A Fault Model for VHDL Descriptions at the Register Transfer Level[*]

T. Riesgo, J. Uceda
Universidad Politécnica de Madrid
División de Ingeniería Electrónica. E.T.S.I.I
c/José Gutiérrez Abascal, 2. 28006 Madrid (SPAIN)
Tel. 34-1-411 75 17. Fax. 34-1-564 59 66
e-mail: teresa@upmdie.upm.es

Abstract

This paper presents a fault model for VHDL descriptions at the Register Transfer Level and its evaluation with respect to a logic level fault model (single-stuck-at). The proposed fault model may be used for early estimations of the fault coverage before the synthesis is made in the design process of an integrated circuit. The obtained results show a high correlation between the fault coverages achieved with the proposed fault model and logic fault models on a set of examples. The main contribution of this work is the proposal of a new fault model for VHDL/RT descriptions and the demonstration of its usefulness for estimating the achieved fault coverage with a set of test vectors in design phases previous to synthesis.

1. Introduction

Testing is one of the key tasks within the design process of electronic systems and its definition is a specially hard task in integrated circuit design. Test pattern generation is a procedure that defines the input stimuli (*test vectors*) that have to be applied to a manufactured device to assure that it does not have any defect. As the number of possible defects in a complex integrated device is so large, it is almost impossible to assure the total integrity of the circuit with a rational number of test vectors. Therefore, an abstraction of these physical defects has to be made to make the problem affordable for the designer that has to generate them during the design phase. *Fault models* are abstract representations of the circuit behaviour under the effect of a physical failure. Now, the goal of the test vectors defined by the designer is the detection of faults defined by this fault model that are much closer to the design aspects than the physical defects, more closely related to technological aspects [1]. The definition and adoption of a fault model implies a compromise between two main aspects: accuracy of the model, as if it does not represent the circuit defects the test quality will be very bad, and its simplicity, as if it is very complex nor the designer neither the CAD tools will be able to deal with it.

Most of current testing procedures are actually using the *single-stuck-at-'0', '1'* fault model defined at the logic gate level. The normal requirement for industrial designs is that the set of test vectors provided by the designer achieve at least a 95% fault coverage of all detectable stuck-at faults (what means that the test vectors have to detect at the primary outputs of the circuit, the 95% of all the detectable stuck-at faults in the circuit). A lot of work has been and is being made to change this fault model, as it has been shown that it does not correctly model the possible defects inside the integrated circuits. So, new fault models have been proposed based on switch level descriptions [2] (stuck-on, stuck-off), on delay faults and more recently on electrical current testing (what are usually called Iddq techniques [3]).

Another important factor to be considered in the fault model definition, is that the fault model has to be tightly tied to the design process to assure the efficiency in generating the test vectors. Current design processes are based on top-down methodologies, using hardware description languages (HDLs) as input to the design process, automatic synthesis tools, and therefore, the designer is more dedicated to high-level tasks (at the algorithmic or architectural level) more than low-level design (like logic gate or transistor levels) [4]. As well as the design process itself has changed a lot, the test generation and evaluation procedures must also change.

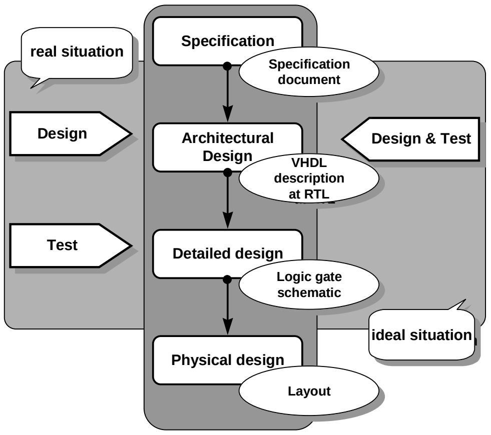


Figure 1. Test in the design process based on automatic synthesis

The left part of Figure 1 shows the disconnection between design and test tasks in the design flow based on a top-down methodology. The right part of this figure shows

[*] This work has been partially funded by ESIP (ESPRIT #8370)

the goal of coupling both design and test in the same phase of the design process.

Two main solutions could be possible to access the test definition from the early stages of the design process:

a) The use of test synthesis tools. This method consists on the insertion of structured test blocks in the design (like Scan Path) and the generation of the test vectors. The designer may use this tool at the same time the circuit is being synthesised with the advantage that the method is almost automatic and all the test logic and vectors are available in the detailed design step. The main disadvantage is that the cost in area of the added logic and pins may not be affordable in some cases. Besides, the circuits have to be totally synchronous to be handled by these tools [5].

The early estimations of fault coverage from the HDL descriptions. This method consists on having a measurement of the testability of the circuit and the quality of a set of test vectors before the synthesis is being made. This way the designer may be aware of the amount of additional vectors required for the test and where are the low detection areas of the circuit. This method does not provide the test vectors but allows the designer to have an early information about the testability of the circuit and the possible requirements of test logic within it.

The work presented in this paper is included in this last approach. The procedure will be the definition of a fault model for register transfer level (RTL) descriptions in VHDL, as they are the descriptions where the designer uses to work in current design technologies.

There is no accepted fault model for RTL, not only in VHDL, but also in any other design environment. Some other fault models have been developed for VHDL or HDL based descriptions. For example, in [6] a fault model and a complete test set tool, including automatic test pattern generation and fault simulation, is presented. The main problem of the solution proposed is the fault model itself, that has not been compared with low level fault models and therefore the accuracy of the test method is not shown. The type of descriptions supported by the method defined in [6] are single-process VHDL descriptions. The fault model presented in [7] for C-hardware descriptions follows the same goal as the one presented in this paper: having a good estimation of the lower level faults. However, it considers some faults defined as “multiple” (not single faults) that are not closely related to the low level faults. Besides, a fault model based on VHDL will be more useful as this language is the design vehicle of most current methodologies.

The main difference between the fault model described in this paper and the already proposed in [6] and [7] is that this one is totally oriented to model hardware faults and its definition has been directly driven to this goal. For this reason, the fault model will deal with synthesisable descriptions, whose translation to logic gates is well known, and those possible faults at RT level with no clear translation to lower levels have not been considered. For estimating its accuracy, several experiments have been run on a set of examples to evaluate how good is the estimation of fault coverage with respect to the normally used fault models (stuck-at-‘0’,’1’).

The paper is organised as follows. Section 1 describes the fault model in detail. Section 3 shows the experimental results obtained in the evaluation of the fault model on a set of examples. Section 4 presents the conclusions of the paper.

2. Fault model description

The use of a fault model at RT level would allow the designer has a previous estimation of the fault coverage achieved with a test vector set, before the synthesis is made. At this point, architectural modifications to improve the circuit testability could be performed and the test generation time could be reduced at later stages of the design flow. Therefore, this fault model is oriented to make easier estimations of the fault coverage achieved in pre-synthesised descriptions in VHDL and its final goal is directly related to manufacturing test more than other possible verification tasks such as design error detection, functional verification, etc. Those faults with no hardware meaning have not been considered in the fault model.

Once the objectives of this fault model have been identified, the input descriptions should also be identified in terms of those VHDL constructs that the fault model will consider. As the main objective is having a definition for pre-synthesis descriptions, a synthesis VHDL subset has been chosen: Level-0 synthesis subset [8]. This subset includes those VHDL constructs that are accepted by the most important commercial synthesis tools and defines not only the VHDL subset but also the synthesis semantics (associated hardware) to the mentioned descriptions.

2.1 Assumptions of the fault model definition

Some assumptions previous to the fault model definition have been established. The main preconditions are:

* The fault model is based on **single** faults. This means that only one fault is inserted in the VHDL description at a time.
* Intermittent faults are not considered. The fault effect will be present during the whole simulation, as they are **permanent** faults.
* Faults will have **local scope**. Each fault will be injected in a particular line of code where the faulty object is referenced.
* Input descriptions (fault-free) will be compliant to the Level-0 subset. The faulty descriptions may not be compliant to the subset, although they must be VHDL correct, so that they can be simulated in a standard VHDL simulator.

* The internal code (body) of the functions or procedures called in the description will be affected by the same fault model as the VHDL architectures.
* The fault model is defined for **behavioural** descriptions. Structural descriptions will be affected by structural faults based on the “stuck-at” fault model on the interconnections among behavioural modules, as shown in Figure 2. Level-0 subset does not allow mixed descriptions (structural-behavioural).

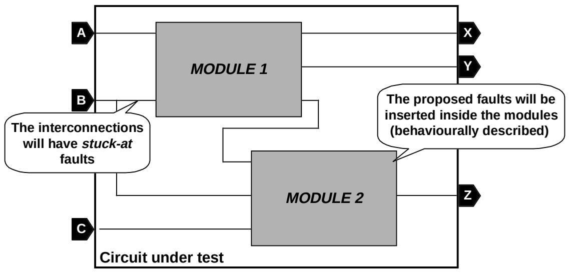

A
B
C
X
Y
Z
MODULE 1
MODULE 2
The interconnections will have *stuck-at* faults
The proposed faults will be inserted inside the modules (behaviourally described)
Circuit under test

Figure 2. Fault insertion for RT structural descriptions

2.2 Detailed description of the fault model

The fault model is divided in three classes, depending on the type of object affected by the fault:

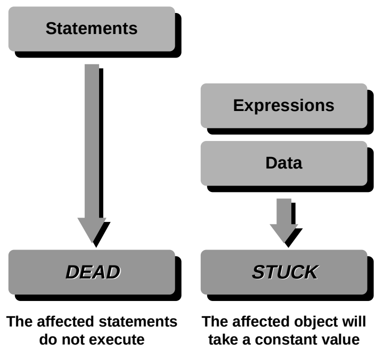

Statements
Expressions
Data
DEAD
STUCK
The affected statements do not execute
The affected object will take a constant value

Figure 3. Summary of RT fault model

* *Faults on data*: the fault model is based on “stuck-at” faults. The affected object (signal, variable) will take a constant value and the insertion will be made in a statement where the object is referenced. The insertion is made on assignment statements, affecting the right part of the assignment (target object).
* *Faults on expressions*: the fault model is based on “stuck-at” faults. The affected expression will take a constant value. Two types of expressions are identified:
  › *Control expressions*: those expressions that appear in the control statements.
  › *Data expressions*: those expressions that appear in the left part of an assignment statement.
* *Faults on statements*: the fault model is based on “dead” faults. The effect of the fault is that the affected statements are not executed.

Faults on data: For each data type, the object under fault may be stuck-at different values

* <u>Scalar data types</u>
  › *bit, std_logic*: The considered fault models are *stuck-at-‘0’, stuck-at-‘1’*. In *std_logic* data types no other possible fault model is considered (*stuck-at-‘Z’*, etc.) because they do not have a particular hardware significance.
  › *enumerated*: The signals or variables of any user defined enumerated type can be *stuck-at-“all possible values”*.
  › *integer, natural, positive*: As the codification at the logic level by the synthesis tools is known, the fault model defined for these types makes that each bit of the resulting bus can be stuck-at-‘0’ or stuck-at-‘1’. The assumed codification is binary for the positive numbers and 2’s complement for the types that have negative values in their range. This solution makes the fault model closer to the hardware structure and reduces the number of possible faults.
* <u>Composite data types</u>
  › *arrays*: Each element of the array will have an individual fault. This makes the fault model closer to the low level fault models with the assumption of single faults. Level-0 subset does not accept record data types, so arrays are the only possible composite data types.

Faults on expressions:

* <u>Control expressions</u>
  › *if__then__else__:* The condition of the if statement may be *stuck-at-true* or *stuck-at-false*
  › *case__is__when__:* The expression which controls the case statement may be *stuck-at-“all possible values”*.
  › *for__in__loop__:* The index controlling the loop may change its range from the *minimum* to the *maximum+1* and from the *minimum-1* to the *maximum*.
  › *concurrent statements*: Concurrent statements will be translated to their equivalent in processes and sequential statements and the same fault model will be applied.
* <u>Data expressions</u>
  › The same fault model used for data faults will be used on expressions, depending on the type of value returned by the expression.

Faults on statements:

* <u>Sequential statements</u>
  › *if__then__else__:* The if statement may fail with two faults*: dead-then, dead-else*. In the first case the statements associated with the *then* part will not execute and in the second case the statements associated with the *else* part will not execute.
  › *case__is__when__:* The fault model applied is *dead-alternative*, which makes that the affected alternative (*when* statement part) does not execute.

› *for__in__loop__:* The fault model is *dead-loop*, which makes the body of the loop does not execute.
› *procedure call:* The fault model causes that the procedure call does not take place (*dead-call*).
› *signal or variable assignment:* The signal or variable assignment statement does not execute (*dead-assignment*).

* Concurrent statements: Concurrent statements will be translated to their equivalent in processes and sequential statements and the same fault model will be applied (*dead-then, dead-else, dead-assignment, dead-call*).

2.3 Insertion mechanisms

For this fault model, the fault list will have a particular structure. Each fault will be identified by a label, that will be the name of the corresponding faulty architecture, a number referred to the line of the VHDL where the fault will be inserted, and a set of particular features that depend on the fault class (value to *stuck-at* to, name of the affected signal or variable, etc.).

The fault insertion is made on the VHDL code (fault-free description), by adding code perturbations to this input description. This way, several architectures of the same entity are generated, each one representing one fault. The code perturbations consist of **adding**, **switching** or **eliminating** code sentences, to model the circuit behaviour under a fault condition.

* Data faults: An additional line of code is added after the affected line, which assigns the faulty value to the data object under fault.
* Data expression faults: The fault-free expression is switched to the faulty one.
* Control expression faults: The faulty expression is switched to a variable that takes the faulty value and is locally defined for the process where the faulty line is located.
* Statement faults: The lines of code affected by the fault are eliminated by adding comments (--) to them.

3. Experimental results

This section presents the experimental results obtained in the comparison of the proposed fault model with the stuck-at fault model at the logic gate level.

The comparison has been made using a set of examples and following the method proposed in Figure 4. On one hand, the input description is synthesised, with a commercial automatic synthesis tool, and the fault simulation is performed on the equivalent netlist with a set of test vectors. The same set of vectors is used to perform a VHDL fault simulation using the proposed fault model. The resulting fault coverages (logic and VHDL/RT) are then compared.

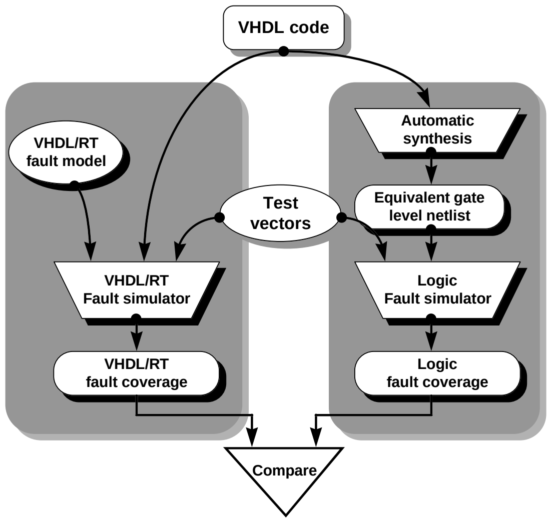


Figure 4. Proposed method for fault model evaluation

The following table shows a brief description of the example descriptions used for these experiments. For the first two examples (*ALU*, *ALU16*) three different synthesis processes have been run, optimising time or area. This is shown with different number of logic faults for each experiment.

Circuit	*nr.VHDL faults*	*nr.logic faults*	*Description*
ALU	302	1481	8-bit ALU with 8 operations
		2160	
		2530	
ALU16	493	6662	16-bit ALU with 8 operations
		7164	
		7174	
MULTIS	503	3244	sequential multiplier (8 bits)
SHREG	66	480	shift register (8 bits)
SUBST	586	2621	contains ALU and a FSM
TCONT16	208	1146	16-bit counter with test logic
TRAREC	223	1468	reception-transmission unit

Table 1. Description of the example circuits

The VHDL fault simulation has been made with the following steps:

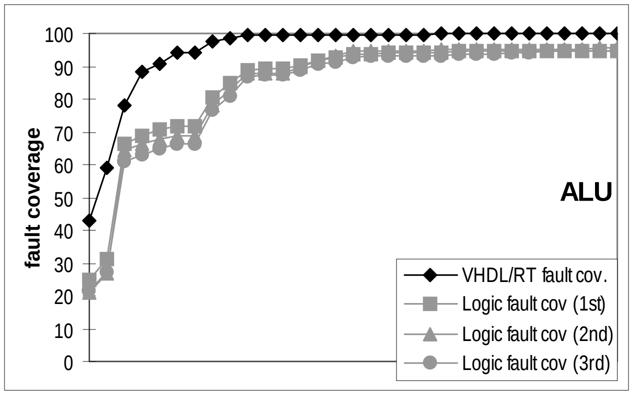
1. The fault list is extracted from the VHDL code and the defined fault model.
2. From the fault list and the input VHDL code, the perturbed codes are generated, following the insertion scheme presented before. As a result, there will be n (n = number of faults) faulty architectures of the entity under test, plus the free-fault architecture.
3. All the architectures (fault-free and faulty ones) are simulated in parallel, using a commercial VHDL simulator (Synopsys VSS). The output values of all these architectures are stored in arrays that are compared with the fault-free output value to indicate the detection of faults. These comparisons are made at given times, defined as strobe windows, that can be defined by the user. In the same VHDL simulation file,

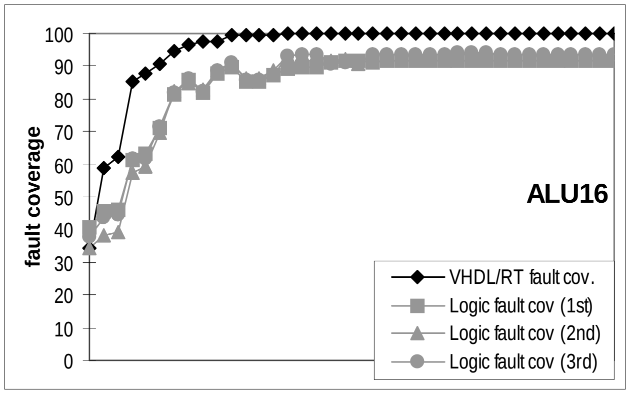
and as a result of the comparisons, the achieved fault coverage is computed.

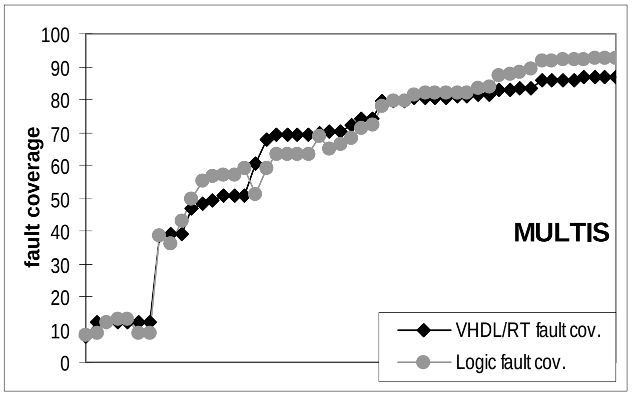
Most of these steps are automatically performed as some C programs have been developed to generate the faulty codes, the simulation file, etc. The process is not clearly optimised in simulation time but it gives an easy way to compute the fault coverage for the VHDL/RT proposed fault model.

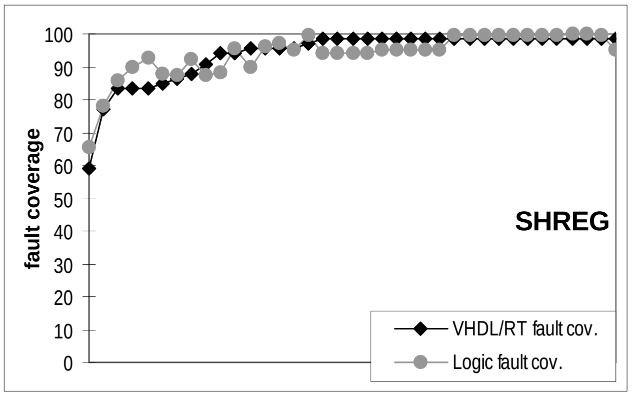
The test vectors used for most of these simulations have been randomly generated. In some cases, the initialisation signals have been manually generated, because the achieved fault coverages were so low. It has been decided to use random test vectors because they are not better suited for logic nor high level test, although it is known that real test is not made with random vectors due to their low quality (specially for sequential circuits).

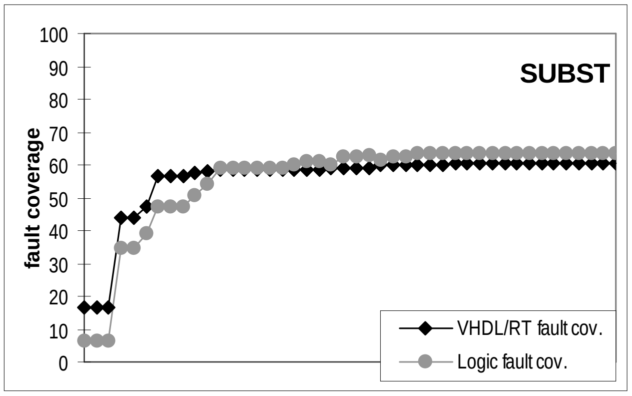
The following figures show the experimental results obtained for the selected examples. In these graphs, the VHDL/RT and logic fault coverages are represented for each experiment. In the first two examples three different logic fault coverages are shown, corresponding to the different logic circuits obtained in the synthesis processes. At the end, the results are summarised showing the average error, which measures the differences between the logic and the VHDL/RT fault coverages, and the correlation coefficient, which measures the deviation of tendency between both data series (the closer to 1, the better).

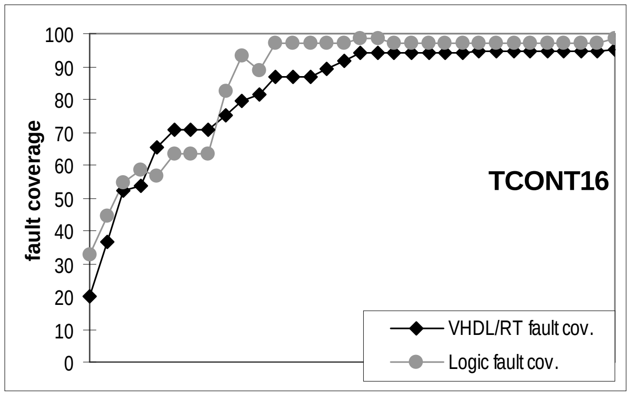












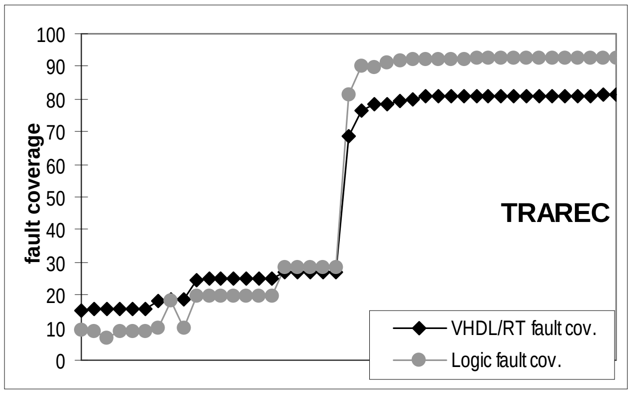


Figure 5. Experimental results

Circuit name	*Average error*	*Correlation coef.*
ALU (1st synthesis)	12.0 %	0.956
ALU (2nd synthesis)	12.9 %	0.944
ALU (3rd synthesis)	14.3 %	0.935
ALU16 (1st synthesis)	11.9 %	0.933
ALU16 (2nd synthesis)	10.9 %	0.929
ALU16 (3rd synthesis)	10.5 %	0.932
MULTIS	7.7 %	0.985
SHREG	3.1 %	0.907
SUBST	10.3 %	0.974
TCONT16	8.4 %	0.961
TRAREC	20.3 %	0.998

Table 2. Summary of the experimental results

As general conclusions of the experimental results obtained with the set of examples it is possible to state:

* The fault model presented does not give a precise value of the fault coverage achieved at the lower levels. However, it gives an estimation of the fault coverage achieved with a set of test vectors using pre-synthesised descriptions: low fault coverages at the RT level imply low fault coverages at the logic level.
* The quality of a set of test vectors can be estimated with the proposed method from RT descriptions, as it has been shown how RT and logic fault coverages are correlated for the selected set of examples. Therefore, the increase of fault coverage at the RT level is similar to the increase at logic gate level with the same set of test vectors (the correlation coefficients are close to 1).
* For those circuits with a large combinational part, the estimation of fault coverage at RTL is “optimistic” and the synthesis options have an influence on the accuracy of the estimation. This is due to the fact that in these cases it is difficult to represent the logic faults at the RT level. However, we find “pesimistic” results in highly sequential circuits (as *TRAREC*). In this case the error is due to a large number of faults at the RT level that are very difficult to detect or even undetectable.
* The achieved results are better than the ones presented in previous fault models [6], [7], as they get better correlation coefficients (larger than 90%). The errors were not reported in the previous references. The main difference is the use of single-bit fault models for integer data types and arrays which make this fault model more oriented to the hardware represented by the VHDL code, as the synthesis process has been taken into account in its definition.

4. Conclusions

A fault model for VHDL descriptions at the register transfer level has been presented. The fault model has been defined in a systematic way, analysing all the objects that can appear on the considered VHDL subset (Level-0 subset). This fault model will allow the designer to have an estimation of the achieved fault coverage before the circuit synthesis is made and therefore, to take some decisions on the test strategy to be followed before closing the architectural design. The experimental results presented in the paper have been run on a set of examples to measure the precision of the fault model compared to the logic level fault models. The results show a high correlation between both fault coverages and a tight relationship between them.

In order to accelerate the VHDL/RT fault simulation process, it would be necessary to build a VHDL fault simulator from the beginning. This means a new elaboration procedure and a new simulation cycle to be able to inject faults in the VHDL descriptions. At the moment, this is not available although there have been interesting contributions to this field as in [9].

5. References

[1] C.F. Hawkins, H.T. Nagle, R.R. Fritzemeier, J.R. Guth, “The VLSI Circuit Test Problem - A Tutorial*”, IEEE Trans. on Industrial Electronics*, vol. 36, pp. 111-116, May 1989.

[2] R. L. Wadsack, “Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits”, *Bell System Technical Journal*, vol. 57, May-June 1978.

[3] J. M. Soden, C. F. Hawkins, R. K. Gulati, W. Mao, “I_{DDQ} Testing: A Review”, *Journal of Electronic Testing: Theory and Applications*, Kluwer Academic Pub., December 1992.

[4] T. Riesgo, Y. Torroja, J. Uceda, “Design Process Based on Hardware Description Languages”, *Proc.ISIE’94*, Santiago de Chile, May 1994.

[5] B. Bennetts, “Test Synthesis: Towards Higher Levels of Abstraction”, *Proc. 2nd Archimedes Workshop on Synthesis of Testable Circuits*, Barcelona (Spain), February 1995.

[6] J. R. Armstrong, F. Lam, and P. C. Ward, "Test Generation and Fault Simulation for Behavioral Models." In Joel M. Schoen (Ed.*), Performance and Fault Modelling with VHDL*, pp. 240-303, Prentice Hall, 1992,.

[7] S. Gosh, T. J. Chakraborty, “On Behavior Fault Modeling for Digital Designs”, *Journal of Electronic Testing: Theory and Applications*, Kluwer Academic Pub., pp 135-151, 1991.

[8] ESIP, *Deliverable #203: Standardization Activities: The Synthesis Package*, July 1994.

[9] M. García, I. Hidalgo, P. Sánchez, “Simulación de fallos en un entorno VHDL”, *Proc. DCIS’94*, Gran Canaria (Spain), November 1994. (*in Spanish*)