A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robot
May 23-25, 2002, Cluj-Napoca, Romania

THE TESTING FACILITIES FOR THE TDL SOFTWARE ENVIRONMENT

Daniela E. Popescu, Corneliu Popescu, Marius Bonaciu

University of Oradea, Str. Armatei Romane nr.5, Oradea, România
depopescu@uoradea.ro

Abstract: The Testing Digital Language environment was developed by a team from the Computer Science Department of University of Oradea during the INFOSOC research program founded by the Research and Education Ministry. The goal of implementing such a software environment is the development of a new CAD tool with strong testing facilities, which can be used in the designing process for IC with high testability facilities.

Key words: CAD tool, logic simulation, fault simulation, test pattern generation

1. INTRODUCTION

In our days, the CAD tools used to design and analyze digital circuits use extensions that allow only the logical simulation of the implemented circuits [1]. The generated stimulate test vectors, are calculated without taking into account the conditions obtained through the minimization of their number. The programming environment that we are trying to design and implement – at The Computer Science Department from the University of Oradea - will be able to generate a minimal number of tests for the digital circuit analyzed and then to perform the calculations in order to evaluate de testability and reliability for the signal lines of the entire circuit. In this way it is possible to modify the structure level of the circuit in order to improve it, and if it's necessary BIST elements [7] may be included for self-testability.

The logical products analyzed with TDL will be changed into high testability projects, this being a substantial gain for digital circuits' producers because they'll be able to shorten the manufacturing cycle and the functionality checking.

The objectives of the research team are the following:

- Theoretical and practical thoroughgoing study of the methods used for digital circuits testing, testability evaluation and possible structural modifications in order to obtain the imposed testability
- Implementation of the programming environment TDL containing facilities for digital circuits testing that can be used in the design process or industrial process
- Dissemination of the results obtained in this project, through papers and presentations at national and international conferences
- Results obtained within the project will be included in the courses and seminars of the students attending the thoroughgoing study section.

2. TECHNICAL AND SCIENTIFICAL PRESENTATION OF THE PROJECT

The modules included in TDL (*Testing Description Language*) programming environment are implemented using C++, because it's easy to implement a user-friendly and consistent user interface.

Using TDL environment the user will be able to describe using TDL language the logical scheme for the digital circuit analyzed from testability point of view. Then he compiles (translates) the model obtaining the internal model of the digital circuit in a table form code.

The first modules used are the text editor module that describes the logical scheme and the compiler (translation) module used to obtain the internal model. In this stage the modelled circuit can be utilized in the TDL environment to the following operations and assessments:

- Logical simulation – SLC module (logical simulation through compilation [1])
  The logical simulation of the digital circuits is implemented immediately after the design phase because it is necessary to validate the lack of conception faults (errors) at logical project level.
- Generate the stimulate test vectors [2] [3]– GVST module
- Parallel simulation of single stuck at 0/1 faults – SDP module (faults parallel simulation).
  The SDP module implements the faults parallel simulation based on fault injection in any line of the analyzed circuit. For fault injection we associate for every circuit line two masks named – *mask(s)* and *fvalue(s).* This module is used to assess fault coverage.
- Deductive simulation of single stuck at 0/1 faults - SDD module
- Simulation of faults with delay – SDI module – conforming with the method presented in [6]
- Minimization of stimulus test vector sets using the minimal coverage method for stimulate test vectors - MVST module
  The module is used to minimize the stimulus test vector sets contained in an ASCII file, for a logical scheme code included in a specified file, the results are written in an output file. The contents of the output file is printed on the screen – so the user can see the number of initial tests, the minimized number of stimulus test vectors obtained after minimization.
- Assessment of the testability measures based on an algorithm used to solve this problem [4]– H-Assign module
  The calculations for the testability measures obtained using the H-Assign module is important to estimate the test patterns used for pseudo-randomized testing of the circuit corresponding to the logical scheme implemented.

In order to implement the TDL environment it is necessary to combine hardware and software methods. The programming environment contains an editor for TDL language, which has all the facilities of a text editor, a code generator (used to simplify the adding of new modules in the program code), a compiler used to translate the program code in an internal source, a truth table analyzer. This analyzer is used to verify input, intermediate and output signals in two distinct cases when the circuit is functioning without faults and when we inject single stuck at 0/1 faults. It also includes a logical simulator for time functioning analyses and another one for fault simulation.

It is very important to notice that translation of the logical schemes using TDL language offers different perspectives regarding the circuits, based on the level of abstraction specified.

The design and implementation of TDL software environment includes the software support capable of an optimal generation of stimulus test vectors, optimal simulation and testability assessment, and if it's necessary any modification at structure level in order to increase the digital circuit testability, conforming with [5].

3. USER INTERFACE

From the following figure (Figure1), we can notice that the user interface is divided in 3 zones:

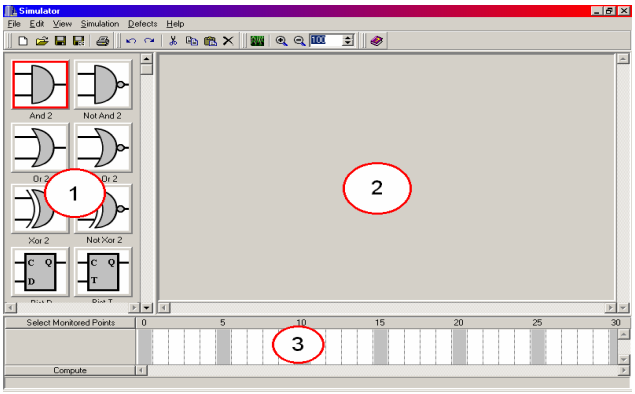

Figure 1

1) The zone from where we can select the components that will be introduced in the scheme
2) This is practically the zone where the scheme will be edited. Here we will introduce the components, make the connections using wires, etc
3) This is the zone used for displaying the results of the logical simulation of the current scheme.

For a better understanding and use of this program, and related to the preferences of the user, the size of any of these zones can be customized through a simple drag.

The selection of the components that we want to introduce in the scheme it's done from Zone 1.

For every component, the direction will be set simple by choosing the option "Direction". In figure 2, we can see an example through which we can choose to select the direction of the selected OR component. Also, we can see that from there we can choose the number of the inputs.

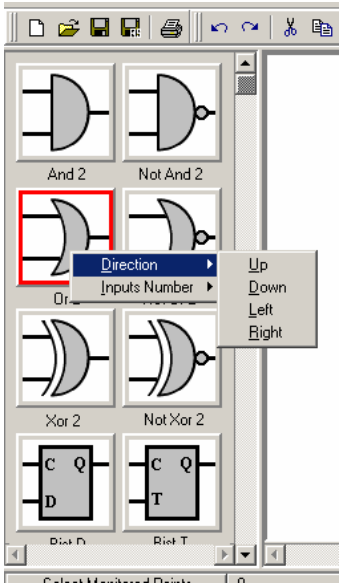

Figure 2

In case of an INPUT type of component, it is known that there are to types of components of this kind: static (the input value is constant all the time), or variable (the input value changes through the time, and usually these are considered as a CLOCK type). In this program, the type of an input component can be selected by a Right-Click on the component IN in zone 1(Figure 3).

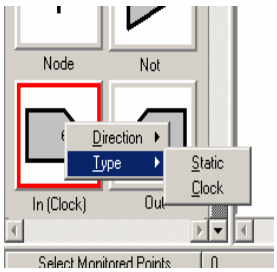

Figure 3

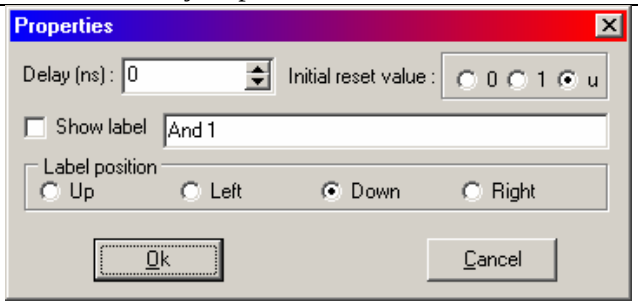

Figure 4.

The program supports up to 5.000.000 components (in future in might be improved).

Still, there are some restrictions related to how a wire can be connected to a component:

- We can't connect an exit to another exit, or an input to another input.
- We can't connect two pins from the same component
- We can't connect a pin that is already connected (linked to another wire)

To set the characteristics of a component, we will Right-Click on that component, resulting in opening a window used for modifying the parameters of that component. Of course, these will differ regarding the type of the component. In the next paragraphs, we will talk about these parameters.

The window from the Figure 4 will open for the following components: AND, nAND, OR, nOR, XOR, nXOR, BIST D, BIST T, BIST J-K, DELAY and NOT.

From here we can select the name of this component, we can set if we want to display it and the position related to the component (Ex. At the left of the component).

It is known that in reality, every component induce a delay caused by the times needed for the transistors to commute. By default, this is set on 0, so by default the functioning of that component is considered ideal. But, if it's necessary, it is possible to set a delay.

Also, we can set an initial re-set value. This can be known (0 or 1) or unknown (u).

In the window used for setting the characteristics for a CLOCK type IN gate, we can select the name of the components, just like at the previous window. But there will be needed to introduce the pulse duration value and the start value.

For a STATIC type IN gate it will only be needed to introduce the logical value of the inputs (which is constant all the time). For an OUT gate, the only option selected will be the name of the component.

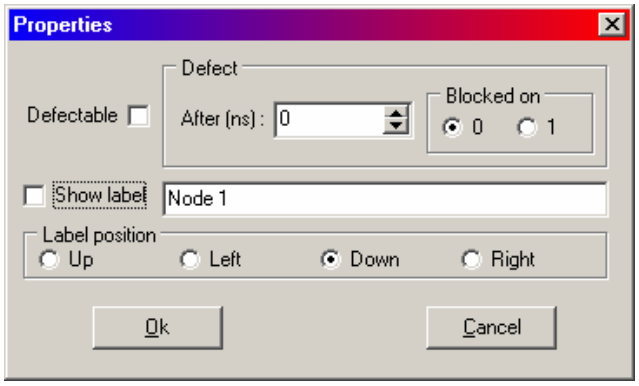

Figure 5

In case of a NODE (Figure 5), we will be able to select the name of the component, but, what's most important, even a possible defect. For a NODE it's possible to enter a defect after a specified time and also the value on which it will stacked.

If we want to simulate how the circuit works, it's preferable that we'll be able to visualize the values in time of the components. For this, it's necessary to select the

components that we will monitories. This can be done selecting the option *Simulation* > *Monitored Points*.

For our circuit, the window for selecting the components will be divided in two sections:

- In the left, there is the list of the components that can be introduced in the list of the components that are already selected for monitoring.
- In the right section, we can find the names of the components that are already selected.

After finishing this selection, we can see that in Zone 3 appeared the names of these selected components.

To be able to simulate the circuit, it's necessary to know the time for which the simulation will be done. By default, this is 10ns. But it can be changed, by selecting the options *Simulation* > *Properties.*

Usually, these dates should be sufficient to run a simulation. To begin generating a simulation, we will choose the option *Simulation* > *Start simulation* .The first step that the program will execute is checking the circuit for errors. If case that at least one will be found, it will announce the error and the component for which the error is, and it will halt the simulation process.

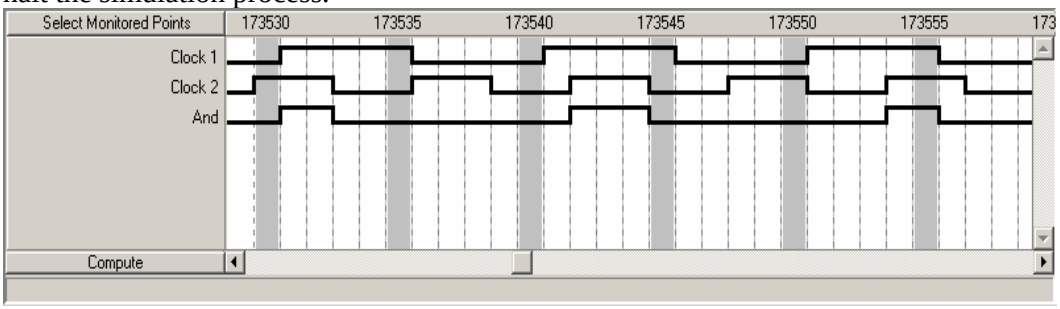

Figure 6

If everything it's in order, it will start to "compute" the simulation. After it finished it, the graphic will be displayed. In Figure 6 is given a simulation example for a simple scheme.

We can notice that at moment 173.535, the value for Clock 1 is 1, for Clock 2 is 0 and for And is 0.

In case we will set a 2ns delay for the And gate (mostly caused by the times needed for the transistors to commute), the results of the simulation will be as it is shown in Figure 7.

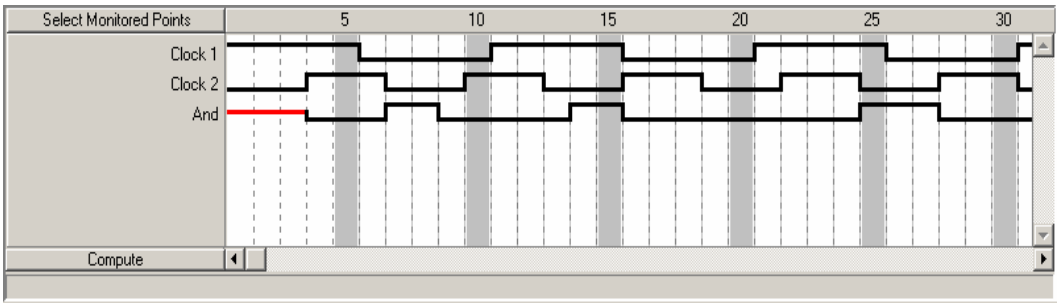

Figure 7

We can notice that the signal at the exit of the AND gate, is delayed with 2ns. The red line indicates an unknown value, and it's equal with the initial reset value.

If it's necessary, we can select the components that we want to be monitored, but this time the values will appear immediately in the simulation display zone, without any need to "recalculate" the simulation.

But, in case we will modify one of the attributes, delete a component, add one, modify one of the links or the time of the simulation, the simulation needs to be "recalculated".

CONCLUSIONS

The project belongs to a dynamic field with an intense research activity. So, the project could be used as a starting point for a new research activity if it will be required by the necessity of the project finally moment.

The project is in line with the strategic development plan of Oradea University, because:

- It propose a theoretical research with immediate applicability;
- It contributes to the improvement of the research basis from the university and to its integration to the world professional circuit;
- It habituates the students from the thoroughgoing study with the research activity
- It ensures a basis for cooperation with other universities from our country
- It stimulates the research team by facilitating its participation to national and international scientific manifestations; it also stimulates material the team in a symbolic fashion.

We intend to realize a long distance processing and portability for the TDL software environment; so, it will allow the users to access application and simulation modules.
The possible user (students or any other category interested in this domain), will have the possibility to access the TDL using local networks (Windows, Novell, UNIX) or through Internet (TCP-IP).

REFERENCES

1. Abramovici, M., Breuer, M.A., Friedman, A.D., (1996), Digital Systems And Testable Design, *Computer Science Press*
2. Dotan, Y., Arazi, B., (1990), Concurrent Logic Programming As A Hardware Description Tool, IEEE *Trans. on Comp., vol. 39, no. 1, January 1990*, pp. 72-88
3. Fujiwara, Hidao, (1990), Logic Testing And Design For Testability, *Computer Systems Series The MIT Press*, pp.30-75
4. Popescu D.E., Popescu C., (1994), An Algorithm for Solving the Generalize Probability Problem for Boolean Circuits, *Proceedings of FEI '25 Conference on Electronic Computers and Informatics*, Kosice Herl'any Slovakia, pp.126-132
5. Popescu C., Popescu, D.E., (2000) Some aspects about applying Boundary Scan Standard, *RSEEE 2000 – Oradea, Computer Science And Reability*, pp.87-93
6. Popescu, D.E., Popescu C., (1996), Nonrobust path delay fault simulation by parallel processing of patterns, *International Symposium on Systems Theory Robotics, Computers & Process Informatics, Section Computer Science & Engineering*
7. Popescu, C. Popescu D.E., (2000), A comparative study for using MISR as PRPG, *International Symposium on Systems Theory Robotics, Computers & Process Informatics, SINTES 10 Section Automation Computer Electronics*, pp.E49-E51