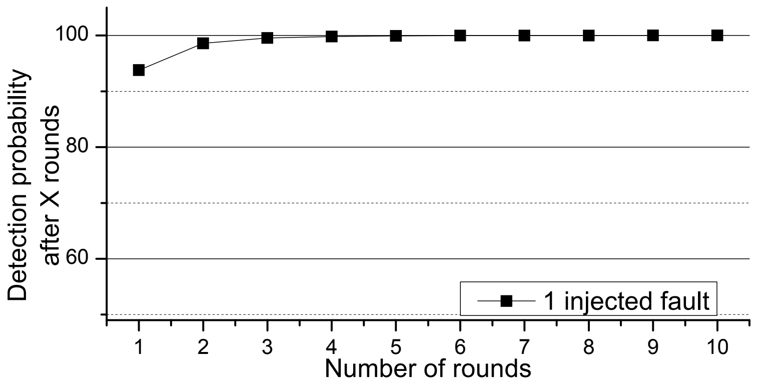


Figure 5.14 Detection rate for the first concurrent error detection architecture when 1 stuck-at fault is injected

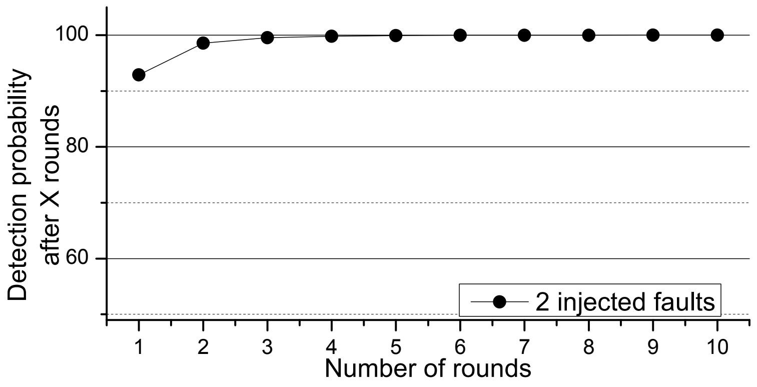


Figure 5.15 Detection rate for the first concurrent error detection architecture when 2 stuck-at faults are injected

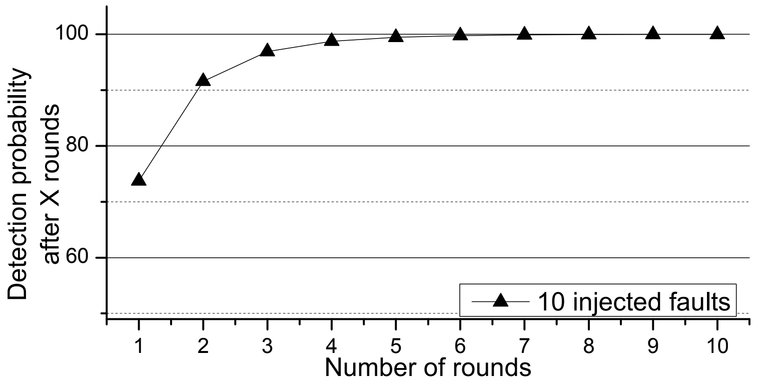


Figure 5.16 Detection rate for the first concurrent error detection architecture when 10 stuck-at faults are injected

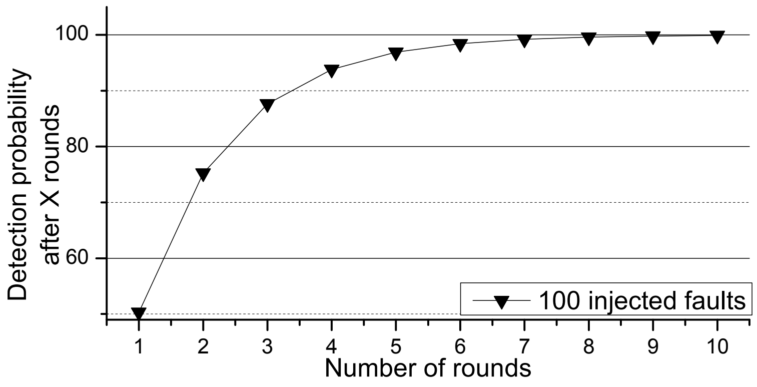


Figure 5.17 Detection rate for the first concurrent error detection architecture when 100 stuck-at faults are injected

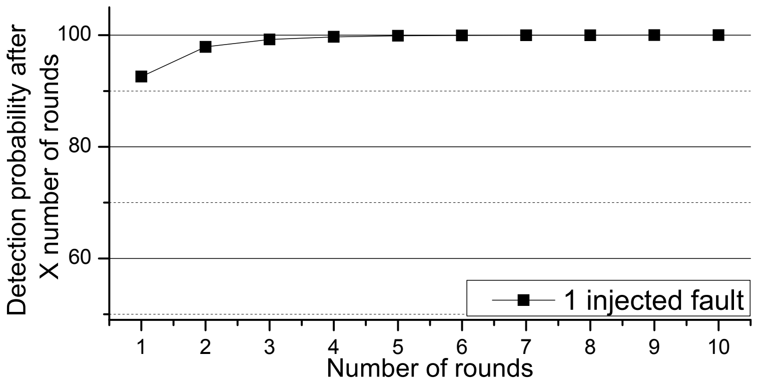


Figure 5.18 Detection rate for the first concurrent error detection architecture when 1 stuck-open fault is injected

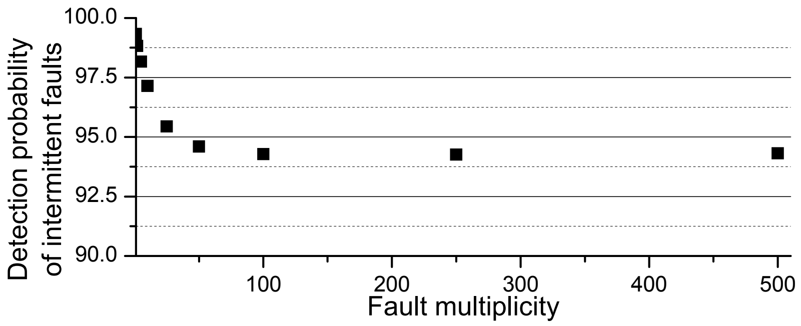


Figure 5.19 Detection rate for the second concurrent error detection architecture when multiple intermittent faults are injected

The area requirement for implementing the two proposed architectures is presented in Table 5.5. The AES design area without concurrent checking is presented in order to compute the overhead incurred by the proposed architectures as well as their efficiency compared to the hardware duplication method. The table reveals that for AES encryption-decryption the first method incur only 9.45% area overhead compared to the 44.30% incurred by hardware duplication. For the same AES design the second verification architecture incur an area overhead of 33.70% compared to the 44.30% of the hardware duplication method.

5.5 Conclusions

The chapter presented a thoroughly description of testing solutions adopted for the AES implementations as they are presented in the literature. A wide range of concurrent checking as well as off-line testing solutions was covered. It also presented our research and results concerning testability features applied to the AES hardware implementations.

The first solution makes use of the parity prediction error detection techniques, applied to AES's datapath and round key in order to obtain an efficient concurrent BIST design. Two designs were described: a basic one synthesizing the solution already existing in the field's literature and a proposed one, constructing the parity prediction channel in a non-intrusive manner in order to attain efficient and reduced implementation overhead. The proposed architecture, together with the mechanisms for predicting the parity for AES' various modules were detailed together with experimental results regarding the area overhead, power consumption and performance reduction implied by both the basic and the proposed models.

The second testing strategy was directed toward off-line BIST, with the goal to analyze the applicability of LFSR-MISR based error detection techniques in the context of the non-linear AES operations. The rationales for selecting a MISR configuration response compaction scheme is presented together with conclusive remarks drawn from the experimental results of simulating the off-line BIST design with single and multiple stuck-at faults. An on-line adaptation of the scheme is presented detailing also the design modification it implies.

Finally a concurrent error detection mechanism was constructed based on a mathematical property of the multiplicative inversion in finite fields. The presented architectures correspond to the on-line non-intrusive error detection mechanism. The first installation can detect single and double intermittent faults manifested as stuck-at or stuck-open faults. For faulty conditions of higher multiplicity it can detect permanent defects after no more than 4 AES rounds. For critical applications, the second solution was constructed. Although implying a higher area overhead, the detection capabilities are virtually the same to those of a hardware duplication mechanism, compared to which it still saves silicon area.

Chapter 6
Conclusions

The thesis undertakes the problem of secure testing in the context of cryptographic hardware implementations. Moreover, the thesis was constructed gradually, starting from the problems raised by defects in the VLSI technology. The mechanisms by which a defect affects a device were described as well as the manifestation characteristics for each fault model. The defects were traced throughout all design phases, detailing the correspondence between fault models and circuit description levels. At the gate level and below, the faults are characterized by a location, and for some fault models also by size. The mapping of lower level defects in terms of the stuck-at fault model was presented, together with the defective situations not mapped to or not covered by the stuck-at faults. For the physical situations not covered by the gate-level defects, the switching layer fault models were introduced: the transistor's stuck-open and short defects. Moreover the perspective of physical faults from the design's layout netlist was discussed. The bridging defects were presented as a generalization of the short faults together with the delay faults specific for high integration VLSI processes.

After consolidating the problematic of fault modeling, the thesis follows with analyzing the available solutions for achieving the goal of secure, autonomous testing. As a consequence the test engineering domain is briefly introduced, enumerating the conventional test strategies both for concurrent and non-concurrent error detection. The thesis describes in detail the challenges faced by the testing process and the appropriateness of Built-In Self Test methodology, both on-line and off-line, with respect to secure test. First it reveals the importance of testing from an economical point of view. The provisioned test challenges for the nanoscale integration process were given as yet another justification for our research. Finally the BIST methodology is described, and its various implementation methods, revealing the LFSR-based solutions to be the most appropriate secure testing approach in terms of fault coverage and hardware overhead. The conventional on-line testing mechanisms were also presented. The link with the fault models is established regarding the appropriateness of stuck-at fault model in modeling the behavior of intermittent faults. The distinction between the intrusive concurrent checking and non-intrusive concurrent error detection mechanism was accentuated, detailing also the advantages of a non-intrusive approach. The presentation of the self-checking architecture and the supplementary concurrent detection mechanism followed the same direction as the authoritative references from the literature. The hardware duplication, code-based concurrent checking and time redundancy solutions were presented. A special attention was given to the error detecting codes and their implications concerning the Circuit Under Test design, the predictor construction and the checker.

The particularities of the AES algorithm were presented also, with emphasis on its high speed implementations in order to exploit possible advantages with respect to implementing a secure test mechanism. Special attention was offered to AES' non-linear operations: being the most complex ones their efficient implementation has an important impact on system's overall performance. The decryption algorithm was detailed pointing the architectural issues that favor an encryption-decryption capable high speed AES design. The AES hardware implementation solutions were presented at the following scientific presentations:

- **F. Opritoiu**, M. Vladutiu: "Cryptochip implementations with Buil-In Self Test features applied to AES standard," The Claude Shannon Institute Workshop on Coding & Cryptography, May 19-20, 2008.
- **F. Opritoiu**, M. Vladutiu, L. Prodan, M. Udrescu: "A High-Speed AES Architecture Implementation," ACM International Conference on Computing Frontiers, pp. 95-96, May 17-19, 2010. (**BDI Rated**)

The thesis culminates with a detailed description of the proposed architectural approaches for integrating test strategies into a high speed AES design: all techniques were/are about to be presented on international conferences:

- **F. Opritoiu**, M. Vladutiu, M. Udrescu*,* 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. (**ISI Rated**)
- **F. Opritoiu**, M. Vladutiu, L. Prodan, 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. (**ISI Rated**)
- **F. Opritoiu**, M. Vladutiu, M. Udrescu*,* L. Prodan, "Concurrent Error Detection for Multiplicative Inversion of Advanced Encryption Standard," *The 10th IEEE International Conference on Computer and Information Technology,* pp. 582-588, June 29-July 01, 2010. (**ISI Rated**)

The first solution constructs an on-line BIST based on parity prediction error detection mechanism: a new design is proposed justifying its efficiency through experimental results. The second testability measure refers to an off-line BIST strategy for which the error stimulation and detection is assured by LFSR-based structures: a LFSR for test vector generation and a MISR for response compaction and evaluation. The experimental results reveal a detection rate higher than 99.78% for multiple stuck-at faults. The third approach presents two architecture: the first being appropriate for intermittent faults of small multiplicity and for permanent faults, being also adaptable into an off-line BIST mechanism; and the second architecture being suited for detection of intermittent faults of higher multiplicity. The error detection architectures were analyzed with respect to the associated area overhead and error detection probability for stuck-at and stuck-open defects.

A brief summary of the contributions to the hardware design and test engineering presented in this PhD thesis are the following:

- A comprehensive review of the literature concerning the secure testing mechanism in general and their applicability to the encryption algorithms, and AES in particular. The solution found in the literature represented the starting point and the comparing measure for our proposed solutions. It also defined the research directions more clearly as well as emphasizing the importance offered by academic research to the secure testing solutions. Moreover it presents the authors perspective on the subject.
- A high speed iterative AES implementation, capable of performing one round of the algorithm in each clock cycle. The solution we constructed aims to reach a good trade-off between area overhead and operating speed. The principal contributions refers to the construction of the datapaths based on the reusability principle (applied to the most area-demanding operation of the design – the field multiplicative inversion)

resulting in a separation of the encryption and decryption sub-datapaths. The key generation unit was also designed specifically to reduce the difference in delays from encryption and decryption by duplicating the 32-bit field adders along its datapath.

- A parity-based on-line BIST design protecting the AES round implemented in hardware. The solution materializes as a distinct parity-prediction channel, following the principle of non-intrusive testing. As with the other non-intrusive solution, the net advantage relates to the independence of the test strategy on the particular round implementation details. The proposed solution computes the parity bytes associated with the state matrix, row-wise, allowing for efficient implementation of the parity predictors. Moreover, the mathematical relations for the prediction strategies for each transformation as well as for the key unit were derived.
- The applicability of the off-line BIST with respect to the three non-linear operations of the AES algorithm was investigated as well. As a consequence, the details for an off-line BIST designed protecting the field inversion module were offered with respect to the implementation of the stimulus generation and response validation units. The objective of the research was to find suitable LFSR-based configurations for the test generation and response compression. The search for appropriate LFSR structures was guided by simulations, allowing determining the LFSR-MISR pairs capable of detecting all single stuck-at defects. In addition the structures were simulated with respect to multiple faults yielding a detection rate higher than 99%. Comparative results were provided for the LFSR-based solution with respect to similar test stimuli generation: binary counter and deterministic generated test set.
- An on-line test mechanism adequate for the proposed AES architecture, and the round-level AES on-line protection was proposed. It verifies the shared multiplicative inversion units, by relying on a convenient mathematical property of the field multiplicative inversion. Dependent on the level of redundancy used in the verification scheme, the solution can be used for detecting permanent faults (both stuck-at and stuck-open) and intermittent faults of small multiplicity or multiple intermittent faults. The detection rate for the complex solution, targeting multiple faults is comparable to the one provided by the hardware duplication strategy while still saving silicon.
- A fault simulation framework, capable of generating all relevant faults for a design and simulating the architecture in a fault-free or defective context. The software approach assures high simulation speed, allowing to concurrently evaluating the output vectors for all input configurations. The framework is capable of simulating gate-level defects (stuck-at faults) as well as transistor-level faults (stuck-open, stuck-on)

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

- Flavius Opritoiu, PhD Report I, Politehnica University of Timisoara, July 2009
- Flavius Opritoiu, PhD Report II, Politehnica University of Timisoara, March 2010.

Concerning the further research in the domains of testability engineering and security algorithms, as well as a continuation of the proposed testing solutions and AES implementation architecture, we envision the following possible further refinements:

- Analyzing the impact of the test level (operation, round or algorithm) over the security of the AES on-line testing mechanisms already proposed
- Evaluating the effectiveness of Cellular Automata based test stimuli generators and response compactors with respect to the non-linear operations of the AES
- Constructing a signature-based concurrent checking mechanism, adaptable for both the non-linear as well as linear transformations of the Advanced Encryption Standard
- Building a dedicated off-line test strategy, to take advantage of the inherent pseudo-randomness associated with encryption algorithms in general and AES in particular.
- Extending the simulation framework with support for bridging and gate delay:
    - fault set generation for the fault models
    - fault simulation
- Factoring into the fault simulation framework the known fault distributions in order to increase the accuracy of the simulation.

Bibliography

[1] *Strategic Roadmap for Cryptography,* Technical Report, 2002.

[2] F. Rodriguez-Henriquez, N. A. Saqib, A. Diaz-Perez and C. K. Koc, *Cryptographic Algorithms on Reconfigurable Hardware*: Springer-Verlag, 2006.

[3] "ECRYPT NoE, Technical Report," *http://www.ecrypt.eu.org/*, 2002.

[4] B. Preneel, "ECRYPT II NoE," *ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/security/2008-02-29-ecryptii_en.pdf*, 2008.

[5] SIA, "The International Technology Roadmap for Semiconductors," *http://public.itrs.net*, 2004.

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fourth Edition: A Quantitative Approach*: Morgan Kaufmann Publishers, 2006.

[7] F. Opritoiu and M. Vladutiu, "Research Concerning Configuration of Cryptochips with Testability Facilities," *Diploma Project*, 2007.

[8] E. Larsson, *Introduction to Advanced System-on-Chip Test Design and Optimization*: Springer-Verlag, 2005.

[9] R. Goering, "Scan design called portal for hackers," 2004.

[10] *Maestra comprehensive guide to satellite TV testing,* Technical Report, 2002.

[11] 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.

[12] C. H. Kim and J.-J. Quisquater, "Faults, Injection Methods, and Fault Attacks," *IEEE Design & Test,* vol. 24, no. 6, pp. 544-545, 2007.

[13] S. P. Skorobogatov, *Semi-invasive attacks – A new approach to hardware security analysis,* Technical Report, University of Cambridge, 2005.

[14] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall and C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE,* vol. 94, no. 2, pp. 370-382, 2006.

[15] S. P. Skorobogatov and R. J. Anderson, "Optical Fault Induction Attacks," in Lecture Notes in Computer Science, Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems, 2003, pp. 2-12.

[16] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, no. 1, pp. 11-33, 2004.

[17] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems (3rd ed.): design and evaluation*: A. K. Peters, Ltd., 1998.

[18] M. Tehranipoor and N. Ahmed, *Nanometer Technology Designs: High-Quality Delay Tests*: Springer Publishing, 2007.

[19] A. Benso and P. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems*: Kluwer Academic Publishers, 2003.

[20] L.-T. Wang, C.-W. Wu and X. Wen, *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*: Morgan Kaufmann Publishers, 2006.

[21] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*: Springer-Verlag, 2000.

[22] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*: Wiley-IEEE Press, 1994.

[23] N. K. Jha and S. Gupta, *Testing of Digital Systems*: Cambridge University Press, 2002.

[24] M. Sachdev and J. P. d. Gyvez, *Defect-Oriented Testing for Nano-Metric CMOS VLSI Circuits (Frontiers in Electronic Testing)*: Springer-Verlag, 2007.

[25] K. Radecka and Z. Zilic, *Verification by Error Modeling: Using Testing Techniques in Hardware Verification*: Kluwer Academic Publishers, 2004.

[26] S. Ghosh and T. J. Chakraborty, "On behavior fault modeling for digital designs," *Journal of Electronic Testing,* vol. 2, no. 2, pp. 135-151, 1991.
[27] M. S. Reorda, Z. Peng and M. Violante, *System-level Test and Validation of Hardware/Software Systems*: Springer-Verlag, 2005.
[28] Z. Navabi, *VHDL: Analysis and Modeling of Digital Systems*: McGraw-Hill, 1997.
[29] P. J. Ashenden, *The Designer's Guide to VHDL, Volume 3, Third Edition*: Morgan Kaufmann Publishers, 2008.
[30] L.-T. Wang, Y.-W. Chang and K.-T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*: Morgan Kaufmann, 2009.
[31] R. D. Adams, *High Performance Memory Testing: Design Principles, Fault Modeling and Self-Test*: Springer, 2002.
[32] V. V. Belkin, "Testing diagnostics of modern microprocessors with the use of functional models," *Automation and Remote Control,* vol. 69, no. 8, pp. 1398-1410, 2008.
[33] T. Chao-Wen, "An Evaluation of Pseudo Random Testing for Detecting Real Defects," in IEEE VLSI Test Symposium, 2001, pp. 404-409.
[34] V. G. Oklobdzija, *Digital Design and Fabrication*: CRC Press, 2007.
[35] 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.
[36] C. Metra, M. Favalli and B. Ricco, "Embedded two-rail checkers with on-line testing ability," in Proceedings of the 14th IEEE VLSI Test Symposium, 1996, pp. 145.
[37] 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.
[38] J. Segura and A. Rubio, "A Detailed Analysis of CMOS SRAM's with Gate Oxide Short Defects," *IEEE Journal of Solid-State Circuits,* vol. 32, no. 10, pp. 1543-1550, 1997.
[39] R. Perry, "IDDQ testing in CMOS digital ASICs," *Journal of Electronic Testing: Theory and Applications,* vol. 3, no. 4, pp. 317-325, 1992.
[40] K. Baker, "QTAG: A Standard for Test Fixture Based IDDQ/ISSQ Monitors," in Proceedings of the IEEE International Test Conference, 1994, pp. 194-202.
[41] M. Sachdev, "Deep Sub-Micron IDDQ Testing: Issues and Solutions," in Proceedings of the 1997 European conference on Design and Test, 1997, pp. 271.
[42] J. M. Soden, C. F. Hawkins, R. K. Gulati and W. Mao, "IDDQ Testing: A Review," *Journal of Electronic Testing: Theory and Applications,* vol. 3, pp. 291-303, 1992.
[43] P. Lala, *An Introduction to Logic Circuit Testing*: Morgan and Claypool Publishers, 2008.
[44] A. Pancholy, J. Rajski and L. J. McNaughton, "Empirical Failure Analysis and Validation of Fault Models in CMOS VLSI Circuits," *IEEE Design & Test,* vol. 9, no. 1, pp. 72-83, 1992.
[45] H. K. Lee and D. S. Ha, *Atalanta: an Efficient ATPG for Combinational Circuits,* Technical Report, Virginia Polytechnic Institute and State University, Blacksburg, 1993.
[46] G. Zobrist, *VLSI Fault Modeling and Testing Techniques*: Ablex Publishing, 1993.
[47] J. Segura and C. F. Hawkins, *CMOS Electronics: How It Works, How It Fails*: John Wiley & Sons, 2004.
[48] S. Ramachandran, *Digital VLSI Systems Design: A Design Manual for Implementation of Projects on FPGAs and ASICs Using Verilog*: Springer-Verlag, 2007.
[49] J. E. Stine, J. Grad, I. Castellanos, J. Blank, V. Dave, M. Prakash, N. Iliev and N. Jachimiec, "A Framework for High-Level Synthesis of System-on-Chip Designs," in International Conference on Microelectronic Systems Education, 2005, pp. 11-12.
[50] C. Hu. "Berkeley MOSFET model," http://www-device.eecs.berkeley.edu/~bsim3/.
[51] D. Gizopoulos, *Advances in Electronic Testing: Challenges and Methodologies*: Springer-Verlag, 2006.
[52] R. C. Aitken, "Finding Defects with Fault Models," in Proceedings of the IEEE International Test Conference on Driving Down the Cost of Test, 1995, pp. 498-505.
[53] J. Emmert, C. Stroud and J. Bailey, "A new bridging fault model for more accurate fault behavior," *Proceedings of the Test Conference (AUTOTESTCON)*, pp. 481-485, 2000.

[54] P. Maxwell and R. Aitken, "Biased voting: a method for simulating CMOS bridging faults in the presence of variable gate logic thresholds," *Proceedings of the IEEE International Test Conference on Designing, Testing, and Diagnostics*, pp. 63-72, 1993.

[55] S. Ma, I. Shaik and R. S. Fetherston, "A Comparison of Bridging Fault Simulation Methods," in Proceedings of the 1999 IEEE International Test Conference, 1999, pp. 587.

[56] A. Saldanha, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Equivalence of robust delay-fault and single stuck-fault test generation," in Proceedings of the 29th ACM/IEEE Design Automation Conference, Anaheim, California, United States, 1992, pp. 173-176.

[57] H. Takahashi, K. K. Saluja and Y. Takamatsu, "An Alternative Method of Generating Tests for Path Delay Faults Using N -Detection Test Sets," in Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing, 2002, pp. 275.

[58] 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.

[59] W. J. Dally and J. W. Poulton, *Digital systems engineering*: Cambridge University Press, 1998.

[60] 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.

[61] 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.

[62] L.-T. Wang, C. Stroud and N. Touba, *System-on-Chip Test Architectures*: Morgan Kaufmann Publishing, 2007.

[63] SIA, "The International Technology Roadmap for Semiconductors - Update," *http://public.itrs.net*, 2005.

[64] SIA, "The International Technology Roadmap for Semiconductors - Update," *http://public.itrs.net*, 2006.

[65] E. A. Amerasekera and F. N. Najm, *Failure Mechanisms in Semiconductor Devices*: Wiley Publishing, 1997.

[66] 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.

[67] S. L. Hurst, *VLSI Testing: Digital and Mixed Analogue/Digital Techniques*: The Institution of Engineering and Technology, 1999.

[68] C. E. Stroud, *A Designer's Guide to Built-in Self-Test*: Springer-Verlag, 2002.

[69] K. P. Parker, *The Boundary-Scan Handbook*: Springer-Verlag, 2003.

[70] K. Stanley, "High-Accuracy Flush-and-Scan Software Diagnostic," *IEEE Design & Test,* vol. 18, no. 6, pp. 56-62, 2001.

[71] 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.

[72] 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.

[73] "Security Requirements for Cryptographic Modules," *Federal Information Processing Standards Publication*, May 25, 2001.

[74] J. E. Gentle, *Random Number Generation and Monte Carlo Methods, 2nd Ed*: Springer, 2003.

[75] T. Klove, *Codes for Error Detection*: World Scientific Publishing Company, 2007.

[76] S. Chang, "Reconfigurable Computing Approach for Tate Pairing Cryptosystems over Binary Fields," *IEEE Transactions on Computers,* vol. 58, pp. 1221-1237, 2009.

[77] K. K. Saluja and C.-F. See, "An Efficient Signature Computation Method," *IEEE Design & Test,* vol. 9, no. 4, pp. 22-26, 1992.

[78] R. Rajsuman, *System-on-a-Chip: Design and Test*: Artech House Publishers 2000.

[79] H. Bonnenberg, "Secure Testing of VLSI Cryptographic Equipment," PhD Thesis, 1993.

[80] M. Gössel, V. Ocheretny, E. Sogomonyan and D. Marienfeld, *New Methods of Concurrent Checking*: Springer-Verlag, 2008.

[81] S. L. Hurst, *VLSI Custom Microelectronics: Digital: Analog, and Mixed-Signal*: CRC Press, 1998.

[82] 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.

[83] 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.

[84] 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.

[85] D. K. Pradhan, *Fault-tolerant computer system design*: Prentice-Hall, 1996.

[86] E. J. McCluskey and C.-W. Tseng, "Stuck-fault tests vs. actual defects," in Proceedings IEEE International Test Conference, October, 2000, pp. 336-343.

[87] 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.

[88] 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.

[89] I. Koren and C. M. Krishna, *Fault Tolerant Systems*: Morgan Kaufmann Publishers, 2007.

[90] 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.

[91] 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.

[92] O. Kommerling and M. G. Kuhn, "Design principles for tamper-resistant smartcard processors," in Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology, Chicago, Illinois, 1999, pp. 2-2.

[93] M. Mogollon, *Cryptography and Security Services: Mechanisms and Applications*: CyberTech Publishing, 2008.

[94] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*: CRC, 1996.

[95] D. Stinson, *Cryptography: Theory and Practice, Third Edition*: Chapman & Hall/CRC, 2005.

[96] NIST, *Recommendation for Key Management - Part 1: General (Revised)*: NIST Special Publications, 2007.

[97] R. Anderson, M. Bond, J. Clulow and S. Skorobogatov, "Cryptographic Processors-A Survey," *Proceedings of the IEEE,* vol. 94, no. 2, pp. 357-369, 2006.

[98] J. Daemen and V. Rijmen, *The Design of Rijndael*: Springer-Verlag, 2002.

[99] M. Feldhofer, J. Wolkerstorfer and V. Rijmen, "AES implementation on a grain of sand," *IEEE Proceedings on Information Security,* vol. 152, no. 1, pp. 13-20, 2005.

[100] G. D. Natale, M.-L. Flottes and B. Rouzeyre, "On-Line Self-Test of AES Hardware Implementations," in The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007.

[101] N. M. Kosaraju, M. Varanasi and S. P. Mohanty, "A high-performance VLSI architecture for advanced encryption standard (AES) algorithm," *The 19th International Conference on VLSI Design*, pp. 4 pp., 2006.

[102] Z. Xinmiao and K. K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm," *IEEE Transactions on Circuits and Systems Part II: Express Briefs,* vol. 53, no. 10, pp. 1153-1157, 2006.

[103] A. Satoh, S. Morioka, K. Takano and S. Munetoh, "A Compact Rijndael Hardware Architecture with S-Box Optimization," in Proceedings of the 7th International

Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, 2001.

[104] A. Hodjat and I. Verbauwhede, "Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors," *IEEE Transactions on Computers,* vol. 55, no. 4, pp. 366-372, 2006.

[105] M. F. Elisabeth Oswald, Kerstin Lemke, Francois-Xavier Standaert, Thomas Wollinger, Johannes Wolkerstorfer, "State of the Art in Hardware Architectures," *European Network of Excellence in Cryptology*, 2005.

[106] F. Opritoiu, M. Vladutiu, L. Prodan and M. Udrescu, "A High-Speed AES Architecture Implementation," in ACM International Conference on Computing Frontiers, May 17-19, 2010, pp. 95-96.

[107] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard," *IEEE Transactions on Computers,* vol. 59, no. 5, pp. 608-622, 2010.

[108] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "A Structure-independent Approach for Fault Detection Hardware Implementations of the Advanced Encryption Standard," in Fault Diagnosis and Tolerance in Cryptography, 2007. FDTC 2007. Workshop on, 2007, pp. 47-53.

[109] Y. Chih-Hsu and W. Bing-Fei, "Simple error detection methods for hardware implementation of Advanced Encryption Standard," *IEEE Transactions on Computers,* vol. 55, no. 6, pp. 720-731, 2006.

[110] M. Karpovsky, K. J. Kulikowski and A. Taubin, "Differential Fault Analysis Attack Resistant Architectures for the Advanced Encryption Standard," *Smart Card Technologies and Applications VI*, pp. 177-192: Springer-Verlag, 2004.

[111] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Parity-Based Fault Detection Architecture of S-box for Advanced Encryption Standard," in 21st IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2006, pp. 572-580.

[112] L. Breveglieri, I. Koren and P. Maistri, "Incorporating Error Detection and Online Reconfiguration into a Regular Architecture for the Advanced Encryption Standard," in Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2005.

[113] G. Bertoni, L. Breveglieri, I. Koren and P. Maistri, "An efficient hardware-based fault diagnosis scheme for AES: performances and cost," in Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2004, pp. 130-138.

[114] S. Mangard, M. Aigner and S. Dominikus, "A highly regular and scalable AES hardware architecture," *IEEE Transactions on Computers,* vol. 52, no. 4, pp. 483-491, 2003.

[115] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri and V. Piuri, "Detecting and locating faults in VLSI implementations of the Advanced Encryption Standard," in Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2003, pp. 105-113.

[116] G. D. Natale, M. L. Flottes and B. Rouzeyre, "A Novel Parity Bit Scheme for SBox in AES Circuits," in Design and Diagnostics of Electronic Circuits and Systems, 2007. DDECS '07. IEEE, 2007, pp. 1-5.

[117] K. Wu, K. Ramesh, G. Kuznetsov and M. Goessel, "Low cost concurrent error detection for the advanced encryption standard," in Proceedings of the International Test Conference, 2004, pp. 1242-1248.

[118] R. Karri, K. Wu, P. Mishra and K. Yongkook, "Fault-based side-channel cryptanalysis tolerant Rijndael symmetric block cipher architecture," in Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001, pp. 427-435.

[119] N. Yu and H. M. Heys, "A Compact ASIC Implementation of the Advanced Encryption Standard with Concurrent Error Detection," in Proceedings of Circuits, Signals, and Systems, July, 2007.

[120] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Application*: Wiley-Interscience, 2006.

[121] P. K. Lala, *Self-checking and fault-tolerant digital design*: Morgan Kaufmann Publishers, 2001.

[122] F. Opritoiu, M. Vladutiu, M. Udrescu and L. Prodan, “Concurrent Error Detection for Multiplicative Inversion of Advanced Encryption Standard,” in The 10th IEEE International Conference on Computer and Information Technology, June 29-July 01, 2010, pp. 582-588.

[123] J. Gallian, *Contemporary Abstract Algebra, 7th Ed*: Cengage Learning, 2009.