Generation of Digital System Test Patterns Based on VHDL Simulations

Miljana SOKOLOVIĆ[1], *Andy KUIPER*[2]

[1] LEDA laboratory, Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia & Montenegro
[2] Dept. of Radioelectronics, Brno University of Technology, Purkynova 118, 61200 Brno, Czech Republic

miljana@venus.elfak.ni.ac.yu, xkuipe00@stud.feec.vutbr.cz

Abstract. *In this paper an approach for test pattern generation and verification for a digital system-on-chip is proposed. It is based on digital system simulation using a standard VHDL simulator and on an additional program in MATLAB, that generates minimal test set for covering all stuck-at defects in the circuit. The approach is verified for two large arithmetic blocks which are parts of an integrated power-meter and represent large combinational digital systems. This approach is very useful because it can offer automatic minimal test set generation for a particular circuit, and speed up the IC design and testing process, which are essential for nowadays IC industry.*

Keywords

Testing, VHDL, stuck-at fault, minimal test set

1. Introduction

Integrated Circuit (IC) fabrication process consists of many different steps such as photolithographic printing, etching, doping, implanting, masking and chemical vapor depositioning. After carrying out these steps, a complete IC is obtained. IC surfaces are exaggerated in diagrams in order to distinguish between different layers of oxide, polysilicon and metal. On the contrary, in reality, they are not at all flat. Even with exaggerations, the diagrams represent an idealized approximation of actual fabricated circuit structures [1]. The actual circuit structures are not nearly as well defined as textbook diagrams would lead one to believe. Cross sections of real integrated circuit expose a variety of nonideal physical characteristics that are not entirely under the semiconductor manufacturer's control. Thus, no fabrication process can be perfect and free of defects.

One real digital integrated systems can have a variety of defects. By testing them, a manufacturer can easily separate good and bad ICs. The IC quality is improved by testing since defective devices are not shipped to market. IC testing is a very expensive activity because an IC does not get any additional value.

Testing is an activity that presents the comparison of the fault free (ff) circuit response with the one obtained from the observed circuit, CUT (Circuit Under Test). There are two general concepts for testing approach: functional and structural testing. Verification that the circuit satisfies all required functions is referred to as functional testing. For combinational digital circuits this is a very uninteresting and time consuming process, because all possible combinations of input patterns must be applied to the circuit inputs in order to make sure that its function is correct. It is also very difficult to apply this to circuits with a large number of inputs.

The structural testing is on the otherhand, defect-oriented. Instead of checking if the circuit functions correctly, the test here searches for defects. The aim of such testing is to determine a test signal that will ensure that the responses of the ff circuit and the faulty one are different. The algorithm for test signal generation based on this approach is shown in Fig. 1.

```
Prepare the list of the defects
For each defect from the list
    {
    Select the next defect from the list of defects;
    Generate the test for the selected defect;
    For all other elements from the defects' list
        {
            Remove those (defects) that are covered
        with the generated test;
        }
    }
```

Fig. 1. The algorithm for test signal generation based on the structural testing concept

Generating a test signal that will cover every possible defect in the circuit is a very complex job, especially in an industrial environment. Thus, it is necessary to avoid having a list of all theoretically possible defects and create a list of defects that is both short and realistic.

It is impossible to perform structural testing at a high level of design abstraction. Thus the HDL description of the system must be loaded into the synthesis tool, then the synthesis must be performed, and after that the real netlist of the system with the actual gates and connections between them can be obtained.

This paper presents a VHDL-based approach for minimal test set generation for large digital combinational systems. This approach assumes that the synthesis of the system has already been performed and that the post synthesis netlist is available. Only in that way one can deal defects at the gate level of abstraction.

The paper is organized as follows. In the first section faults and defect issues are discussed. Some basic principles of digital systems testing are given in the second section. After that the steps in the digital system design are explained. In one of those steps, ff gates are replaced with faulty ones. Then the general approach of test pattern generation is given. The section after that, gives the principles of modeling faulty combinational gates. This approach is applied to two examples of large combinational arithmetical circuits which are parts of the power-meter IC. These examples and the obtained results are presented in the last section.

2. Faults in Digital Circuits

Physical causes of faults are called defects. Defects in most cases consist of missing or an extra material, or of an impurity. Such defects at the layout level of the chip are translated into electrical faults and then into logical faults, such that they can be tested with logical signals. A fault is a model of behavior due to the defect or it can be defined as an abstract model of the defect.

Faults can be single and structural. Single faults are related to fan-out issues, i.e. stems and branches. On the other hand, structural faults are related to interconnections and components. Interconnect faults are stuck-at faults, bridging (short) and open (break) faults. Component (transistor) faults are divided into stuck-open and stuck-short faults.

To make the test pattern generation easier, some assumptions about faults and physical defects must be made [3]. Mapping of defects into electrical, and thereafter into logical faults is called fault modeling. The principle of fault modeling is to reduce the number of effects to be tested by considering how defects manifest themselves. About 50% of faults that appear during tests in manufacturing are static faults. They are modeled with a single stuck-at fault model. According to this model a fault at one node is represented as either stuck at high level, that is 1, or low voltage level, that is 0.

This kind of modeling has many advantages of. First, it can represent many physical faults. It is independent of technology, as shown in Fig. 3. Multiple faults also appear in digital circuits. But their relative probability of appearing is much lower. Most test pattern generation is based on single stuck-at faults, because detecting single stuck-at faults also detects many other types of faults. This kind of modeling significantly reduces the test size to a reasonable value. For an n-net circuit it gives approximately 2n faults. This representation can also be used to model other digital circuit faults.

Examples of defects modeled by stuck-at faults for different technologies are shown in Fig. 3.

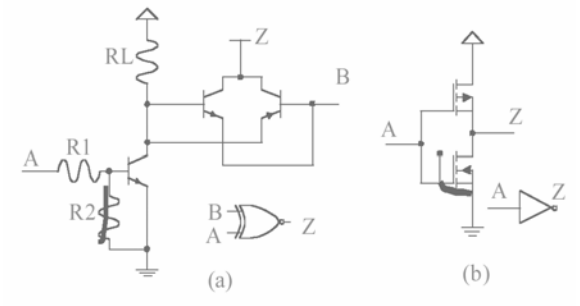

Fig. 3. Examples of static defects in logic gates; a) XOR BJT gate b) CMOS inverter

3. Digital Circuit Testing Principles

The main aim here is to generate a test for a selected defect. This is the most important and the most difficult issue in the algorithm shown in Fig. 1 [2]. One test can be used for detecting a certain defect, only if it can ensure controllability and observability.

Controllability is the ability of the test to force a state at the defect node different to the state caused by the defect. Observability, on the other hand, is the ability of the test to force the effect of the defect to at least one output of the circuit.

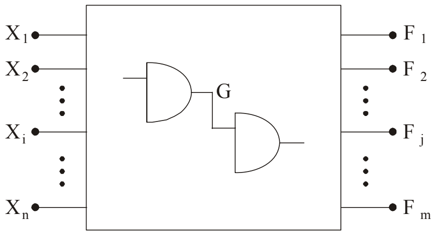

Fig. 4. The combinational circuit for the testing problem formulation

For the circuit shown in Fig. 4, n inputs are denoted with a vector **X**=[X_1, X_2, ... X_i, ... , X_n]; m outputs are denoted with a vector **F**=[F_1, F_2, ... F_j, ... , F_m]. Assume that it is necessary to create a test for the stuck-at fault at the node G, which is here denoted as G/s. The state at node G, can be expressed as the function of the input vector. The test must satisfy two requirements expressed by the following equations.

$$G(X) = \bar{s} \tag{1}$$

and,

$$F_j = (1, X) \oplus F_j(0, X) = 1\,. \tag{2}$$

These two equations present controllability and observability conditions. If at least one of these two conditions is not satisfied, the defect is not testable in this way, and it has to be detected using some other approach (for example I_{DDQ}).

4. Digital System Design Flow

It is almost impossible to generate a structural test sequence for the VHDL behavioral description of a digital circuit. The description itself does not contain any information about the logic gates that will implement the design after synthesis. Because of that, in order to get the final testable post-synthesis netlist of the circuit, it is necessary to go through all digital circuit design steps. For this purpose the Cadence system can be used [3].

The design flow of an ASIC begins with simulation of the RTL (Register Transfer Level) description of the design in VHDL in order to verify the circuit's functionality. This simulation can be performed using a standard VHDL simulator Active HDL. The next process is logic synthesis where a netlist of components and their connections is obtained. Initial and final timing analysis can be done after this step. The tool for logic synthesis takes a VHDL description of the design and appropriate technology libraries and generates a standard cell netlist. That netlist is imported into another tool to perform floorplanning, cell placement and routing. The obtained layout is verified at the end. Back annotation based on the extracted parasitics from the layout can be performed for more accurate timing analysis.

After these steps, a netlist of the circuit containing all actual library logic gates and their connections is available. For this netlist it is now reasonable to create a defect oriented test and to perform an estimation of the defect coverage.

In this way two arithmetic blocks' netlists of interest were extracted. They were: a 24-bit combinational subtraction unit and a 48-bit addition-subtracting circuit.

5. The Approach of Minimal Test Pattern Generation

In order to perform minimal test set generation (MTS) it is necessary to have a post-synthesis netlist of the circuit, and models of the faulty library elements used during the synthesis phase. One of the important steps in this process is to determine the fault coverage of the proposed test sequence.

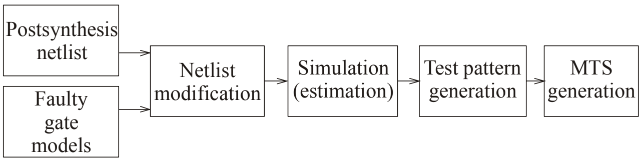

Fig. 5. The test pattern verification approach

The approach is shown in Fig. 5. First, the library logic gates used in the synthesis must be available in the VHDL netlist. Every logic gate must be modeled with all possible stuck-at defects. The modeling of the faulty gates will be explained later. The ff gates are then replaced in the netlist with the faulty ones. For a specified test sequences, the modified netlist is then simulated (using VHDL) for each of the defects specified in each logic gate. This simulation at the same time performs an estimation of the fault coverage. In this way we determine how many and which faults are left undetected with the proposed test sequence. Based on the results of the VHDL simulations, the special Matlab program performs MTS extraction. In this way it is possible to achieve 100% fault coverage.

5.1 Modeling Faulty Library Logic Gates

Modeling of faulty gates here assumes the stuck-at faults models of the gates and devices are incorporated into their VHDL descriptions [4, 5]. For testing the arithmetic circuits of interest, VHDL descriptions of faulty models for an inverter (inv), two input OR gate (OR2_fault), NXOR gate (EN_fault), and full adder (FA_fault), should be available. The 24-bit subtraction block and the 48-bit addition-subtraction circuit are shown in Fig. 6, while the modified library cells are shown in Fig. 7.

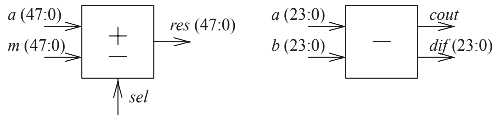

Fig. 6. Block diagrams of the observed combinational circuits: addition-subtraction and subtraction

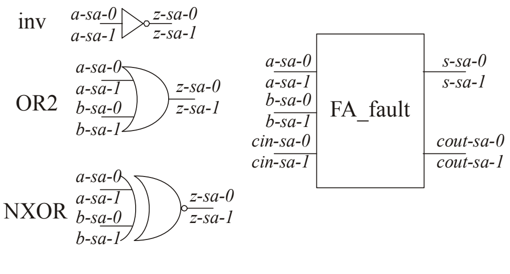

Fig. 7. Faulty logic gates

The model of each logic gate must contain the description of its behavior for every possible stuck-at fault. Stuck-at faults are related to each gates' input or output. Fig. 8. gives the VHDL description of the NXOR logic gate. It is based on that in [4, 5]. All other gates and circuits are similarly described.

6. Simulations and Results

Now instead of the ff components from the library, these faulty model components are instantiated in the modified netlist. After this modification, the resulting circuits are simulated according to the testbench description given in [4, 5]. In this VHDL testbench program, a file that contains the test pattern to be analyzed for the fault coverage, is specified.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.fault_inject.all;
entity EN_fault is port (
    z: out STD_LOGIC;
    a: in STD_LOGIC;
    b: in STD_LOGIC);
end EN_fault;
architecture inject_fault of EN_fault is
begin
nn: process(a,b) is
variable z_sa1, z_sa0, a_sa0, a_sa1, b_sa0, b_sa1 : fault_ptr:=null;
begin
if z_sa1=null then
z_sa1:=new fault_model'(new
string'(inject_fault'instance_name&"z_sa1"), false,false,first_fault);
first_fault:=z_sa1;
.
.
.
end if;
if z_sa1.simulating then      z<='1'after 1ns;
elsif z_sa0.simulating then   z<='0'after 1ns;
elsif a_sa1.simulating then   z<= b after 1ns;
elsif a_sa0.simulating then   z<=not b after 1ns;
elsif b_sa1.simulating then   z<= a after 1ns;
elsif b_sa0.simulating then   z<=not a after 1ns;
else                          z<=not (a xor b)after 1ns;
end if;
end process nn;
end architecture inject_fault;
```

Fig. 8. VHDL model of the faulty NXOR logic gate

After running this simulation a file with the correct results is obtained as well as a file with a list of defects covered by the test pattern. At the end of this report file, the exact number of stack-at faults covered by the proposed test sequence is given.

The obtained test set is now to be minimized since one test may detect more faults.

A	B	Diff
000000H	000000H	000000H
FFFFFFH	FFFFFFH	000000H
FFFFFFH	000000H	000000H
000000H	FFFFFFH	000000H

Tab. 1. Obtained subtraction results

A	M	sel	Diff
000000000000H	000000000000H	0	000000000000H
000000000000H	000000000000H	1	000000000000H
FFFFFFFFFFFFH	FFFFFFFFFFFFH	0	000000000000H
FFFFFFFFFFFFH	FFFFFFFFFFFFH	1	FFFFFFFFFFFEH

Tab. 2. Obtained adding-subtraction results

It should be mentioned that for many other different combinational circuits, whose tests have been verified in this way, the most covering test patterns (always covers at least 90% of all stuck-at faults) are all zeros and all ones.

The file that gives the correct result (results.txt) of the subtraction is shown in Table I. Part of the obtained covering report (file faults.txt) is given in Fig. 9. Similar results are obtained for the addition-subtraction circuit. The results and fault coverage for this circuit are shown in Table II and in Fig. 10. 100% fault coverage is achieved for both examples.

```
      :test(test):uut@oduzimacoff1_n24(netlist):
      fad12_23@fa_fault(inject_fault)cin_sa1
      Fault #1
      Detected          by          input:          000000000000000000000000
111111111111111111111111 outputs: 100000000000000000000001
      expected outputs: 000000000000000000000001 at    400 ns   900 ns
      .
      .
      .
      :test(test):uut@oduzimacoff1_n24(netlist):
      i_18450@inv(inject_fault)z_sa1
      Fault #287
      Detected          by          input:          000000000000000000000000
111111111111111111111111 outputs: 100000000000000000000001
      expected outputs: 000000000000000000000001 at    400 ns143900 ns
      Undetected:
      Fault cover:
      287faults,   287detected
```

Fig. 9. Partial verification report for the subtraction circuit

```
      :test(test):uut@oduzimacoff1_n24(netlist):
      fad12_23@fa_fault(inject_fault)cin_sa1
      Fault #1
      Detected          by          input:          000000000000000000000000
111111111111111111111111 outputs: 100000000000000000000001
      expected outputs: 000000000000000000000001 at    400 ns   900 ns
      .
      .
      .
      :test(test):uut@oduzimacoff1_n24(netlist):
      i_18450@inv(inject_fault)z_sa1
      Fault #287
      Detected          by          input:          000000000000000000000000
111111111111111111111111 outputs: 100000000000000000000001
      expected outputs: 000000000000000000000001 at    400 ns143900 ns
      Undetected:
      Fault cover:
      287faults,   287detected
```

Fig. 10. Partial verification report for the adder-subtraction circuit

These two files are now to be processed in one additional Matlab program. This program generates minimal test set that will cover all stuck-at defects in the circuit. One of the files obtained after this processing is shown in Fig. 11.

```
      Amount of faults: 287
INPUT: 000000000000000000000000 000000000000000000000000
FAULTS: 2 4 5 7 8 9 12 14 15 17 18 19 20 22 24 25 27 28 29 30 32 34 35
37 38 39 40 42 44 45 47 48 49 50 52 54 55 57 58 59 60 62 64 65 67
          .          .          .
INPUT: 000000000000000000000000 111111111111111111111111
FAULTS: 1 11 21 31 41 51 61 71 81 91 101 111 121 131 141 151 161
171 181 191 201 211 221 239
INPUT: 111111111111111111111111 000000000000000000000000
FAULTS: 10 235
```

```
INPUT: 111111111111111111111111 111111111111111111111111
FAULTS:  3 6 13 16 23 26 33 36 43 46 53 56 63
```

Fig. 11. MTS generation results

7. Conclusion

A method for minimal test pattern generation in combinational circuits is presented in this paper. It is based on VHDL simulations and is used and verified with examples of two arithmetic circuits of the integrated power-meter. Future work will extend the concept to sequential circuits.

References

[1] BURNS, M., ROBERTS, G. An introduction to mixed-signal IC test and measurement, Oxford University Press, New York, 2001.

[2] LITOVSKI, V. Electronic circuit design, in Serbian, Nova Jugoslavija-Vranje, Niš, 2000.

[3] DIMITRIJEVIĆ, M., JOVANOVIĆ, B., ANĐELKOVIĆ, B., SOKOLOVIĆ, M. Experiences in using Cadence – the industry standard for electronic circuit design, *Proc. of the XLVII Conference of ETRAN,* Herceg Novi, 2003, Vol. 1., pp. 31-34.

[4] ZWOLINSKI, M. Digital system design with VHDL, Prentice Hall, UK, 2004.

[5] ZWOLINSKI, M., SOKOLOVIĆ, M. Verification of digital system test patterns using a VHDL simulator, *Proceedings of the Small Systems Simulation Symposium 2005*, Niš, pp. 12-15.

About Authors...

Miljana SOKOLOVIĆ was born in Bor, Serbia and Montenegro, in 1977. She graduated at the faculty of Electronic Engineering, University of Niš, Serbia and Montenegro in 2001 as the best student. During 2001, she was working in Melexis gmbh, Germany on RF transceivers design. She joined LEDA (Laboratory for Electronic Design Automation) in Niš in 2001. and received M. S. degree in 2005 at the same faculty, where she now works as a teaching assistant and is a PhD student. Her main research are IC design, Design for testability and IC simulation and verification using VHDL. She received few awards for her scientific work.

Andy KUIPER completed his studies at FH-Wiesbaden (Germany) and was awarded his diploma in computer science in 2002. In 2003 he was involved in a project at Trinity College Dublin (Ireland). At present he is a PhD-student at Brno University with main interests on restoration of old films media.