

Achieving Simulation–Based Test Program Verification and Fault Simulation Capabilities for Mixed–Signal Systems

Pascal Caunegre – Claude Abraham
SIEMENS AUTOMOTIVE
BP 1149, 31023 Toulouse, France.

Abstract

A simulation–based methodology for test program verification is presented. Executing test programs on a virtual test system allows simulation of Device Under Test (DUT) behavior. Simulating both device and test hardware and software allows test engineers to check and debug test programs without fabricated devices. It results in a gain of time and permits design and test to occur concurrently. This also yields better overall circuit testability.

Our effort concerns development of Automatic Testing Equipment (ATE) model and software interfaces to link the simulator with both design tools and test tools.

Further, we propose extending the general fault simulation concept to analog world. Fault models are suggested and tools for fault simulation automation have been developed. They include fault list generation, simulation control and result analysis as well.

1: Introduction

Testing devices after manufacturing requires to run on an ATE a specific test program applying stimuli and carrying out measurements on the DUT. During test development phase test engineer has to check the test program by verifying that [BUC89] :

- Tests pass good devices. The test software is checked on a sample of fabricated devices known as good devices (golden boards).
- Tests reject faulty devices. Typical faults are introduced one by one on a device and it is verified that the test software detects them.

These tasks generate delays coming from the wait–time for fabricated devices, to build the ATE/DUT hardware interface and to wait for the busy test system. As underlined by Austin [AUS93], the test program debug process is known to be the major source of delay in the development cycle of a new product. Because of these delays, test development occurs a long time after design development such that design and testing are presently disjointed tasks. Therefore, it is usually too late to modify a completed design in order

to improve the testability of the system. This leads in a more complicated and costly test program.

The project we present here was initiated to contribute to the challenges of reducing the time–to–market of new products, and ensuring high quality of manufactured products. As far as testing is concerned two improvements have been defined.

First, test engineering time can be reduced by enhancing the test debug process. Therefore we present a new methodology for test development aiming at the concurrent development of design and test. It is described in the section 2 of this paper.

Second, better product quality involves test quality improvements. Test quality may be checked against fault simulation. The section 3 of this paper presents the methods and tools we have developed for fault simulation automation.

This work is illustrated by an example in the section 4.

2: A new methodology for test program development

2.1: Test development methodology

Our methodology consist in running the test program on a simulated test system. The main idea is that the test program can be debugged on a work–station rather than on the actual test system by means of software tools capable of simulating the ATE, the DUT and of executing the test program. Instead of using an actual device, the schematic of the device allows the test program simulation and is available as soon as it has been designed provided that all the models exist.

The methodology we propose for test program development consists of the following steps :

- Once a functional block of a device has been designed, the designer uses the simulator to check his design, and then to compute the worst cases of operation. This analysis is also useful to determine test limits.
- A set of tests concerning the designed block is generated by the test engineer using a graphical test description lan-

guage.

- The test engineer uses the schematic tool to build the test configuration schematic. The device schematic is linked with ATE schematic.
- The simulation tool is invoked causing the translation of the previous schematic into a netlist file for the simulator while the test program is translated into a stimuli file.
- The simulation is then run. The virtual ATE supplies stimuli on the DUT model and performs measurements through its instrument models. These results are available for comparison to what was expected. Now the debug process can begin and the test engineer can use simulation facilities to view or plot the signals of each node of the circuit.
- If testing impossibilities occur, a design modification could be discussed with the designers.
- When all the tests have been debugged, the test program is compiled into an executable and sent to the ATE.
- The process of the design continues and leads to a sample of fabricated devices.
- A final check of the test on the actual ATE is performed to validate the simulation work and to solve some possible differences between virtual and physical test system.

Simulation is not only a way to start test development sooner, it also provides users with information on device/ATE interactions (impedance matching, settling time, transient effects...) and test program behavior (instruments setting, instruments precision, measurements synchronization...)

To achieve this, we have to build efficient models of our ATE. In addition, the test program must be understood by the simulator so that test engineers can use simulation with their familiar test environment. This involves building a link to exchange data between test program and simulator. Our work comprises two main tasks :

- achieving ATE and DUT simulation : it means to choose and check a simulation tool and to develop specific models,
- building the links to exchange data between the simulation tools and both the design data and the test tools.

2.2: Achieving ATE and DUT simulation

Simulation tools

The simulator Saber was chosen because it can simultaneously handle analog and digital circuitry [ANA94]. This is due to the existence in Saber of both a differential equation solver and an event-driven simulator. Digital and analog parts of a system are interfaced by "hypermodels" which are characterized according to the technology of the actual device (TTL, ECL, CMOS...)

Moreover, Saber provides a large digital and analog models library and allows different levels of modelling. While detailed and precise models are necessary for some parts, behavioral models may be sufficient for some others but allow a gain in computation time.

Mixed-signal device simulation poses two important questions. How to keep reasonable simulation time even for complex circuits ? How to simulate complex digital components such as micro-controllers ?

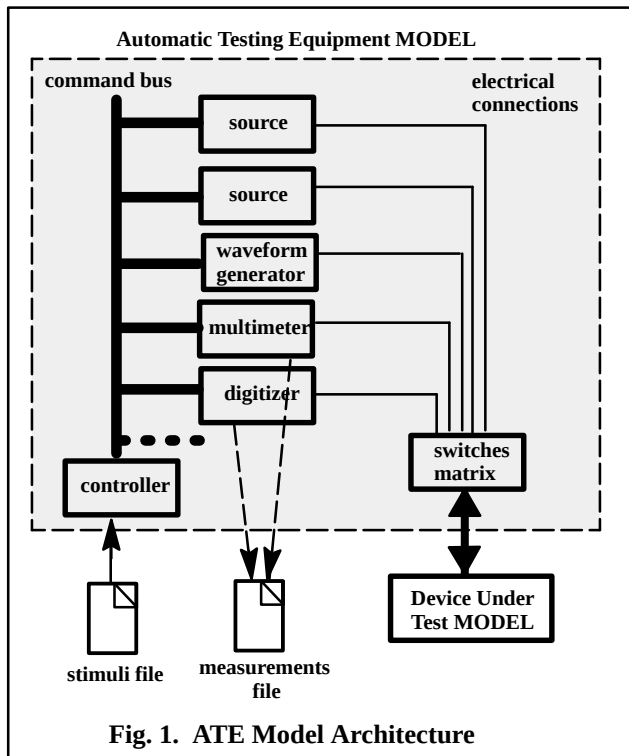
The solution proposed to solve the first problem is to partition the circuit as suggested in [ROS89]. Every complex circuit can be seen as a collection of smaller blocks or sub-circuits performing a given function. Since the simulation time increases with the circuit size, such partitioning could help save computation time. Nevertheless it could also be necessary to simulate some surrounding blocks if they are interacting with the block under test.

The second problem was to model complex components like micro-controllers which are commonly used in our products. In fact, during board test, micro-controllers perform simple functions such as reading a voltage on input ports, setting a voltage on output ports, receiving or sending messages on communication ports... So the model of a micro-controller used in limited, known configurations during test can be simply described by a behavioral model.

Finally, using Saber software with its mixed-signal capability and specific models that we have developed could meet our requirements for mixed-signal system simulation.

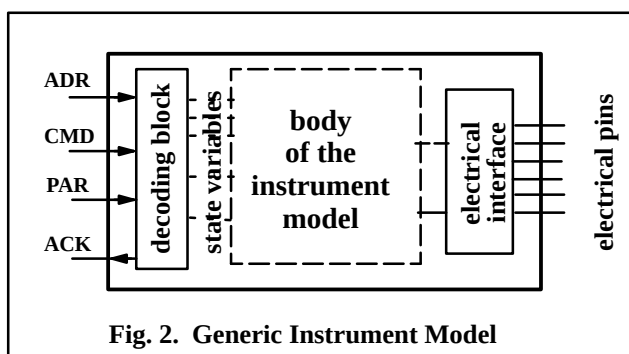
ATE Model

The ATE model follows the ATE architecture (see fig. 1) and is made up of many instrument models such as sources, waveform generator, multimeter, timer, communication board, switches matrix... Some of these instrument models carry out measurements or receive digital data from the DUT and relay this data to us. This is done by means of an output file where the instrument models write their measurement results. Stimuli are sent to the instrument models through a three line command bus encoding the Address of the instrument, the Command we want the instrument to perform, and a Parameter related to the command. A controller model reads a set of three coded stimuli values from the stimuli file and sets them on the bus generating an event. The stimuli is received and executed by the addressed instrument model and after it completes its task, it sends back an acknowledgement signal to the controller which fetches and sends a new stimuli. This process is continued until the end of the stimuli file is reached.



Additionally, we gave the instrument models a generic structure (see figure 2).

- They all own a decoding sub-model which is connected to the command bus. It analyses the received commands and discards the bad commands with an error message. If the commands are correct, it sets internal state variables. The behavior of the model depends on the values of the state variables. It also generates an acknowledgement signal after a delay depending on the executed command.
- An electrical interface sub-model is used to connect the analog input/output pins with the event-driven internal body of the model. This block takes into account electrical considerations like I/O impedance, filters and current throughput.
- Moreover, some instruments exchange data with external files by means of C sub-routines.



2.3: Linking test and design with simulation

Linking design and simulation

The schematic captured by designers is reused as an entry for the simulator in order to avoid any human mistakes or redundant work. Analogy provides an interface between schematic tool (Mentor) and simulator (Saber). Yet, generating an operational netlist requires us to add information in the components library of the schematic tool so that every component of the schematic library point to a particular model in the simulator library.

Additionally, we found it is useful to partition the large circuits before simulation in order to reduce computation time. To achieve this we developed a specific tool. It allows test engineer to select the block he wants to test and the surrounding blocks necessary for simulation. Then, our tool takes the initial netlist and simplifies it keeping only the selected blocks. After this process, continuity is checked. If disconnected nodes appear, the tool connects them to ground through an additional high value resistor in order to avoid aborted simulator executions. The simplified netlist is then available for simulation.

Linking test and simulation

Simulating test programs requires that the test-program data can be used by the simulator. Such a link is not easy to achieve because test programs are usually rather complicated and no standard test language exists although some standards have been proposed in the past years (e.g. ATLAS in the 70's or SCPI in the 80's). Each ATE manufacturer still uses his own test program language.

Nevertheless, the present trend is to propose graphical programming interfaces because they are more readable and easily understood than textual language and are usable without any lengthy session of training. This is, for example, the approach of Schlumberger which provides us with CATE (Computer Aided Test Engineering). With this tool, test engineers define test sequences by means of icons of instrument tasks. From this graphical test plan, CATE generates a test program in a specific language usable to run the ATE.

We built a tool capable of understanding this textual language and of producing a stimuli file for the simulation. Unfortunately, such a link is strongly dependant on the test program language and is not portable on the test system of an other ATE supplier.

3: Analog fault simulation

3.1: Purpose of fault simulation

Fault simulation is widely used in digital circuit design to help generate test vectors and to produce fault coverage evaluation [FUJ85].

For the mixed A/D systems designed by our company, test programs are evaluated by means of a manual fault cov-

erage measurement. Faults are physically introduced on a board, one at a time. The faulty device is then put on the ATE and the test program is run. It is checked if test program detects or not the defective device. If not the test program must be improved. Because of the number of components, and consequently the number of likely faults, this analysis takes a lot of time. Practically, it is restricted to the most important failures in terms of security or functionality; so minor faults are neglected.

If fault coverage is a good way to judge the test program effectiveness it is obvious that this analysis must be automatised. The extension of fault simulation to the analog world is still in progress. In the following, we first summarise some previous work on this subject. Then, a brief overview of usual defects is presented and fault models are suggested. Finally, specific tools for fault simulation automation are presented.

3.2: Previous work

While much research work and tools have addressed digital fault simulation, only a few have dealt with fault simulation for analogue circuit.

- FSPICE [REN85] is a SPICE-based tool to introduce faults at the circuit level and simulate these faulty circuits using SPICE. The major drawback is that circuit level simulation is too time-consuming for large circuits.
- A Discretized Analog Fault Simulator (DRAFTS) has been presented in [NAG93]. Its advantage is to be faster than circuit level simulator but it is unfortunately restricted to linear analog circuits.
- Anafault [SEB93] is an Eldo-based fault simulation tool where user-defined fault models can be introduced by adding appropriate sub-circuits in simulation file.
- Finally, the work we present here is an application of the Saber simulator to the fault simulation problem. The main advantage of this solution is to benefit from a high level simulator (as compared to a circuit level simulator) provided with an analog modelling language, and to benefit from a mixed-signal simulation capability.

3.3: Fault modeling

Defect assumptions

While faults in digital circuits are traditionally modelled by *stuck-at* (SA0 and SA1) and *stuck-open* models, faults in analogue circuits are classified in two categories [OHL91]:

- catastrophic faults
- deviation faults

Catastrophic (or hard) faults concern major modifications in the circuit like a short circuit (e.g. due to an excess of solder resulting in a bridge between two pins of a component) or open circuit (e.g. due to a lack of solder or a broken pin of a component)
Deviation (or soft) faults concern variations of component

parameters out of their tolerance (e.g. value of a resistor, gain of a bipolar transistor, transconductance of a MOS transistor...)

Even if most failures could be represented by hard faults, soft faults cannot be neglected. This is due to the analog nature of circuits and signals, involving accurate measurements [SLA92]. Output parameters characterizing analog circuits (steady state voltage or current, impedance, rise time, fall time, period, gain, frequencies...) are sensible to component parameters.

On the other hand, there is a greater number of parameters in a circuit so that it becomes prohibitive to take into account all the possible variations. Moreover, this would be of limited use since a lot of parameter deviations do not alter the circuit functionality. In fact, the problem of handling soft faults is choosing the efficient set of parameters to alter. Another problem is determining the amount we want to deviate each parameter from its nominal value. It may be, for example, twice the allowed tolerance.

Defect assumptions specific to printed circuit board system

Because our company is PCB technology oriented, we extend our defect assumptions list with faults due to printed circuit board manufacturing. The manufacturing process failure statistics gives the following faults :

- missing component : this fault is often equivalent to an open circuit. Yet, this case is not always trivial to test : e.g. the lack of a regulation capacitor is rather hard to detect.
- improperly oriented component : it occurs for polarized components like diodes.
- another type of assembly fault consists in a component put in the place of another. It occurs for components with the same geometry, e.g. SMT resistors and capacitors may be swapped.

Fault models

Here we look at modeling the previously described electrical failures. The general fault models we propose are as follows :

• Short-circuit model

A short-circuit is simulated by introducing a parallel small value resistor (1Ω typically or a user-defined value) into the circuit.

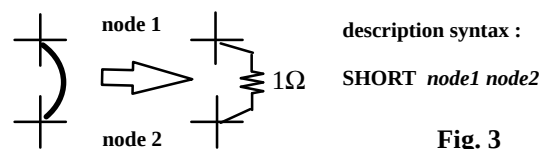


Fig. 3

• Open-circuit model

A faulty component is disconnected from a given node by introducing a serial high value resistor ($1G\Omega$ typically or a

user-defined value).

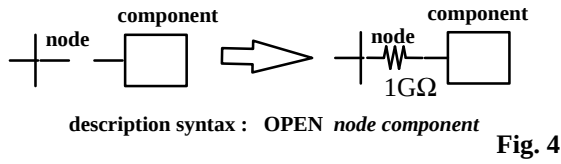


Fig. 4

Nevertheless, we enhanced this model by an additional option which consists in fixing the voltage (at either VCC or Ground) on the disconnected pin. Note that the resistor remains useful to keep circuit continuity.

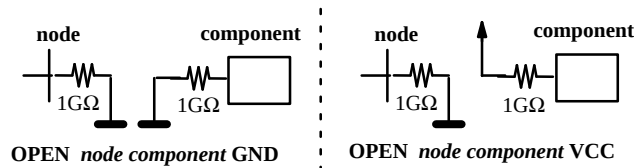


Fig. 5. Additional options for open circuit modelling

This feature may be helpful as shown in the following example. Assume we want to simulate a break in a nMOS transistor gate. If we merely insert a resistor, the gate voltage remains the same because no current flows through the gate (at low frequency) and then no break is simulated. With the previous model, gate voltage is set to 0V so that break is effective.

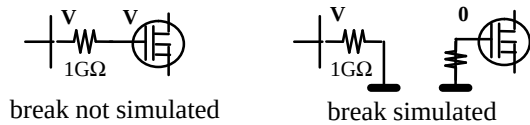


Fig. 6.

• Inversion and replace models

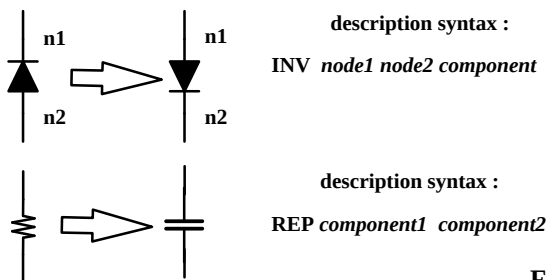


Fig. 7.

• Stuck-at model

This model allows simulation of electrical failures in digital parts. It consists of 1/ disconnecting the component from

the circuit, 2/ setting a logical "0" or "1" at the faulty pin.

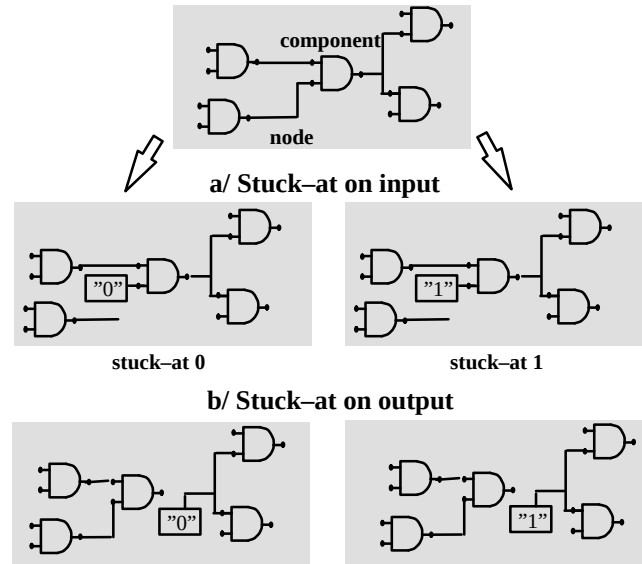


Fig.8.Digital fault models

• Soft fault model

It is used to alter a component parameter.

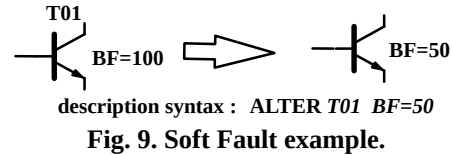


Fig. 9. Soft Fault example.

3.4: Tools for fault simulation automation

Because of the size of the circuits and the number of faults, faults cannot be manually introduced and simulated. Our effort consists in building tools to automatize fault simulation without modifying the core simulator (Saber). With a few modifications, these tools could run with other simulators as well. The tools we developed for this purpose are described below.

• Fault list generator.

The function of this tool is to provide a list of possible faults from a given circuit schematic. As an input, a netlist describing the circuit is required. A simulator format netlist or an EDIF standard format netlist may generally be provided by the schematic tool. The tool searches for appropriate fault models in a *component fault catalog* for each component referenced in the circuit netlist. The fault models are then instantiated, i.e. the actual names of nodes and components are introduced. It provides a fault list.(see fig. 10). During fault list generation, equivalent faults are reduced in order to prevent redundant simulations. Reduction rules have been implemented :

- shorts applied to two parallel components are equivalent and result in the same fault,
- opens applied to two serial components are also equivalent.

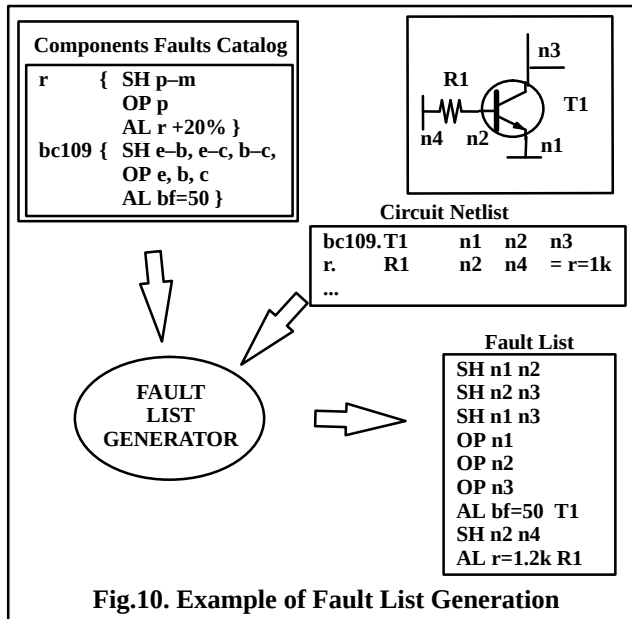


Fig.10. Example of Fault List Generation

Components Faults Catalog.

This catalog completed and updated by a quality expert, provides a generic description of the fault models to be used for each library component.

Note that several fault models may describe the same physical failure. The expert will choose the best model, in terms of accuracy or simulation time.

For instance, let us consider a nMOS transistor (see fig. 11).

A drain break failure could be simulated by several models:

- it could be seen as a drain (1) or a source (2) open, since $I_d=0$ implies $I_s=0$,
- it could be seen as a non-conducting transistor modelled by a grounded gate (3),
- it could be modelled by increasing the turning on voltage V_{on} parameter (4) so that the transistor will never be conducting...

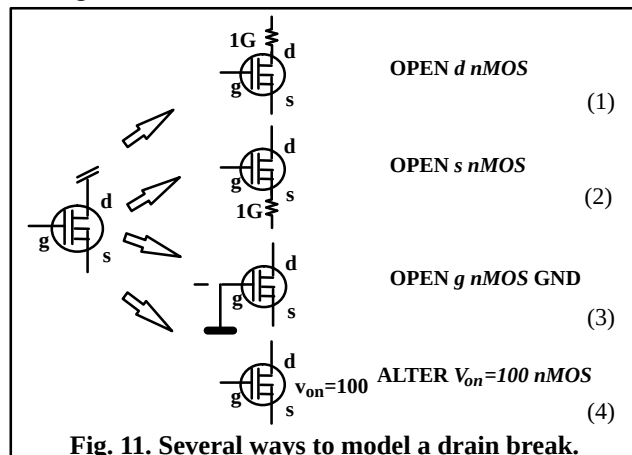


Fig. 11. Several ways to model a drain break.

• Simulation control file generator.

This tool creates two command files by processing the previously built reduced fault list.

One of these files contains a command script to control the simulator. When it is executed, the simulator is automatically run. At each fault to process, a topology modification (for hard faults) is automatically introduced in the original netlist according to the fault model. At each run, the Saber simulator is invoked with the faulty netlist which is re-compiled.

The other file contains parameter modifications commands (for soft faults) which are introduced in the simulator without netlist re-compilation.(Fig.12).

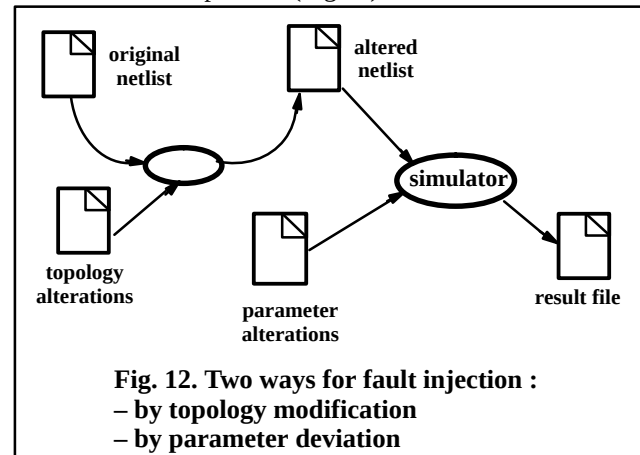


Fig. 12. Two ways for fault injection :

- by topology modification
- by parameter deviation

Once these files have been generated fault simulation may be started and will work alone. During simulation process measurements are performed on the circuit by instrument models (as described in the first part) and are recorded in a result file. In fact, for each measurement two values are stored depending on the measurement accuracy. For instance, if 10V is measured with a 2% instrument precision the couple (9.8V ; 10.2V) will appear in the result file.

• Fault coverage analyzer.

This tool determines if the test detected each fault by comparing the simulation measurements with the expected values. Expected values may be obtained from a "gold simulation", i.e. with a fault-free circuit. A tolerance margin is affected to these values, resulting in a lower limit and an upper limit of allowed values. Detection or no detection of the fault is evaluated as follow (see also fig. 13).

If the measurement interval is outside of the allowed interval (1) it means the test rejects the device for a given fault in the circuit, that is the fault is well detected. On the other hand, if the measurement falls inside the allowed interval (2), the test passes and the fault will never be detected. In the intermediate case (3) the detection will depend upon measurement fluctuation. We consider the fault as not detected.

Finally, the fault coverage ratio n_d/n_s is computed with the number of detected faults n_d and the number of simu-

lated faults n_s and a report is generated. A weighted fault coverage ratio is also computed taking into account the occurrence factor for each fault (because faults may have different probabilities to appear). This report underlines the critical failures that test will not detect. It is a base for the test improvement.

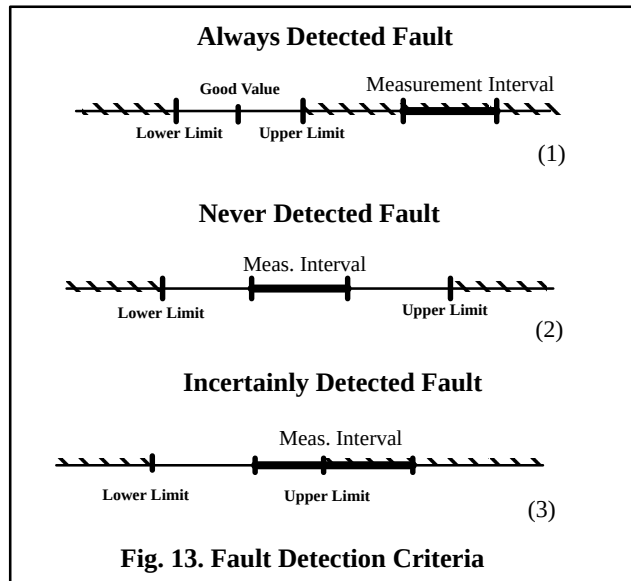


Fig. 13. Fault Detection Criteria

4: Application example

As an example, let us take a very simple mixed-signal circuit performing a pulse modulation function (similar to the famous 555 circuit). Figure 14 gives the circuit schematic. To build the test system configuration, we added to the schematic the specific ATE/DUT interface circuitry and we connected the ATE symbol using a voltage stimuli channel and a frequency measurement channel. A strobe output allows us to see when the measurements are really executed. The test program to be verified is shown on the same figure. It consists of applying a steady-state voltage of successively 2V, then 3V and carrying two frequency measurements. A simulation of this test system provides a textual report of the measurements and a graphical plot of the node activities as shown on the figure 15.

During fault simulation, shorts and opens are applied to resistors, comparators, and transistors. Deviation of $\pm 40\%$ are also applied on resistors. Stuck-at 0 and 1 are applied to digital parts. This lead to 54 potential faults. Figure 16 shows the fault simulation report. For each fault, the measured values are reported. Here test1 and test2 correspond to the frequency value at the times 100us and 200us. These values are compared to the test limits which have been set by applying a 20% tolerance to gold simulation results (respectively 51492 Hz and 33457 Hz for test1 and

test2). The non-detected faults are underlined in the report. The fault coverage ratios are computed (0.833). (Note that in our example the occurrence factor was left to 1 for all the faults.)

5: Conclusion

The test program development methodology we first proposed aims to achieve concurrent design and test development. The existence of a mixed-signal simulator has been helpful but specific models of ATE instruments were needed.

Additionally, a software interface to transfer test data from the test tool to the simulation tool also had to be developed. This effort allows test engineers to debug their test programs without waiting for a fabricated device thus reducing the overall product development cycle.

Finally, we added a new capability in the test engineer tool box : a fault simulation tool providing an evaluation of test program efficiency. Using the previous simulation workbench, this tool targets analog or mixed-signal circuits.

Acknowledgements

Thanks to our colleagues Malcom Grant and Mike O'Dell for reviewing this paper.

References

- [ANA94] Analogy Inc. Beaverton, Oregon : Saber Reference Manual, release 3.3, 1994.
- [AUS93] Tom Austin : Creating a mixed-signal simulation capability for concurrent IC design and test program development, International Test Conference, 1993, pp 125-132.
- [BUC89] Allen Buckroyd : Computer Integrated Testing, BSP Professional books, 1989.
- [FUJ85] H. Fujiwara : Logic testing and design for Testability, Siljthoff & Noordhoff, MIT Press Computer System Series, 1985.
- [NAG93] N. Nagi, A. Chatterjee, J. Abraham : Fault Simulation of Linear Analog Circuits, Journal of Electronic Testing , vol.4 1993, pp. 345-360.
- [OHL91] M. Ohletz : Hybrid Built-in Self Test for Mixed Analogue/Digital Integrated Circuits, proc. European Test Conference, 1991, pp. 307-316.
- [REN85] M. Renovell, G. Cambon, D. Auvergne : FSPICE : a tool for fault modelling in MOS circuits, Integration VLSI, Vol.3, 1985, pp. 245-255.
- [ROS89] E. Rosenfeld : Issues for Mixed-Signal CAD-Tester Interface. International Test Conference 1989, pp. 585-590.
- [SLA93] M. Slamani, B. Kaminska : Testing Analog Circuits by Sensitivity Computation. IEEE European Conf. on Design Automation, 1992, pp. 532-537.
- [SEB93] C. Sebecke, M. Ohletz : Simulation Models for the Fault Simulation on single Elements of Analogue Circuits. Archimedes, Esprit III project 7107 task 3.2 Analogue Fault Simulation, 1993.

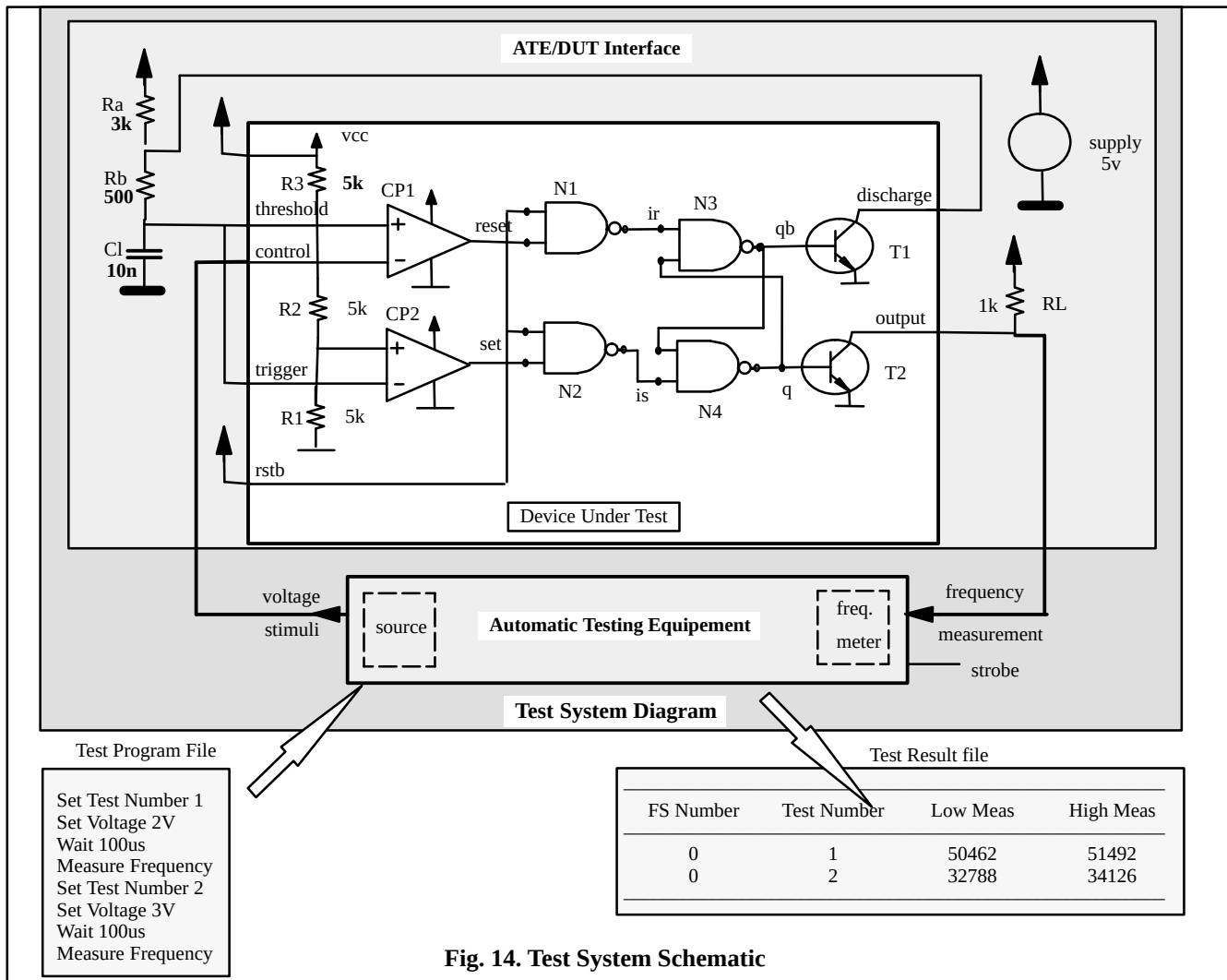


Fig. 14. Test System Schematic

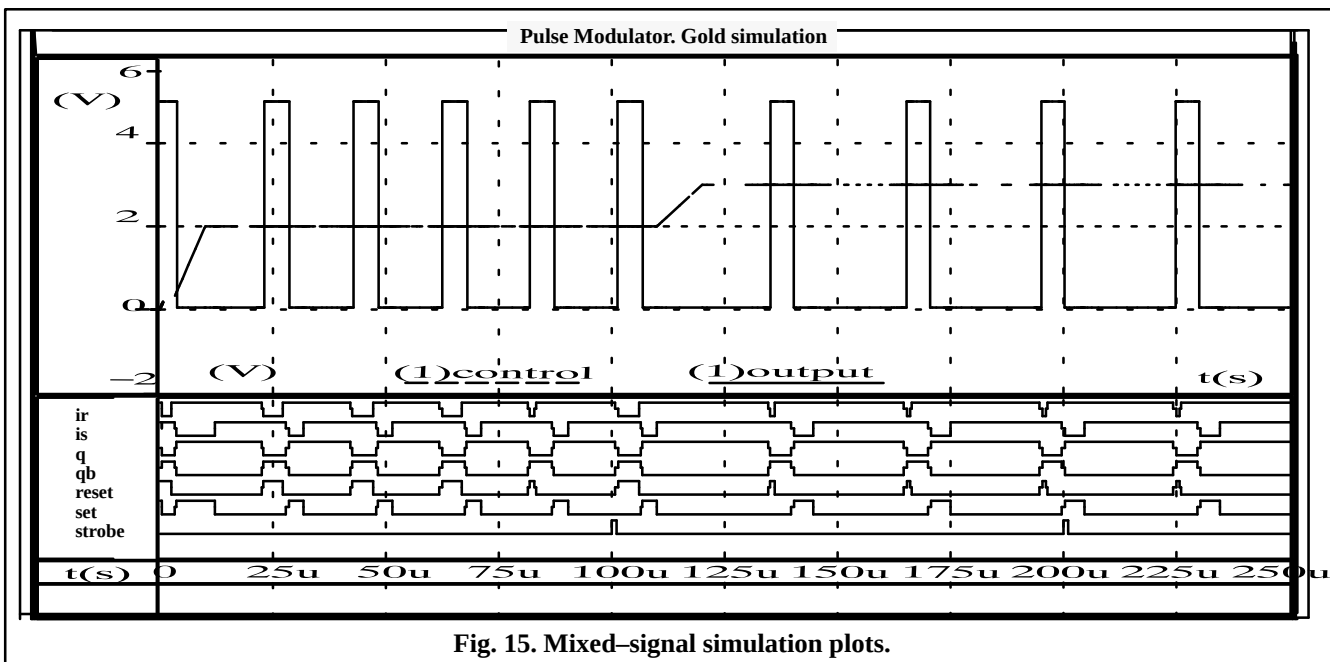


Fig. 15. Mixed-signal simulation plots.

***** FAULT SIMULATION REPORT *****																			
FAULT LIST GENERATION / REDUCTION					FAULT INJECTION														
54 potential faults.					12 faults injected by ALTER.														
6 simplified faults (including 0 inconsistent faults.)					36 faults injected by netlist modification.														
48 faults to be simulated .					0 faults cannot be injected.														
FAULT COVERAGE ANALYSIS of the Pulse Modulator Circuit.																			
Evaluation :	Test1	Lower limit : 41194			Upper limit : 61789														
	Test2	Lower limit : 26766			Lower limit : 40148														
Detected Fault Number : 45, Weighted : 45					Simulated Fault Number : 54, Weighted : 54														
FAULT COVERAGE of the Test Program: 0.833					Weighted : 0.833														
TESTS ->		1	2	DETECTION OCCUR		TESTS ->		1	2	DETECTION OCCUR									
FAULTS :						FAULTS :													
SH_p-m/R1		0	0		2		1	SH_c-e/T2		0	0		2		1				
OP_p/R1		6.8e+04	6.41e+04		2		1	OP_c/T2		0	0		2		1				
AL_r1.4/R1		>> 5.36e+04 <<>>	3.59e+04 <<		0	<<<	ND		1	OP_b/T2		0	0		2		1		
AL_r0.6/R1		>> 4.42e+04 <<>>	2.92e+04 <<		0	<<<	ND		1	SA1_in1/N1		0	0		2		1		
SH_p-m/R2		6.7e+04	6.22e+04		2		1	SA1_in2/N1		>> 5.07e+04 <<>>	3.28e+04 <<		0	<<<	ND		1		
OP_p/R2		0	0		2		1	SA1_out/N1		0	0		2		1				
AL_r1.4/R2		>> 4.6e+04 <<>>	3.08e+04 <<		0	<<<	ND		1	SA0_in1/N1		0	0		2		1		
AL_r0.6/R2		>> 5.97e+04 <<>>	3.77e+04 <<		0	<<<	ND		1	SA0_in2/N1		0	0		2		1		
SH_p-m/R3		0	0		2		1	SA0_out/N1		0	0		2		1				
OP_p/R3		>> 5.07e+04 <<>>	3.29e+04 <<		0	<<<	ND		1	SA1_in1/N2		0	0		2		1		
AL_r1.4/R3		>> 5.08e+04 <<>>	3.29e+04 <<		0	<<<	ND		1	SA1_in2/N2		>> 5.11e+04 <<>>	3.29e+04 <<		0	<<<	ND		1
AL_r0.6/R3		>> 5.06e+04 <<>>	3.28e+04 <<		0	<<<	ND		1	SA1_out/N2		0	0		2		1		
SH_inp-inm/CP1		0	0		2		1	SA0_in1/N2		0	0		2		1				
SH_inp-out/CP1		0	0		2		1	SA0_in2/N2		0	0		2		1				
SH_inm-out/CP1		0	0		2		1	SA0_out/N2		0	0		2		1				
OP_inp/CP1		0	0		2		1	SA1_in1/N3		0	0		2		1				
OP_inm/CP1		0	0		2		1	SA1_in2/N3		7.07e+04	8.5e+04		2		1				
OP_out/CP1		0	0		2		1	SA1_out/N3		0	0		2		1				
SH_inp-inm/CP2		0	0		2		1	SA0_in1/N3		0	0		2		1				
SH_inp-out/CP2		0	0		2		1	SA0_in2/N3		0	0		2		1				
SH_inm-out/CP2		0	0		2		1	SA0_out/N3		0	0		2		1				
OP_inp/CP2		0	0		2		1	SA1_in1/N4		0	0		2		1				
OP_inm/CP2		0	0		2		1	SA1_in2/N4		0	0		2		1				
OP_out/CP2		0	0		2		1	SA1_out/N4		0	0		2		1				
SH_c-e/T1		0	0		2		1	SA0_in1/N4		0	0		2		1				
OP_c/T1		0	0		2		1	SA0_in2/N4		0	0		2		1				
OP_b/T1		0	0		2		1	SA0_out/N4		0	0		2		1				
The Tests which do not detect the fault are quoted by >> <<																The Tests which do not certainly detect the fault are quoted by > <			

Fig. 14. Fault Simulation Report