Introduction to Fault analysis at Logic Level – An Educational Approach based on DSCH

Belgacem Hamdi [(1)], Etienne Sicard [(2)]

(1) ISSAT, Monastir – Tunisia, Belgacem.Hamdi@issatgb.rnu.tn
(2) *(2) INSA-Dgei, University of Toulouse – France, etienne.sicard@insa-toulouse.fr*

Abstract—This paper presents an educational approach to practice fault analysis at the gate level of digital circuits by means of a specially designed fault injection block. The technique allows injection of single stuck-at fault at the nodes of the circuit. This tool is integrated to DSCH3, and allows the logic simulation of basic blocs in the presence of faults, as well as determining the fault coverage of a set of test vectors.
This cooperative work consists in introducing DFT tools and fault analysis capabilities, in order to improve the skills of students in the field of integrated circuit testing.

***Index Terms* — Integrated circuit design, Testing, Design for Testability, logical stuck-at fault model, DFT CAD educational tools.**

I. Introduction

The development of the teaching of microelectronics in Tunisia follows the international curricula. At the beginning the course was based on the characterization of materials, the technological processes like on the design of the devices and basic cells for ASICs and design flow. Currently it touches the various nano and micro aspects and new electronic devices. It relates to the preceding fundamental aspects in addition to the modern flow of design for reconfigurable supports or not. In this flood of design, the most used tools for description are them: VHDL, VHDL-AMS, SystemC. Initially the didactic tools were based on tools of CAD in version of demonstration of software or on university versions, as published in [1-3].

However, new helps with education programs or agreements of companies like Mentor, AMS… or association protocols with CMP [4] and Europractice [5] open to our university new opportunity with tools accessibility. At our days, in the University of Monastir and Sousse (ISSAT and ENISO), the graduates in master of microelectronics are more than 200. Majority of these students continued in doctorate and already supported there PhD.

On the other hand, the increasing complexity of VLSI circuits, Systems on-Chip (SOC) or even Networks-on-Chip (NOC) has made test generation one of the most complicated and time-consuming problems in digital design. The more complex are getting electronics systems, the more important will be the problems of test and design for testability because of the very high cost of testing electronic products. At present, most system designers and electronics engineers know little about testing, so that companies frequently hire test experts to advise their designers on test problems, and they even pay a higher salary to the test experts than to their VLSI designers. This reflects also today's university education: IC design is widely introduced, but only truly dedicated students learn test. The next generation of engineers involved with System-on-Chip (SoC) technology should be made better aware of the importance of test, and trained in test technology to enable them to produce high quality and defect-free products.

This paper introduces the context of microelectronics education in Tunisia (section 2), gives the theoretical backgrounds of the tool (section 3), and proposes a tool overview as well as an illustration on simple case studies (section 4), followed by a conclusion.

II. Microelectronics Education in tunisia

On April 2008, the CNFM (French Center of Training in Microelectronics) [6] signed a framework agreement for collaboration with the Universities of Sousse and Monastir and the Technology pole of Sousse, Tunisia. The agreement was signed in Tunis in the Mediterranean Forum Business Development-MAD-ALLIA. This framework agreement expresses the intention to develop training, research and technological innovation in the field of microelectronics in Tunisia, as well as increase mobility between the various agencies involved in the two countries.

Specifically, it plans to establish in Tunisia, various measures based on the experience of CNFM network in France: pooling resources and expertise, provision of software, training of trainers, education days, welcoming students and trainers in the central technology of open CNFM, etc.. The partners are also committed to working together to obtain financing including through tender or programs promoting international relations.

This collaboration is one aspect among others who express the desire for Tunisia to take forward in the field of microelectronics. This dynamics of teaching and the formation by research allowed the attraction of foreign industrial activity in engineering in Microelectronics like ST Microelectronics (300 engineers in 2007), ALCATEL, SAGEM (3200 people)… or national industry like TELNET...

This industrial activity was consolidated by the creation of a new technological poles specialized in Microelectronics in Sousse (third city in Tunisia) after that of the technological pole El Ghazala in Tunis although specialized in telecommunication.

On the other hand, DSCH3 and Microwind3 [7] tools have been used from several years in Monastir, Gabes and Sfax (since 1998) in their free version, and, more recently, in ISSAT of Sousse which, actively use DSCH3 and Microwind3 (in their complete version) as a “tester user”. Labs based on DSCH3 and Microwind3 are introduced and taught in a course entitled "Advanced Design" [8][9] in the department of electronic engineering as an option. All students chose this option, find the content very interesting and their feed-back is very positive.

III. THEORETICAL BACKGROUND FOR THE TOOL

Design of logic integrated circuits in CMOS technology is becoming more and more complex since VLSI is the interest of many electronic IC users and manufacturers. A common problem to be solved by designers, manufacturers and users is the testing of these ICs.

A. Testing an IC

Testing an integrated circuit can be expressed by checking if its outputs correspond to the inputs applied to it. If the test is positive, then the system is good for use. If the outputs are different than expected, the IC is rejected (Go/No Go test). A diagnosis may be applied to it, in order to point out and identify the problem's causes.

Testing is applied to detect faults after several operations: design, manufacturing, packaging, as illustrated in figure 1. If a test strategy is considered at IC level, the fault can be detected at early system design stages, located and eliminated at a very low cost. When the faulty chip is soldered on a printed circuit board, the cost of fault remedy would be multiplied by ten. And this cost factors continues to apply until the system has been assembled and packaged and sent to final users, as illustrated in Figure 2.

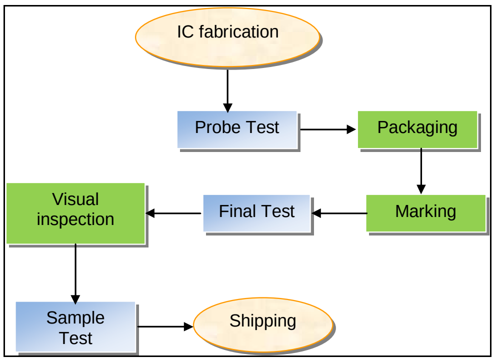


Figure 1: typical IC production flow

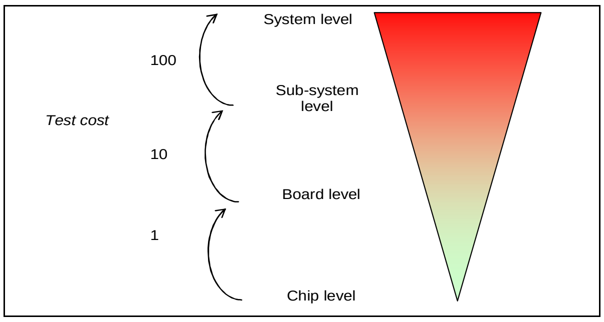


Figure 2: Test cost (The rule of ten)

The first idea to test an N input circuit would be to apply an N-bit counter to the inputs (controllability), then generate all the 2N combinations, and observe the outputs for checking (observability). This is called "exhaustive testing" (Fig. 3), and it is very efficient, but only for few- input circuits. However, this technique becomes very time consuming when the input number increases. Given a set of faults in the circuit under test (CUT), our goal is to obtain the smallest possible number of test patterns which guarantees the highest fault coverage. Test compaction refers to the process of reducing the number of test patterns in a test set without reducing its fault coverage.

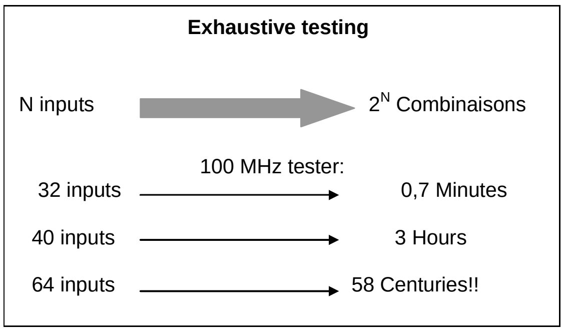


Figure 3: The exhaustive testing time becomes prohibitive with a large number of IC inputs.

A test pattern (or test vector) for a fault *f* in a circuit *C* is an input combination for which the output(s) of *C* is different when *f* is present than when it is not (Fig. 4). A test vector *x* detects fault *f* if:

$$\boxed{C(x) \oplus C_f(x) = 1}$$

Where: *C(x)* is the response of the fault free circuit, and $C_f(x)$ is the response of the faulty circuit.

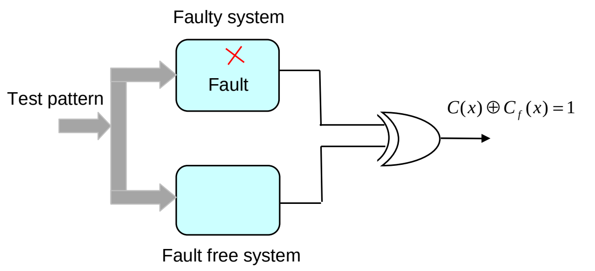


Figure 4: A test pattern detects a fault if the fault free response is different from the faulty response.

B. Fault Testing

1) Fault model

Failure modes are manifested on the logical level as incorrect signal values. A fault is a model that represents the effect of a failure by means of the change that is produced in the system signal. Several defects are usually mapped to one fault model, and it is called a many-to-one mapping. However, some defects may also be represented by more than one fault model. Fault models have the advantage of being a more tractable representation than physical failure modes. It is possible to mark most commonly used fault models (Table 1).

Fault Model	Description
Single stuck-at faults (SSF)	One line takes the value 0 or 1.
Multiple stuck-at faults (MSF)	One, two or more lines have fixed values, not necessarily the same.
Bridging faults	Two or more lines that are normally independent become electrically connected.
Delay faults	A fault is caused by delays in one or more paths in the circuit.
Intermittent faults	Caused by internal parameter degradation. Incorrect signal values occur for some but not all states of the circuit. Degradation is progressive until permanent failure occurs.
Transient faults	Incorrect signal values caused by coupled disturbances. Coupling may be via power bus capacitive or inductive coupling. Includes internal and external sources as well as particle irradiation.

Table 1: Most commonly used fault models

As a model, the fault does not have to be an exact representation of the defects, but rather, to be useful in detecting the defects. For example, the most common fault model assumes single stuck-at (SSF) lines even though it is clear that this model does not accurately represent all actual physical failures. The rational for continuing to use stuck-at fault model is the fact that it has been satisfactory in the past. In addition, test sets that have been generated for this fault type have been effective in detecting other types of faults. However, as with any model, a fault cannot represent all failures. Further will be discussed a bit closer the fault models that have been brought in Table 1.

2) Stuck-at-faults

As it was mentioned earlier, a single stuck-at fault (SSF) represents a line in the circuit that is fixed to logic value 0 or 1. We consider here permanent faults that are faults that are continuous and stable, whose nature do not change before, during, and after testing. These faults are affecting the functional behavior of the system permanently. These faults are usually localized and can be modeled. Other faults such as temporary faults or intermittent faults are not considered in this application note.

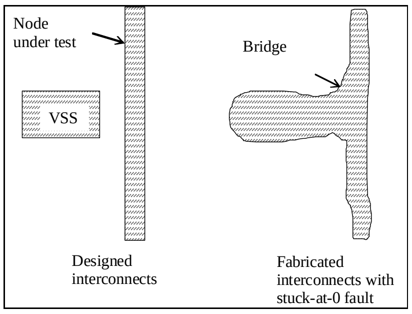


Figure 5: Physical origin of a node fault stuck at 0.

Fig. 5 illustrates a possible origin for a node stuck at 0 voltage: the implementation is close to a VSS node (here situated close, same layer), and a faulty metal bridge makes a robust connection to the ground.

3) Other faults

The manufacturing of interconnects may result in interruptions or short-cuts, which may have catastrophic consequences on the behavior of the integrated circuit. Fig. 6 illustrates the case of “Open” and “Short” faults, not considered in DSCH for fault simulation.

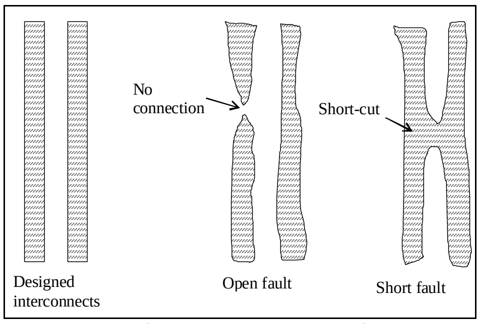


Figure 6: Physical origin of the “Open” and “Short” fault.

Many other faults are also considered in the literature: transistor stuck-on and stuck-open faults interconnect transition and delay faults, etc... These faults are not considered in this work. Independent of how accurately the stuck-at fault represents the physical defect, we next continue investigating how to generate patterns that detect these faults.

4) Testing and fault coverage

Testing is the process of determining whether a device functions correctly or not. The question is: How much testing of an IC is enough? The Yield (Y) is defined as the ratio of the number of good dies per wafer to the number of dies per wafer. Fault coverage (FC) is the measure of the ability of a test set T to detect a given set of faults that may occur on the DUT (Device Under Test). We shall try to achieve FC=1, that is a fault coverage of 100%.

$$\boxed{FC = (\#\text{detected faults})/(\#\text{possible faults})}$$

Defect level (DL) is the fraction of bad parts among the parts that pass all tests.

$$\boxed{DL = 1 - Y^{(1-FC)}}$$

Where FC refers to the real defect coverage (probability that T detects any possible fault in F or not) and DL is the DPM (defects per million). Typical values claimed are less than 200 DPM, or 0.02%.

IV. Tool Overview and Case Study

DSCH3 is software [8-9] for logic design, companion of Microwind, an educational tool for CMOS IC design [7]. Based on primitives, a hierarchical circuit is built and simulated. Interactive symbols are used to friendly simulation, which includes delay and power consumption evaluation.

In this work, we introduce the concept of fault, consider the Single Stuck-at Fault model (SSF), and show how these faults may be injected and simulated. Then, using DSCH, we show how to build a reference truth table, and how to simulate these faults applied to input and output nodes of the circuit under test. We investigate how test patterns detect these faults. The ultimate goal is to classify the efficiency of test patterns, in order to select the most efficient test vectors, and therefore reduce the number of test patterns.

A. *Fault simulation concepts*

1) *Introduction to fault simulation*

Fault simulation is performed during the design cycle to achieve the following goals:

- Testing specific faulty conditions
- Guiding the test pattern generator program
- Measuring the effectiveness of the test patterns

To perform its task, the fault simulation program requires, in addition to the circuit model, the stimuli, and the responses of a good circuit to the stimuli, a fault model and a fault list. As was mentioned earlier, there are different fault models, and the most widely used is the stuck-at model. Test patterns generated for this model have proven to be useful for other types of models, such as multiply stuck-at, bridging, and delay faults. The responses deduced by the fault simulator are used to determine the fault coverage.

In the fault simulation process, a fault is considered from the list and a pattern is applied to the circuit. If the fault is detected, it is dropped from the fault list and the next fault is considered. Otherwise, another pattern is applied, and if the fault is not detected when all patterns are applied, the fault is then considered undetectable by the test and is removed from the fault list. The process is continued until the fault list is empty.

Another way to perform fault simulation is to consider first, the fault free circuit, simulate it and extract a reference truth table. After that, simulate faulty circuit by injecting faults one by one, and for each fault extract a faulty truth table that should be compared with the reference table in order to find out test vectors for the considered fault. This is done until the fault set is empty.

2) *Fault simulation result*

The output of a fault simulator separates faults into several fault categories. If we can detect a fault at a location, it is a testable fault. A testable fault must be placed on a controllable net, so that we can change the logic level at that location from 0 to 1 and from 1 to 0. A testable fault must also be on an observable net, so that we can see the effect of the fault at a primary output (PO). This means that uncontrollable nets and unobservable nets result in faults we cannot detect. We call these faults untested faults, untestable faults, or impossible faults. In this section we investigate the testing of two circuits, a Nand-Or combination and a full-adder.

B. *Case study1: Fault injection in Nand-Or Circuit*

1) *Manual fault injection*

The Nand-Or circuit is a simple combination of a 2-input NAND gate and a OR gate. The concept of manual fault injection is presented in fig. 7. The fault injection at a node N consists in opening the connection and inserting a multiplexor circuit. An example of s@0 and s@1 injection circuit is proposed, based on two multiplexors, one for the function mode (normal function/fault injection mode), the second for the injected fault type (s@0/s@1). A manual implementation in DSCH is reported in Fig. 8 and the corresponding simulation in Fig. 9, with both normal and fault injection.

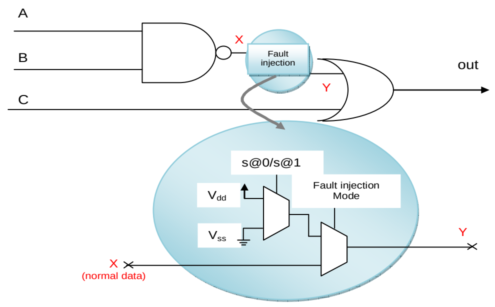


Figure 7: Fault injection principles

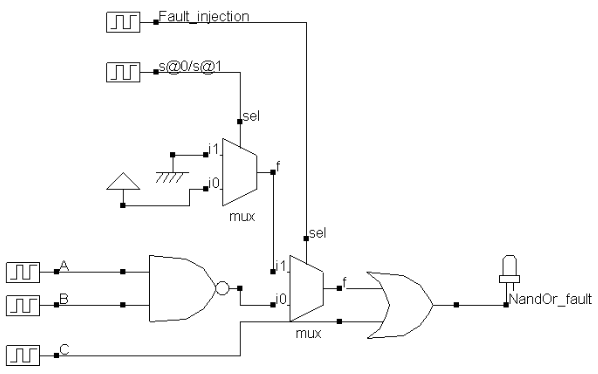


Figure 8: Manual fault injection in a NAND-OR circuit

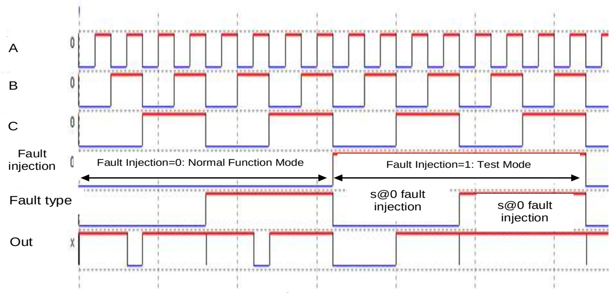
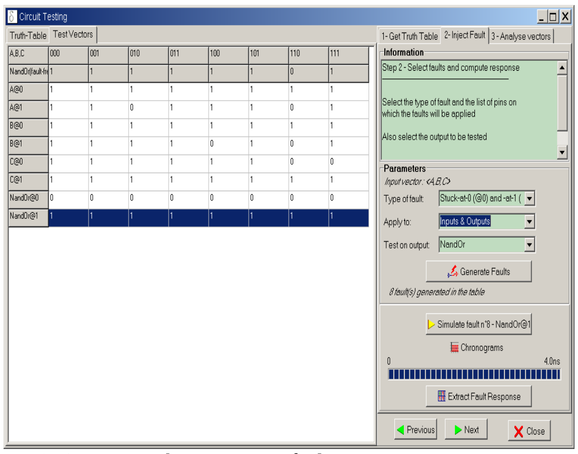


Figure 9: Simulation (test/NandOr_fault.SCH)

2) *Automatic fault injection*

The NandOr circuit has five nodes, therefore 10 possible stuck-at faults, if we also consider the internal node linking the NAND2 output to the OR input. The automatic Fault Analysis Tool is executed by following the next steps. First, the student launches the logic simulation which feeds the table with the values obtained in the chronograms of the circuit logic simulation. Second, the student specifies the type of fault and injection nodes: the student selects the type of fault “s@0 & s@1”, and applies it to “All nodes” (Fig. 10).



A,B,C	000	001	010	011	100	101	110	111
NandOr(fault-fre	1	1	1	1	1	1	0	1
A@0	1	1	1	1	1	1	1	1
A@1	1	1	0	1	1	1	0	1
B@0	1	1	1	1	1	1	1	1
B@1	1	1	1	1	0	1	0	1
C@0	1	1	1	1	1	1	0	0
C@1	1	1	1	1	1	1	1	1
NandOr@0	0	0	0	0	0	0	0	0
NandOr@1	1	1	1	1	1	1	1	1

Information

Step 2 - Select faults and compute response

Select the type of fault and the list of pins on which the faults will be applied

Also select the output to be tested

Parameters

Input vector : <A,B,C>

Type of fault: Stuck-at-0 (@0) and -at-1 (

Apply to: Inputs & Outputs

Test on output: NandOr

Generate Faults

8 fault(s) generated in the table

Simulate fault n°8 - NandOr@1

Chronograms

0 4.0ns

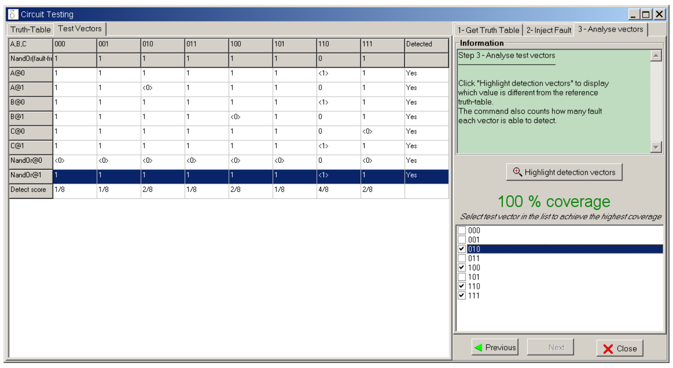
Extract Fault Response

Previous Next Close

Figure 10: Computing the response to fault injection

The student generates the fault list and pilots the simulation which evaluates the consequence of all faults one by one. The logic values are transferred to the corresponding line. The student repeats the last two steps until the table is completed (Fig. 10). By a click on “Highlight Detection Vectors”, the result is shown in Fig. 11.

From Fig. 11, it can be seen that the vector 110 detects A@0, B@0, C@1 and NandOr@1 (4/8 faults). The vector 100 detects B@1 and NandOr@0 (2/8 faults). The vector 010 detects the fault A@1. The vector 111detects the remaining fault C@0. Therefore 4 test vectors (010, 100, 110, 111) detect all stuck-at faults (test time will be the half of an exhaustive test).



A,B,C	000	001	010	011	100	101	110	111	Detected
NandOr(fault-fre	1	1	1	1	1	1	0	1	
A@0	1	1	1	1	1	1	<1>	1	Yes
A@1	1	1	<0>	1	1	1	0	1	Yes
B@0	1	1	1	1	1	1	<1>	1	Yes
B@1	1	1	1	1	<0>	1	0	1	Yes
C@0	1	1	1	1	1	1	0	<0>	Yes
C@1	1	1	1	1	1	1	<1>	1	Yes
NandOr@0	<0>	<0>	<0>	<0>	<0>	<0>	0	<0>	Yes
NandOr@1	1	1	1	1	1	1	<1>	1	Yes
Detect score	1/8	1/8	2/8	1/8	2/8	1/8	4/8	2/8	

Information

Step 3 - Analyse test vectors

Click "Highlight detection vectors" to display which value is different from the reference truth-table.
The command also counts how many fault each vector is able to detect.

Highlight detection vectors

100 % coverage

Select test vector in the list to achieve the highest coverage

- ☐ 000
- ☐ 001
- ☑ 010
- ☐ 011
- ☑ 100
- ☐ 101
- ☑ 110
- ☑ 111

Previous Next Close

Figure 11: Vector detection efficiency evaluation, showing a 100% fault coverage

V. Conclusion

This paper has described an educational feature added to DSCH linked with fault injection and simulation at logic level. The mechanisms for logic fault injection, simulation and optimum test vector extraction have been described and illustrated on case-studies. The tool will be introduced in Sept. 2010 in Tunisia and France for conducting practical trainings related to IC test and illustrated theoretical courses about logic gate and logic circuit testing. Future developments will concern a module for fault injection at the layout level using Microwind3. The idea is to model the failure with a spot interfering with lithography at a given process step, and then move it on the layout, simulate with SPICE and observe its effect on the electrical behavior of the circuit in comparison with the defect free circuit.

VI. References

[1] C. Hacker and R. Sitte, “Interactive Teaching of Elementary Digital Logic With WinLogiLab”, IEEE Trans. Education, Vol. 47, no. 2, May 2004, pp 196-203.

[2] C. J. Tseng “Digital System Design Using Microarchitectural Modeling”, IEEE Trans. Education, vol. 51, no. 1, Feb. 2008, pp. 93 – 99.

[3] J. S. Yuan and J. Di, “Teaching Low-Power Electronics Design in Electrical and Computer Engineering”, IEEE Trans. Education, Vol. 48, no. 1, Feb. 2005, pp 169-181.

[4] http://cmp.imag.fr

[5] http://www.europractice.com/

[6] http://www.cnfm.fr

[7] E. Sicard “Microwind User’s Manual, lite version 3.5”, www.microwind.org, INSA editor, 2009

[8] E. Sicard, S. Ben Dhia “Basic CMOS cell design”, November 2006 at McGraw-Hill, USA, ISBN: 007148839, DOI:10.1036/0071488391

[9] E. Sicard, S. Ben Dhia “Advanced CMOS cell design”, McGraw-Hill professional series, November 2006, ISBN: 0071488367, DOI:10.1036/0071488367