VHDL Modelling and Analysis of Fault Secure Systems

Jason Coppens, Dhamin Al-Khalili, and Côme Rozon
Department of Electrical and Computer Engineering
Royal Military College of Canada
Kingston, Ontario K7K 7B4

Abstract

This paper presents an analysis process targeted for the verification of fault secure systems during their design phase. This process deals with a realistic set of micro-defects at the device level which are mapped into mutant and saboteur based VHDL fault models in the form of logical and/or performance degradation faults. Automatic defect injection and simulation are performed through a VHDL test bench. Extensive post processing analysis is performed to determine defect coverage, figure of merit for fault secureness, and MTTF.

1. Introduction

For systems where security and reliability are critical, the design must be tolerant to defects. These tolerant designs must identify conditions where the correct operation of the system is compromised, then react to maintain system integrity. Testing for functionality or performing conventional fault simulation are insufficient to ensure correct operation of these systems. Any verification of critical designs must be comprehensive and satisfy certain requirements. A measure of the level of confidence that one can place in a system's ability to remain fault secure becomes the deciding factor in the selection of the desired design. Therefore, the analysis methodology and the circuit modeling techniques, in conjunction with realistic operational scenarios are the main ingredients for reliable results.

There have been studies aimed at developing statistical metrics which can be applied to critical logic blocks for the determination of their fault security [1,2]. These metrics can be extracted early in the design cycle. However, the determination of the parameters used in the metric calculation is often unclear and too general, leaving in doubt the results of these calculations.

The use of fault injection has been advocated by studies to carry out verifications for fault tolerant systems [2,3,4]. Fault injection involves the deliberate introduction of faulty behavior into a circuit, which is then monitored for a response through simulation. However, most of the proposed techniques do not address the problem at the physical defect level and the implication on circuit behavior.

In contrast to the above previous approaches, the proposed process carries out automatic defect injection into the circuit allowing for technology specific analysis of system behavior in a multi-fault environment. This environment include both logical faults and performance degradation. The proposed methodology meets the goal of developing a realistic technique for the verification of the critical blocks of fault secure ASIC designs.

2. Faulty behavior and security analysis

The progression of circuit behavior from defect to failure can be illustrated by the flow of states shown in Figure 1. If the detection scheme detects the error on the output and flags it as unreliable, then the system is said to have failed secure. Otherwise, if the system's checking scheme does not flag the erroneous output as being unreliable, then the system has failed insecure. Unexcited defects (either due to defect size or input pattern excitation) are considered dormant, and unexcited faults remain latent. The detection abilities of the checking hardware determines if the transition will be to a secure or insecure failure state

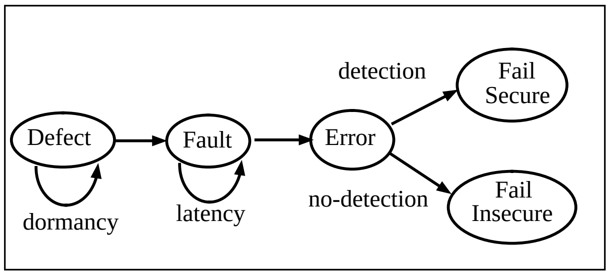

Figure 1. Defect path to failure

The analysis process proposed here has two thrusts: assurance of the fault security defects, and highlight of areas for design improvement. The requirement for an evaluation scheme is to obtain a level of confidence in the system's ability to maintain fault security in the presence of defects. The level of confidence obtained by this verification is best represented by the probability of the design not failing given by equation (1). This probability represents the figure of merit of the system.

$$\begin{aligned} P\{\text{Secure Operation}\} &= 1 - P\{\text{Failure}\} \\ &= 1 - P\{\text{Failure} \mid \text{defect}\} \bullet P\{\text{defect}\} \\ &= 1 - [P\{\text{Failure} \mid \text{Error}\} \bullet P\{\text{Error} \mid \text{Fault}\} \bullet \\ &\quad P\{\text{Fault} \mid \text{defect}\} \bullet P\{\text{defect}\}] \end{aligned} \quad (1)$$

Figure 2 contains the diagram of a probability state model for life cycle analysis in the proposed process where the state transition probabilities are assumed to be Markovian [1]. There are two entries to this model. If a circuit is certified as defect free then it starts in the "DF" state. If however, there exists the possibility of a dormant defect in the circuit, then the circuit starts in the dormant or latent defect "L" state. States "FS" and "FUS" represent the failed secure and failed insecure states respectively. For this Markov model, λ is the defect arrival rate, and *t* represents time.

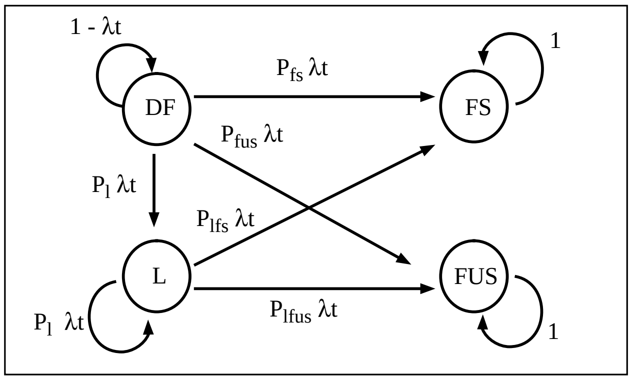

Figure 2. Probability model of a fault secure system

3. Micro-defect modeling

Our work lead to the generation of a MOS transistor model containing nine generic defects. Parasitic circuit elements were used to create a representation of each of the nine defects. Most shorts and opens could be modeled using straight forward resistive and capacitive elements. Modeling of the gate oxide pinholes behavior required special considerations [5].

Two defect classes were studied: (1) "Hard" defects which represent severe physical conditions. Defects of this category manifested faults usually logical in nature, such as stuck-at conditions or changes in the logic function performed by the gate; (2) "Soft" defects which represent abnormalities in the making. Typically, gates with a soft defect exhibits a performance degradation, such as reduced noise margin, change in output voltage swing, or increased propagation delay. The critical circuit parameters (resistances for shorts and opens), defining the boundary between these two classes of defects were determined through extensive simulations [6,8].

Having established a transistor level defect model suitable for CMOS technologies, the response of various gate structures influenced by each of the nine defects was determined. Results were obtained for both "Hard" and "Soft" ranges of defect parameters. From these responses, defect-to-fault mappings was determined for each gate structure.

Nortel's 0.8μm CMOS standard cell library was used to provide a foundation for the defect-to-fault mappings. Analog simulations, including an exhaustive input pattern and defect injection conducted with common gates, yielded the set of generic fault categories appearing in Table 1. The last two columns of the table list the set of logical and performance degradation faults obtained for a typical D-latch gate injected with 108 hard defects and 108 soft defects.

Table 1. D-latch defect-to-fault response

Fault Category	Numbers in Category	
	Hard Defects	Soft Defects
Stuck At	82	19
Iddq Increases	68	60
Delay	0	66
Noise Margin Reduction	3	42
Unobservable Fault	16	27

Note that a single defect may result in multiple faults as demonstrated in the bar chart of Figure 3. This type of data provides the basis for the defect-to-fault mapping which will be part of the VHDL models of defective gates.

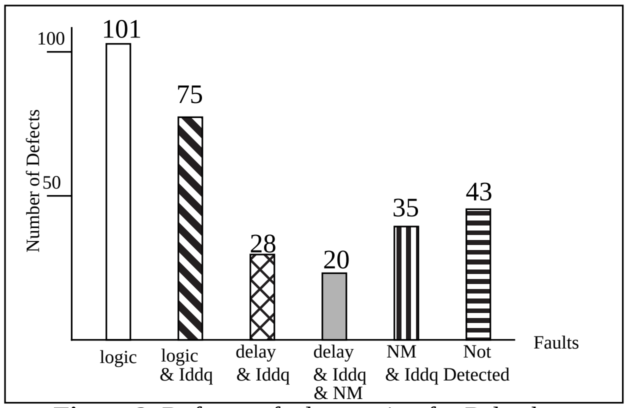

Figure 3. Defect-to-fault mapping for D-latch

4. VHDL fault modeling

Faults are injected into VHDL models of the design and excited by a set of input patterns. The basic methods appear in Figure 4. The first involves the addition of a *saboteur.* This entity is placed external to the existing logic gate and can be used without altering the existing gate models. Saboteurs can be used to model most faults and to simulate environmental conditions such as noise or ESD. However, because they have no input pattern discrimination, saboteurs cannot model faults below the gate level of abstraction. The second method is referred to as *mutant* injection. A mutant is a model which contains dormant code blocks within the normal gate description. These blocks of code are activated by injecting faults, altering the operation of the logic device itself. Because the fault response is generated internally within the model, any level of abstraction for fault injection is possible. However, the use of mutants requires that the original gate models be replaced by the new mutant models.

In our analysis process, the defect injection into the gates is carried out using controllable mutants for defect-to-fault mappings. Saboteurs are used to model interconnect bridging between signal lines external to the logic gates.

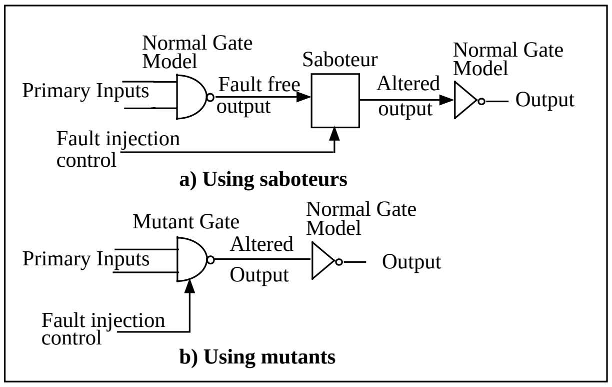

Figure 4. Fault injection methods

5. The analysis process

Our security fault analysis process consists of four major phases: setup, modelling, simulation and analysis. A complete flow diagram is shown in Figure 5. The **setup** phase establishes the modeling environment (test vectors, defect analysis) according to the target technology and generates a structural VHDL description of the design or system blocks of interest. The **modeling** phase prepares the injectable models for VHDL simulation by placing mutants and saboteurs constructs. Previously established VHDL models can also be imported from a fault security analysis (FSA) gate library by default, if the case applies, removing the need to replace the gate models. In the **simulation phase**, the defect injectable model is placed within a simulation test bench. Two output files are generated by the simulation. One file contains the results of I_{DDQ} monitoring, while the other contains response mismatches from output monitoring data. Finally, in the **analysis** phase the data is compiled to obtain defect coverage and failure statistics. Using the statistical calculations, figures describing the probability of fail safe operation are generated.

6. Benchmark application

As a proof of concept of the proposed process, analysis was conducted on two versions of a benchmark. It is an eight bit ALU, synthesized from a behavioral VHDL

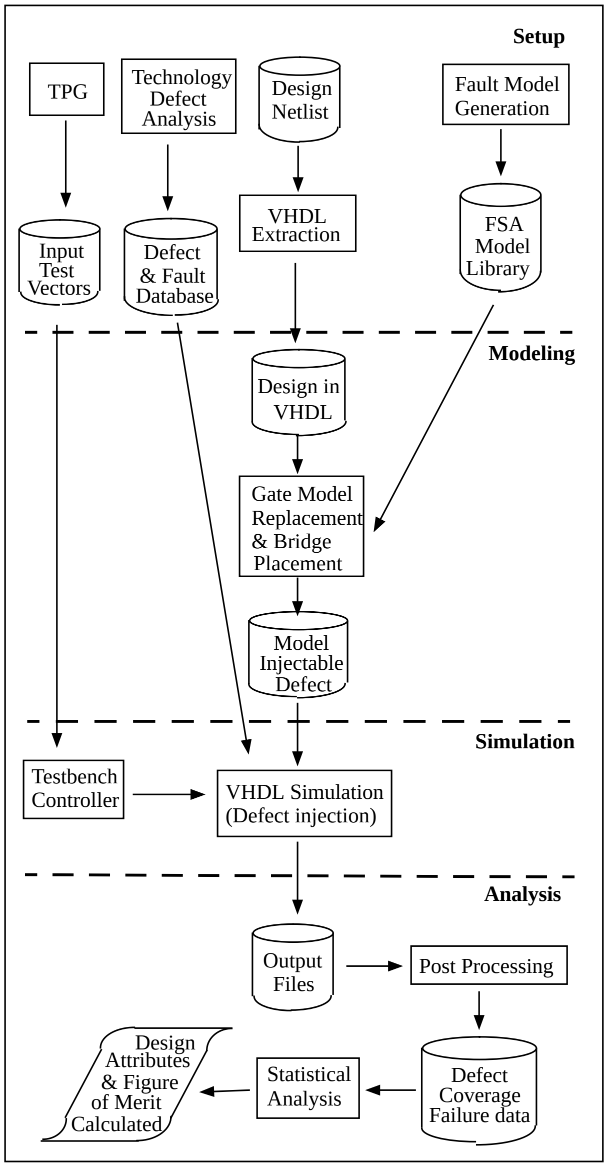

Figure 5. Process flow diagram.

description to a structural architecture, with two types of error checking schemes: parity and complementary. Both checking systems and the ALU block were designed using

Nortel's 0.8µ BiCMOS process. The block diagram of the benchmark circuit is presented in Figure 6.

An exhaustive defect injection was employed. The circuit was stimulated with a set of 250 pseudo-random input vectors to provide high fault excitation. Two simulation scenarios were conducted. The first scenario injects single defects throughout the entire benchmark. Data obtained from this scenario can be used to generate defect coverage statistics of the test pattern set, and a figure of merit for a single defect environment. The second simulation scenario targets the checking hardware block to look for error masking by defects injected within the checker. Such a situation would be analogous to a multiple defect environment.

7. Results and performance analysis

7.1 Defect coverage and failure statistics

The chart of Figure 7 illustrates the single defect injection results. These statistics represent the defect coverage of the benchmark blocks for the input vectors. Percentages are given for those defects which caused a logical error on the circuits outputs (logic errors) and those which caused an unknown or intermediate value on the circuit's

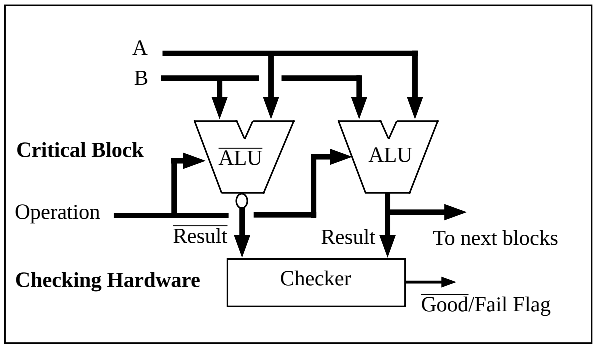

Figure 6. Benchmark block diagram

output (possible errors). Failure statistics for the two checking schemes are presented by the chart of Figure 8. The shaded portions represent an intermediate value on the circuit's output which may or may not trigger the $\overline{\text{Good}}$/ Fail flag.

The graph of Figure 9 illustrates the percentage of defects generating an increase in I_{DDQ} draw, representing the potential defect coverage of an I_{DDQ} test for each block. The defect coverage of an I_{DDQ} test is insufficient to ensure fault secure operations. However, it can be used in combination with another fault detection method. Error masking statistics, which represent failures, are provided in Figure 10. White portion represents a condition of a logic error being masked by the defective checker. The shaded region represents logical errors from the critical

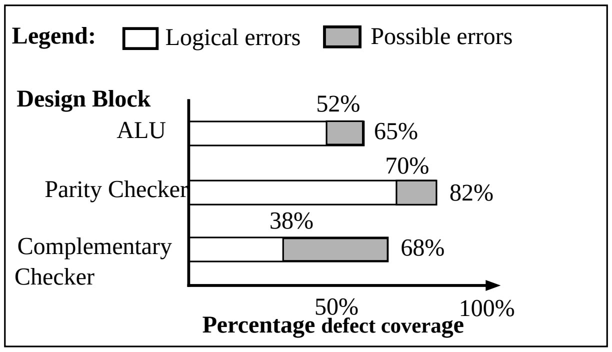

Figure 7. Single defect coverage statistics

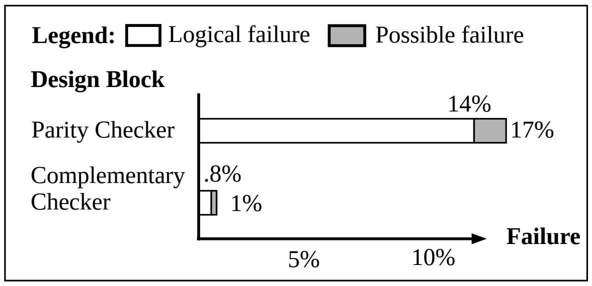

Figure 8. Single defect failure statistics

.

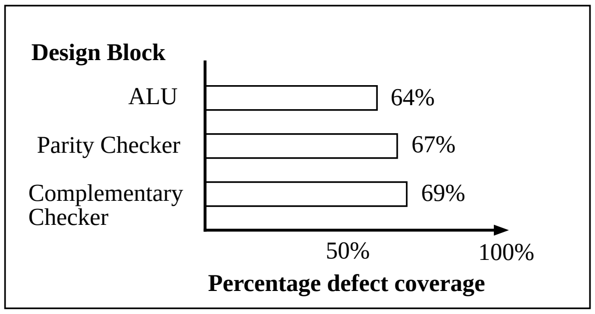

Figure 9. I_{DDQ} Test defect coverage

block (the ALU) which could possibly be masked by the defective checking hardware

7.2 Figures of merit

From the results obtained in the first scenario, and using equation (1), it is possible to determine the figures of merit for both the parity checker and complementary checker of the benchmark. Table 2 presents the results for single defect, combined IDDQ monitoring and multiple defects. The addition of current monitoring improves the figure of merit. For a multiple defect environment, there was accounting for defects in the checking hardware masking a "fail" flag.Therefore the effect of error masking is to increase the probability of a system failing to an insecure state. The decrease is more pronounced for the complementary checker because of its larger size compared to the parity checker.

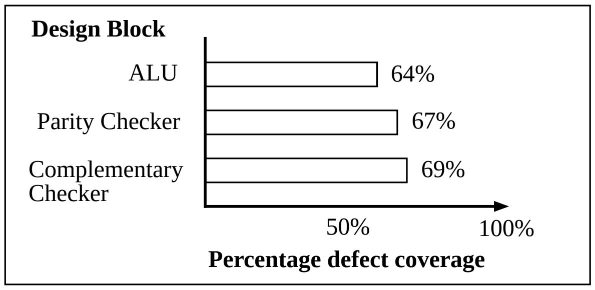

Figure 9. I_{DDQ} Test defect coverage

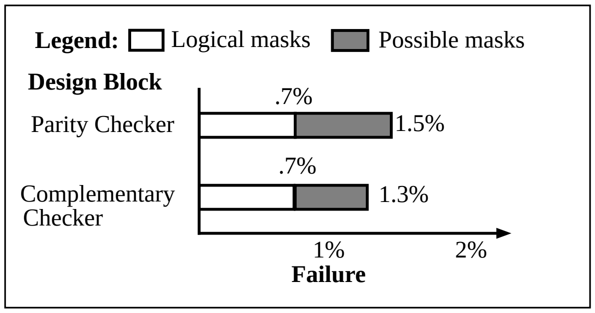

Figure 10. Failures due to masking errors

Table 2. Figures of merit

Checker circuit	Single defect		Multiple defects
	Without IDDQ	With IDDQ	
Parity	83%	83.2%	82.9%
Complementary	99%	99.6%	98.5%

7.3 Life cycle analysis

To perform life cycle analysis, the Markov model presented earlier is used. The various transition probabilities between states can be calculated by extracting the necessary parameters from the simulation data. Once the model is established, a plot can be used to show the dependency between a desired probability of secure operation and the mean-time-to-failure (MTTF) of the system. Such a plot appears in Figure 11 for the parity checker for defect arrival rates λ to represent best case (10-7defects/hr), worst case (10-3 defects/hr), and average value (10-5defects/hr) [1,7]. For the latter value of λ and a desired 95% fault security, the MTTF for the parity checker is found to be 5 years, compared to 50 years for the complementary checker. This shows the importance of the selection of a strong coding scheme to obtain high fault security for a system.

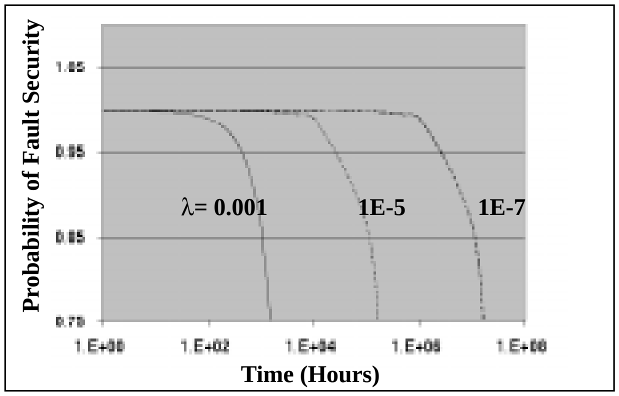

Figure 11. Time analysis curves for the parity checker

8. Conclusion

This paper has presented a technique suitable for conducting fault security analysis with emphasis on integrated circuit technologies. It is believed that this analysis process fulfills the requirements necessary for the verification of highly fault secure or high reliable digital designs. The use of the VHDL simulation environment to achieve statistical results has introduced automation into the analysis.

References

[1] "*Fail-Safe Design and Evaluation Techniques*", US Department of Defence Contract #MDA904-93-C-4027 Final Report, Research Triangle Institute, Sept. 1994.

[2] J. Arlat, M. Auguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "*Fault injection for Dependability Validation: a Methodology and Some Applications*", IEEE Transaction on Software Engineering, Feb. 1990, pp 166-182.

[3] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool", Proc. 24th International Symposium Fault-Tolerant Computing, IEEE, 1994, pp 66-75.

[4] T. Delong, B. Hohnson, and J. Profeta III, "A Fault Injection for VHDL Behavioral-Level Models", IEEE Design and Test of Computers, Winter 1996, pp 24-33

[5] J. Soden, C. Hawkins, and A. Miller, "Identifying Defects in Deep-Submicron CMOS ICs", IEEE Spectrum, Sept 1996, pp 66-71.

[6] C. Hawkins, "IC Trends, Quality, Defects, Testing and Reliability", Internal course, Nortel, 1996.

[7] T. Nanya and T Kawamura, "A Note on Strongly Fault-Secure Sequential Circuits," IEEE Trans. on Computers, Vol C-36, No. 9, Sept 1987, pp1121-1123.

[8] J.M. Coppens, "Logical Fault Analysis of Fault Secure Systems", MSc. Thesis, Royal Military College, May 1997.