Development and optimization tests for detection fabrication faults circuits

Contributor: Belousov A.V.
E-mail: andbelousov@gmail.com

INTRODUCTION

Today's telecommunications, consumer electronics and other demanding market segments, require that more complex, faster and denser circuits are designed in shorter times and at lower costs. Obviously, the ultimate goal is to maximize profits.

However, the development of reliable products cannot be achieved without high quality methods and efficient mechanisms for testing circuits and integrated systems. To ensure the quality required for product competitiveness, one can no more rely on conventional functional tests: a move is needed towards methods that search for manufacturing defects and faults occurring during a system's lifetime. Moreover, to achieve acceptable fault coverage has become a very hard and costly task for external testing: test mechanisms need to be built into integrated circuits early in the design process. Ideally, these mechanisms should be reused to test for internal defects and environmental conditions that may affect the operation of the systems into which the circuits will be embedded. This would provide for amplified payback. To give an estimate about the price to pay for faults escaping the testing process, fault detection costs can increase by a factor of ten, when moving from the circuit to the board level, then from the board to the system level, and lastly, from the final test of the system to its application in the field.

Therefore, design-for-test seems to be the only reasonable answer to the testing challenges posed by state-of-the-art integrated systems. Mechanisms that allow for accessibility to internal test points, that achieve on-chip test generation, on-chip test response evaluation, and that make it possible the detection of errors concurrently to the application, are examples of structures that may be embedded into circuits to ensure system testability. They obviously incur penalties in terms of silicon overhead and performance degradation. These penalties must definitely account for finding the best trade-off between quality and cost. However, the industrial participation to recent test standardization initiatives confirms that, in many commercial applications, design-for-test can prove economical.

Within this context, the aim of this chapter is to give a glimpse into the area of design-for-test of integrated circuits and systems. Digital and analog circuits are considered. First of all, the test methods of interest to existing design-for-test techniques are revisited. Fault modeling, fault simulation and test generation are thus addressed. Then, design-for-testability, built-in self-test and self-checking techniques are discussed and illustrated in the realm of integrated circuits and printed circuit boards. Finally, the extension of these techniques to integrated systems is discussed, considering multi-chip module implementations, microsystems embedding micro-electro-mechanical structures, and core-based integrated circuits.

The organization of a test environment.

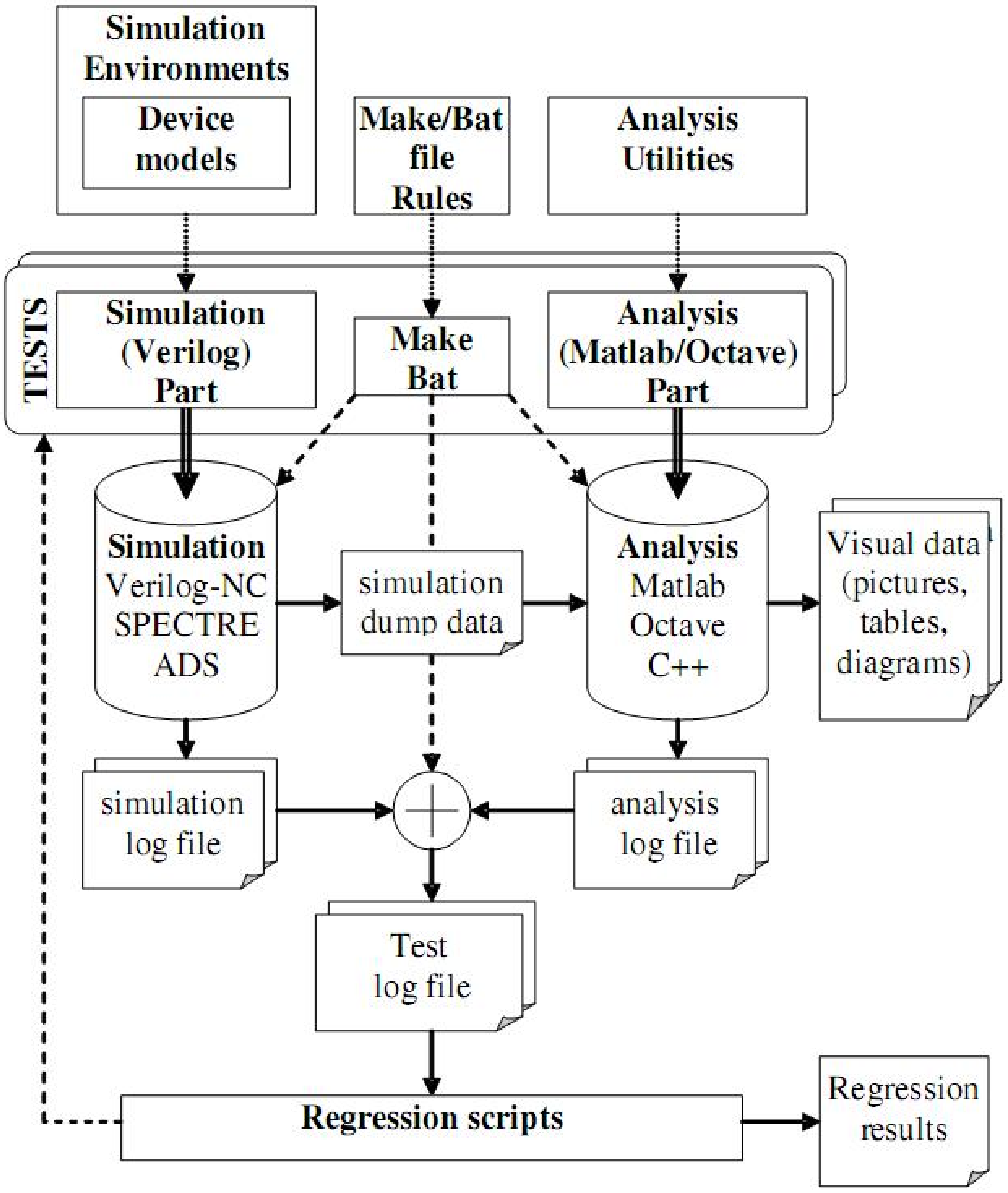
In the design mixed-signal circuits it's necessary to test every analog and digital module. Such tests has functional and parameter verification. For high quality verification it's has to do an integration test of whole circuit, rather than each module separately. While modeling whole scheme, it could be found faults which didn't detected in modular tests. These faults, mostly, related to data exchange between modules. For another thing, carrying out of testing of all scheme as a whole will allow to carry out the algorithmic analysis, and also define influence of blocks against each other and choose optimum structure of the scheme.

The main complexity of verification of analogue schemes - it is great volume of mathematical calculations at modeling extracted SPICE-list of chains. With increase of complexity of projects, it becomes impossible to spend modeling of all chip at transistor level[1]. In similar cases the decision to use the microcircuit description in various possible combinations is made. For example, it is possible to replace the simple analogue block Verilog with the description (dummy). If at testing more exact model is required then any analogue dependence is added (Verilog becomes AMS the description). If it's required as much as possible check approached to the real device, it is used SPICE-description. Similarly for a digital part if the digital part is so bulky that tightens test performance it is possible to use RTL (Register transfer level) description. But for a digital part the synthesized list of chains with a file of delays is usually used. The choice of a concrete variant depends on a demanded parity speed/quality though use of one of variants (for example sometimes it is technically impossible to describe any device on Verilog).

For convenience of modeling it is necessary to design correctly the simulation environment. It is convenient to create additional level of hierarchy round the tested device in which the modules necessary for work of analogue blocks (for example would contain, for testing of a ring of phase-locked loop (PLL) are necessary the filter of low frequencies and voltage-controlled oscillator). In a following level of hierarchy digital modules for generation of signals of management and the monitors fixing changes on any event should settle down. Each test includes some steps, each of them has own statistics about a performance course, for this purpose serve connected utilities, such as calculation of number of errors, preventions and a conclusion of messages. In addition for the test it is necessary to connect a file with some general utilities which would generalize statistics on all steps. Such test can be started in regression mode when results of modeling are estimated by certain criteria and the result stands out in a kind "HAS passed" / "HAS not passed". Use of such test in an interactive mode is possible. In this case from the test there is an unloading of the data in a separate file, then are processed. In such mode results of modeling are deduced in a kind convenient for a visual estimation: schedules, tables, diagrams, numerical values of parameters, etc. By means of such organization of the environment of simulation and a test environment it is possible to achieve enough high test coverage.

After functional verification, microcircuit manufacture, it is necessary for checking up on presence of manufacturing defects. When you perform manufacturing testing, you ensure high-quality integrated circuits by screening out devices with manufacturing defects. You can thoroughly test your integrated circuit when you adopt structured design-for-test (DFT) techniques. The DFT techniques currently supported by TetraMAX ATPG consist of internal scan (both full scan and partial scan) and boundary scan.

Fig. 1 Environment of verification.



Why Perform Manufacturing Testing?

Functional testing verifies that your circuit performs as it was intended to perform. For example, assume you have designed an adder circuit. Functional testing verifies that this circuit performs the addition function and computes the correct results over the range of values tested. However, exhaustive testing of all possible input combinations grows exponentially as the number of inputs increases. To maintain a reasonable test time, you must focus functional test patterns on the general function and corner cases. Manufacturing testing verifies that your circuit does not have manufacturing defects by focusing on circuit structure rather than functional behavior. Manufacturing defects include problems such as the following:

■ Power or ground shorts
■ Open interconnect on the die caused by dust particles
■ Short-circuited source or drain on the transistor caused by metal spike-through

Manufacturing defects might remain undetected by functional testing yet cause undesirable behavior during circuit operation. To provide the highest-quality products, development teams must prevent devices with manufacturing defects from reaching the customers. Manufacturing testing enables development teams to screen devices for manufacturing defects. A development team usually performs both functional and manufacturing testing of devices.

What Are Fault Models?

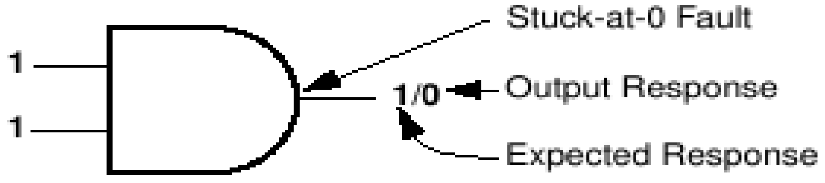
Efficient tests can only be produced if realistic fault models, based on physical failure mechanisms and actual layouts, are considered. Many defects may be inherent to the silicon substrate on which the integrated structures will be fabricated. Those may result from impurities found in the material used to produce the wafers, for example. Others may be due to problems occurring during the various manufacturing steps: the resistivity of contacts, for instance, will depend on the doping quality; the presence of dust particles in the clean room or in the materials may lead to the occurrence of spot defects; the misalignment of masks may result in deviations of transistor sizes, etc. All these defects lead, in general, to faults that simultaneously affect several devices, i.e. multiple faults. Other defects occur during a circuit's lifetime. They are usually due to failure mechanisms associated to transport and electromechanical phenomena, thermal weakness, etc. In general, those defects produce single faults. Permanent faults, like interconnect opens and shorts, floating gates, etc, can be produced by defects resulting from manufacturing and circuit usage. Transient faults, on the contrary, will appear due to intermittent phenomena, such as electro-magnetic interference or space radiations.

In the last decades, many models have appeared that tried to properly translate into faults actual physical defects (Wadsack, 1978; Galay, 1980; Courtois, 1981; Rajsuman, 1992). In the digital domain, gate inputs and outputs stuck-at a logical 0 or 1, always conducting (stuck-on) or never conducting (stuck-open) transistors and slow-to-rise and slow-to-fall gates (gate delays) are examples of device fault models. Shorts (resistive or not) between wires (bridging), stuck-open wires, and the cumulative delay of gates and interconnects in the critical path (path delay) are examples of interconnect fault models. Specific fault models also exist for memories that include, for example, couplings of memory cells and transition faults. Obviously, highly complex microprocessors cannot be analyzed under the lights of these fault models. Hence, those circuits rely on fault models that are based on the functionality of their instruction sets (functional fault model). In general, the existing fault models are complementary, in the sense that none can fully represent all possible physical defects.

Stuck-At Fault Models

The stuck-at-0 model represents a signal that is permanently low regardless of the other signals that normally control the node. The stuck-at-1 model represents a signal that is permanently high regardless of the other signals that normally control the node. For example, Figure 2 shows a two-input AND gate that has a stuck-at-0 fault on the output pin. Regardless of the logic level of the two inputs, the output is always 0.

Figure 2 Stuck-at-0 Fault on Output Pin of 2-input AND Gate



Detecting Stuck-At Faults

The node of a stuck-at fault must be controllable and observable for the fault to be detected.

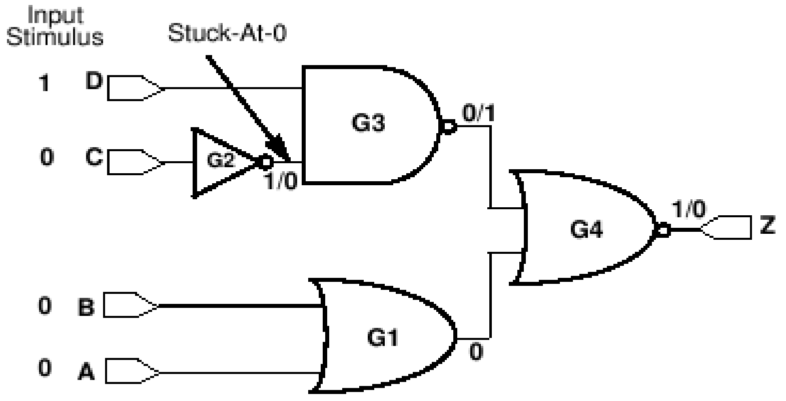
A node is controllable if you can drive it to a specified logic value by setting the primary inputs to specific values. A primary input is an input that can be directly controlled in the test environment. A node is observable if you can predict the response on it and propagate the fault effect to the primary outputs where you can measure the response. A primary output is an output that can be directly observed in the test environment. To detect a stuck-at fault on a target node, you must do the following:

■ Control the target node to the opposite of the stuck-at value by applying data at the primary inputs.

■ Make the node's fault effect observable by controlling the value at all other nodes affecting the output response, so the targeted node is the active (controlling) node.

The set of logic 0s and 1s applied to the primary inputs of a design is called the input stimulus. The set of resulting values at the primary outputs, assuming a fault-free design, is called the expected response. The set of actual values measured at the primary outputs is called the output response. If the output response does not match the expected response for a given input stimulus, the input stimulus has detected the fault. To detect a stuck-at-0 fault, you need to apply an input stimulus that forces that node to 1. For example, to detect a stuck-at-0 fault at the output the two-input AND gate shown in Figure 1, you need to apply a logic 1 at both inputs. The expected response for this input stimulus is logic 1, but the output response is logic 0. This input stimulus detects the stuck-at-0 fault. This method of determining the input stimulus to detect a fault uses the single stuck-at fault model. The single stuck-at fault model assumes that only one node is faulty and that all other nodes in the circuit are good. This type of model greatly reduces the complexity of fault modeling and is technology independent. In a more complex situation, you may need to control all other nodes to ensure observability of a particular target node. Figure 3 shows a circuit with a detectable stuck-at-0 fault at the output of cell G2.

Figure 3 Simple Circuit With Detectable Stuck-at Fault



To detect the fault, you need to control the output of cell G2 to logic 1 (the opposite of the faulty value) by applying a logic 0 value at primary input C. To ensure that the fault effect is observable at primary output Z, you need to control the other nodes in the circuit so that the response value at primary output Z depends only on the output of cell G2. For this example, you can accomplish these goals as follows:

■ Apply a logic 1 at primary input D so that the output of cell G3 depends only on the output of cell G2. The output of cell G2 is the controlling input of cell G3.

■ Apply logic 0s at primary inputs A and B so that the output of cell G4 depends only on the output of cell G2. Given the input stimuli of A = 0, B = 0, C = 0, and D = 1, a fault-free circuit produces a logic 1 at output port Z. If the output of cell G2 is stuck-at-0, the value at output port Z is a logic 0 instead. Thus, this input stimulus detects a stuck-at-0 fault on the output of cell G2.

This set of input stimuli and expected response values is called a test vector. Following the process previously described, you can generate test vectors to detect stuck-at-1 and stuck-at-0 faults for each node in the design.

Using Fault Models to Determine Test Coverage

One definition of a design's testability is the extent to which that design can be tested for the presence of manufacturing defects, as represented by the single stuck-at fault model. Using this definition, the metric used to measure testability is test coverage. For larger designs, it is not feasible to analyze the test coverage results for existing functional test vectors or to manually generate test vectors to achieve high test coverage results. Fault simulation tools determine the test coverage of a set of test vectors. ATPG tools generate manufacturing test vectors. Both of these automated tools require models for all logic elements in your design to calculate the expected response correctly. Your semiconductor vendor provides these models.

Fault Simulation

Fault simulation determines the test coverage of a set of test vectors. It can be thought of as performing many logic simulations concurrently: one that represents the fault-free circuit (the good machine) and many that represent the circuits containing single stuck-at faults (the faulty machine). Fault simulation detects a fault each time the output response of the faulty machine is a non-X value and is different from the output response of the good machine for a given vector. Fault simulation determines all faults detected by a test vector. By fault simulating the test vector that is generated to detect the stuck-at-0 fault on the output of G2 in Figure 3, it becomes clear that this vector also detects the following single stuck-at faults:

- Stuck-at-1 on all pins of G1 (and ports A and B)
- Stuck-at-1 on the input of G2 (and port C)
- Stuck-at-0 on the inputs of G3 (and port D)
- Stuck-at-1 on the output of G3
- Stuck-at-1 on the inputs of G4
- Stuck-at-0 on the output of G4 (and port Z)

You can generate manufacturing test vectors by manually generating test vectors and then fault-simulating them to determine the test coverage. For large or complex designs, however, this process is time consuming and often does not result in high test coverage.

Automatic Test Pattern Generation

ATPG generates test patterns and provides test coverage statistics for the generated pattern set. For now, consider the term "test vector" to be the same as "test pattern." ATPG for combinational circuits is well understood; it is usually possible to generate test vectors that provide high test coverage for combinational designs. Combinational ATPG tools can use both random and deterministic techniques to generate test patterns for stuck-at faults. By default, TetraMAX only uses deterministic pattern generation; using random pattern generation is optional. During random pattern generation, the tool assigns input stimuli in a pseudo-random manner, then fault-simulates the generated vector to determine which faults are detected. As the number of faults detected by successive random patterns decreases, ATPG can change to a deterministic technique. During deterministic pattern generation, the tool uses a pattern generation algorithm based on path-sensitivity concepts to generate a test vector that detects a specific fault in the design. After generating a vector, the tool fault- simulates the vector to determine the complete set of faults detected by the vector. Test pattern generation continues until all faults either have been detected or have been identified as undetectable by the algorithm.

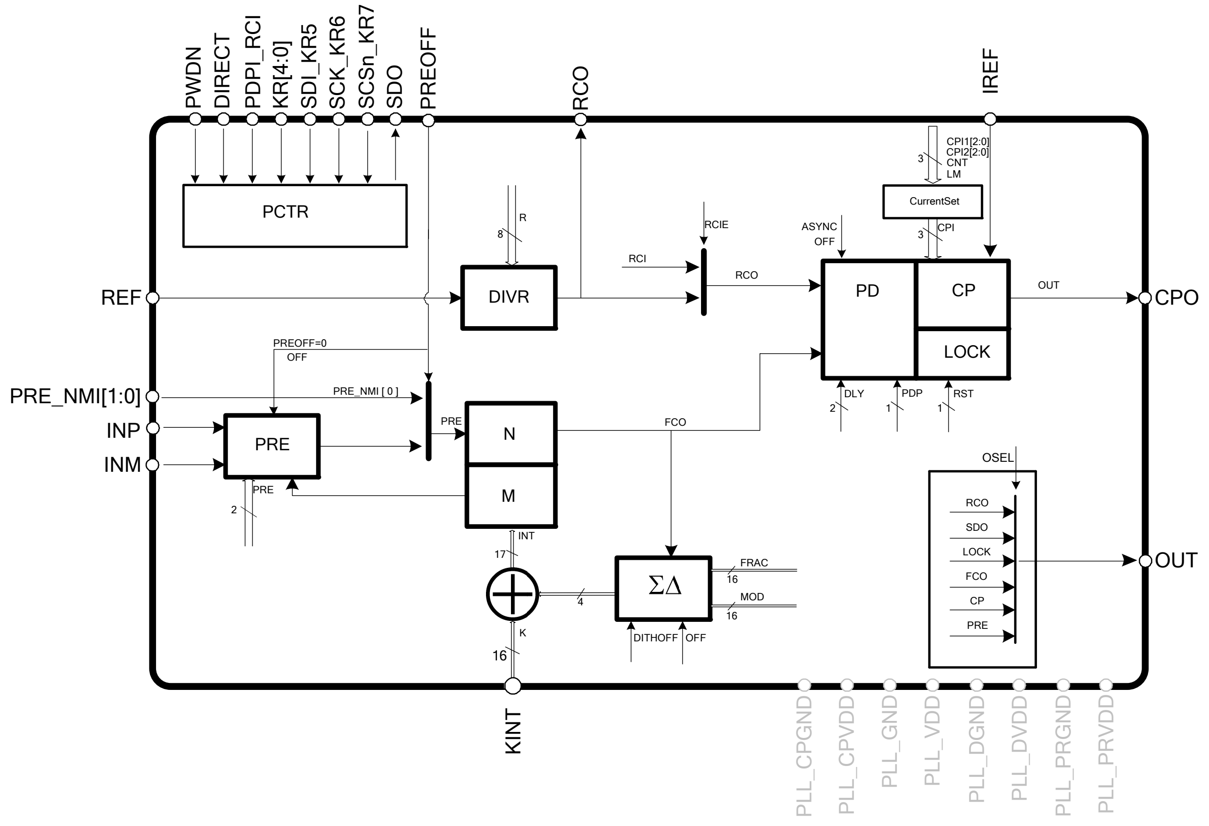
How to be, if technology ATPG for any reasons cannot be applied in the project???

So in the project of synthesizer PLL (Fig. 5 The functional diagram of VLSI PLL) there are rigid frameworks on the occupied area the scheme on a crystal, and as on consumed energy. For this reason it became necessary working out of test scanned chains manually. Six various modes of testing, and as transition of all scheme in a test mode, by means of an operated code word in the interface used by the scheme have been provided. Thus, after transfer of all scheme in a test mode, it is necessary to choose which test mode it is necessary to start, that is to choose on what tested blocks to push generated by functional tests test-vector, how many steps to push a vector, and what scanned chains will be scan_out. Then the test vector on an exit is checked, being compared to result of the similar target channel in the functional test.

As a result the real economy of the area occupied with a crystal has turned out. It is reached by a manual writing and an arrangement on a crystal of multiplexers, and as disposal from used in ATPG interface JTAG, and use of for testing own interface of scheme SPI.

At present work on options of use of automatically generated test vectors in conditions of not automatically created model of technology ATPG, for expansion of a test covering, or for comparison with already available is conducted.

Fig 4 The functional diagram of VLSI PLL.



PWDN
DIRECT
PDPI_RCI
KR[4:0]
SDI_KR5
SCK_KR6
SCSn_KR7
SDO
PREOFF
RCO
IREF
PCTR
REF
DIVR
PRE_NMI[1:0]
INP
INM
PRE
N
M
ΣΔ
PD
CP
LOCK
CurrentSet
CPO
OUT
KINT
FCO
RCI
RCIE
ASYNC OFF
OSEL
FRAC
MOD
DITHOFF
INT
DLY
PDP
RST
PLL_CPGND
PLL_CPVDD
PLL_GND
PLL_VDD
PLL_DGND
PLL_DVDD
PLL_PRGND
PLL_PRVDD