E-Learning Tools for Teaching Self-Test of Digital Electronics

A. Jutman[1], E. Gramatova[2], T. Pikula[2], R. Ubar[1]

[1]Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
[2]Institute of Informatics, Bratislava, Slovakia
Tel: +372 6202253, Fax: +372 6202253, Email: artur@pld.ttu.ee.

Abstract

Our paper describes a successful experience in combining separate educational tools developed at different universities into a single educational workflow aimed at teaching selected topics from the area of Self-Test of digital electronics.

1. Introduction

Today, testing is the major factor that assures the quality level of modern electronics. The expenses of testing commonly absorb up to 70% of all money allocated for the development and production of a new product. The rapid growth of integration levels of modern electronics makes the problem of testing more and more complex. The most noticeable trend in testing during the last several years is the move from traditional external test application paradigm towards embedded self-test solutions. Various Built-In Self-Test (BIST) techniques and related optimization problems became the most important research topics in the field of testing.

A basic BIST framework usually incorporates three main components: an embedded test pattern generator (TPG), response analyzer (RA) and the test access mechanism [1]. All the components are subjects to optimization driven as by the hardware cost (silicon area) as well as by the test set characteristics such as the fault coverage or test application time. The TPG is usually the part the main attention is focused on. The most common sources of test patterns in BIST are the pseudo-random pattern generator (PRPG) and a memory module as the source of deterministic pre-generated patterns. In general, the former one is much cheaper and easier to implement while the latter one provides much shorter test application time. The quality of the PRPG, which is usually represented by a Linear Feedback Shift Register (LFSR), can be improved a lot if the LFSR configuration was carefully selected and optimized [1].

The goal of this paper is to show how several educational tools developed recently in two universities: Tallinn University of Technology (TUT) and Institute of Informatics of Slovak Academy of Sciences (IISAS) can be connected together in order to form an e-learning environment for teaching important basic topics from the area of BIST. In the paper we describe these tools and the related laboratory work scenario.

The TUT tool set is called the Turbo Tester [2,7]. It is used in the first part of the scenario (see Figure 1) where the LFSR configuration is optimized in several ways based on the model of the target device under test (DUT). The students become acquainted to the process and learn how to estimate the quality of obtained solution.

The second phase of the work is the actual synthesis of the BIST hardware and

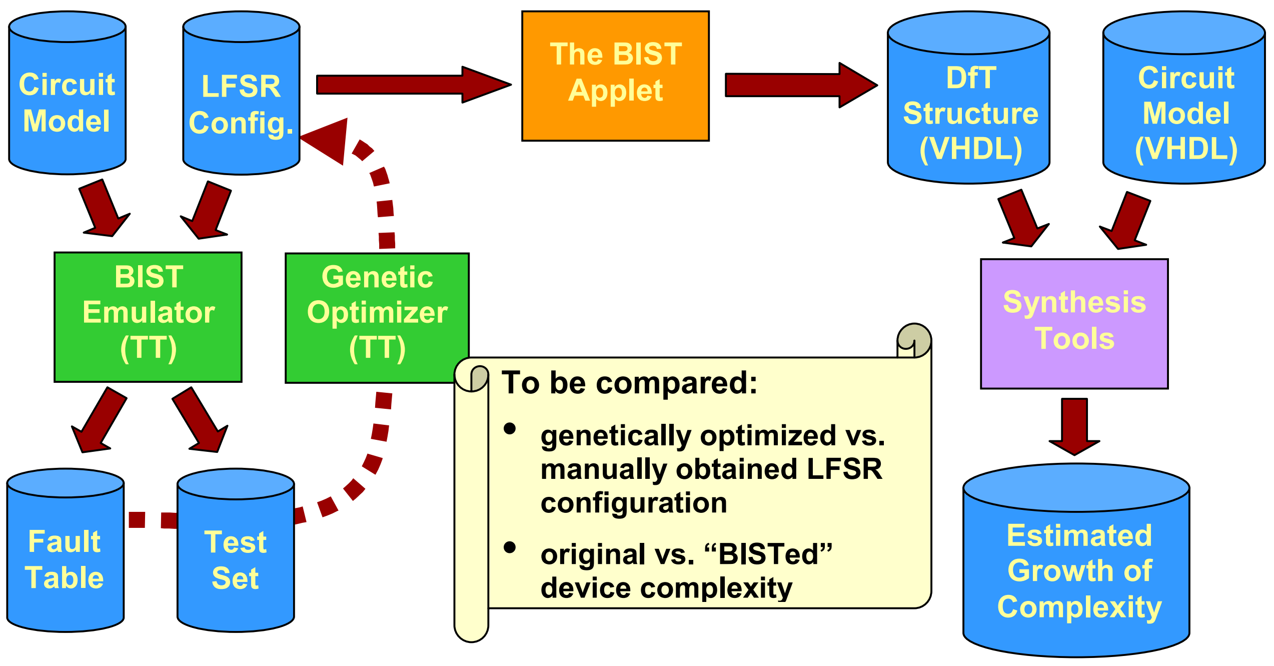


Figure 1. The workflow of the educational scenario

the test access mechanism, which is done by using the web-based Java applet [4,5,8] developed by IISAS. In fact, the applet can illustrate different BIST solutions, one of which will be used for the final implementation in the target device. The applet is designed accordingly to the "living pictures" concept [6] to ensure easy interaction with the system and the game-like style of learning.

Finally, the students should compare the hardware complexity (cost) of the initial device without BIST and the self-testable device to see the price, which must be paid for the quality product. A third party (commercial) logic synthesis tool can be used to synthesize the VHDL code of both variants of the DUT and to calculate the silicon area. In our particular implementation of the scenario we used the LeonardoSpectrum (Fig.2) software from Mentor Graphics [9]. Figure 2 demonstrates the statistics about the synthesized design in form of silicon area, maximum delay of combinational logic, number of inputs/outputs in the module under development.

Main goals of our scenario include:

a) teaching students to systematically combine different unconnected tools into a single instrument targeted at solving an engineering or a research problem;

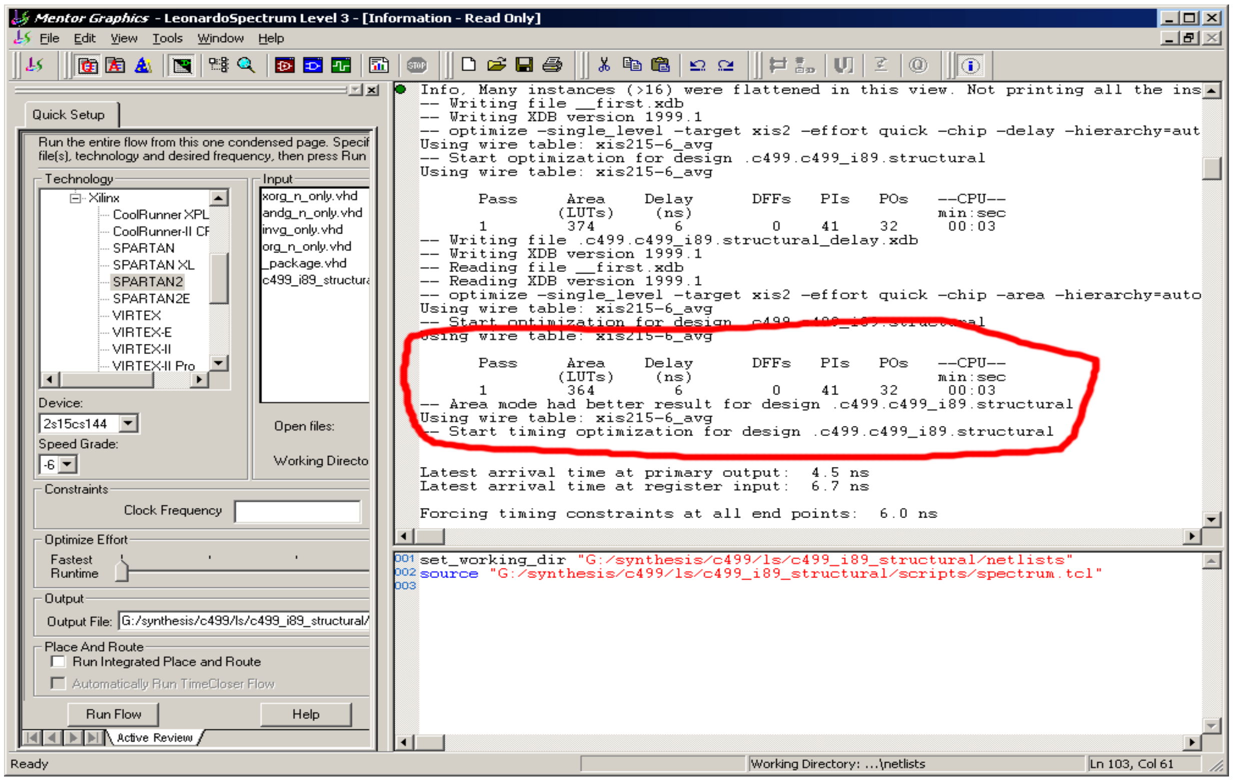
b) teaching selected topics from BIST.

In the following we describe the tools we combined in our workflow. Basic facts about Turbo Tester are given in the next section. Section 3 is devoted to the Java applet and Section 4 is the section for conclusions.

Proposed workflow has been successfully realized in the form of laboratory work for students during the spring semester 2004 in Jönköping University, Sweden.

2. The Turbo Tester tool set

There is a number of scientific papers describing research carried out using Turbo Tester that have been published in international conferences as well as reviewed journals [2]. This became possible due to a homogeneous environment of TT (Fig. 3) where different methods and algorithms for various test problems are implemented and can be investigated as separately of each other as working together in different combinations. The latter provides a variety of different optimization approaches for a particular problem, which is a useful feature for research-like practical exercises.


Mentor Graphics - LeonardoSpectrum Level 3 - [Information - Read Only]
File Edit View Tools Window Help
Quick Setup
Run Flow
Help
Active Review

Run the entire flow from this one condensed page. Specif
file(s), technology and desired frequency, then press Run

Technology: Xilinx — CoolRunner XPL, CoolRunner-II CF, SPARTAN, SPARTAN XL, SPARTAN2, SPARTAN2E, VIRTEX, VIRTEX-E, VIRTEX-II, VIRTEX-II Pro

Input: xorg_n_only.vhd, andg_n_only.vhd, invg_only.vhd, org_n_only.vhd, _package.vhd, c499_i89_structura

Device: 2s15cs144

Speed Grade: -6

Open files:

Working Directo

Constraints — Clock Frequency:

Optimize Effort — Fastest Runtime

Output — Output File: G:/synthesis/c499/ls/c499_i89_structural/

Place And Route — Run Integrated Place and Route; Automatically Run TimeCloser Flow

```
Info, Many instances (>16) were flattened in this view. Not printing all the ins
-- Writing file __first.xdb
-- Writing XDB version 1999.1
-- optimize -single_level -target xis2 -effort quick -chip -delay -hierarchy=aut
Using wire table: xis215-6_avg
-- Start optimization for design .c499.c499_i89.structural
Using wire table: xis215-6_avg

     Pass      Area     Delay       DFFs    PIs    POs    --CPU--
              (LUTs)     (ns)                              min:sec
     1         374        6           0      41     32     00:03
-- Writing file .c499.c499_i89.structural_delay.xdb
-- Writing XDB version 1999.1
-- Reading file __first.xdb
-- Reading XDB version 1999.1
-- optimize -single_level -target xis2 -effort quick -chip -area -hierarchy=auto
Using wire table: xis215-6_avg
-- Start optimization for design .c499.c499_i89.structural
Using wire table: xis215-6_avg

     Pass      Area     Delay       DFFs    PIs    POs    --CPU--
              (LUTs)     (ns)                              min:sec
     1         364        6           0      41     32     00:03
-- Area mode had better result for design .c499.c499_i89.structural
Using wire table: xis215-6_avg
-- Start timing optimization for design .c499.c499_i89.structural

Latest arrival time at primary output:  4.5 ns
Latest arrival time at register input:  6.7 ns

Forcing timing constraints at all end points:  6.0 ns
```

```
set_working_dir "G:/synthesis/c499/ls/c499_i89_structural/netlists"
source "G:/synthesis/c499/ls/c499_i89_structural/scripts/spectrum.tcl"
```

Ready   Working Directory: ...\netlists   Ln 103, Col 61

Figure 2. LeonardoSpectrum [9] synthesis tool from Mentor Graphics

2.1. Model synthesis

The component library of Turbo Tester consists of Binary Decision Diagram (BDD) representations for the library components of the circuits to be processed. The library is open and can be updated for new components. The model generator creates a BDD-representation of the design from the netlist of the design, produced by e.g. schematic editor. The special kind of BBDs is used in Turbo Tester. They are called Structurally Synthesized BDDs (SSBDD) and provide a uniform approach to solving a wide scale of testing tasks, based on a uniform model and a restricted set of standard procedures. For some tasks, such representation gives faster runtimes at the same accuracy [3]. A hierarchical DD model, which combines RT-level DDs and binary DDs is also possible. This allows migration of methods developed for logical level also to higher (behavioral and register-transfer) levels, where tools for hierarchical test generation and simulation have already been implemented.

2.2. Test generation

For automatic test pattern generation (ATPG), there are random, deterministic and genetic test pattern generators (TPG) implemented. Mixed TPG strategies based on different methods can also be investigated. Tests can be generated for both, combinational and sequential circuits. Stuck-at faults and physical defects can be considered. The best test generation efficiency for complex systems can be achieved by using the hierarchical DD representation.

2.3. Test pattern analysis

There are different fault analysis methods implemented in the system (e.g. single-fault simulation or parallel fault simulation). Thus, competing approaches can be investigated and compared for circuits of different complexities and structures. As the result of using these tools, fault tables are calculated and test quality is evaluated for given test sequences. In a defect-oriented

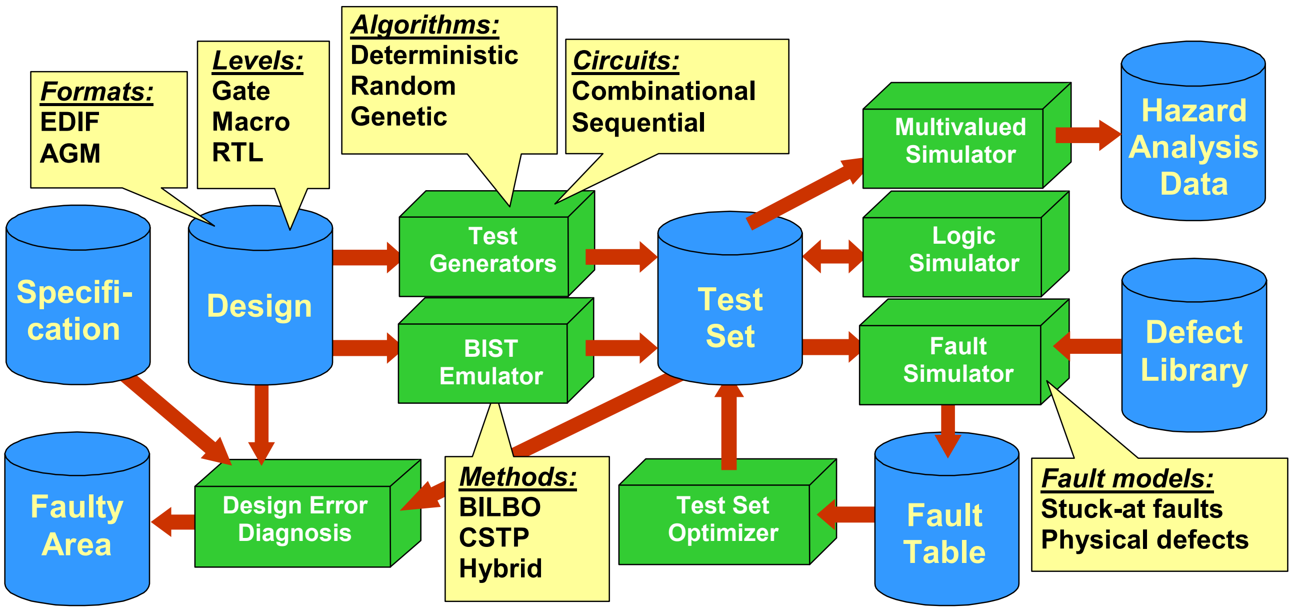


Figure 3. Overview of main Turbo Tester tools

simulation mode the fault simulator uses a special defect (bridging fault) library.

2.4. Test set optimization

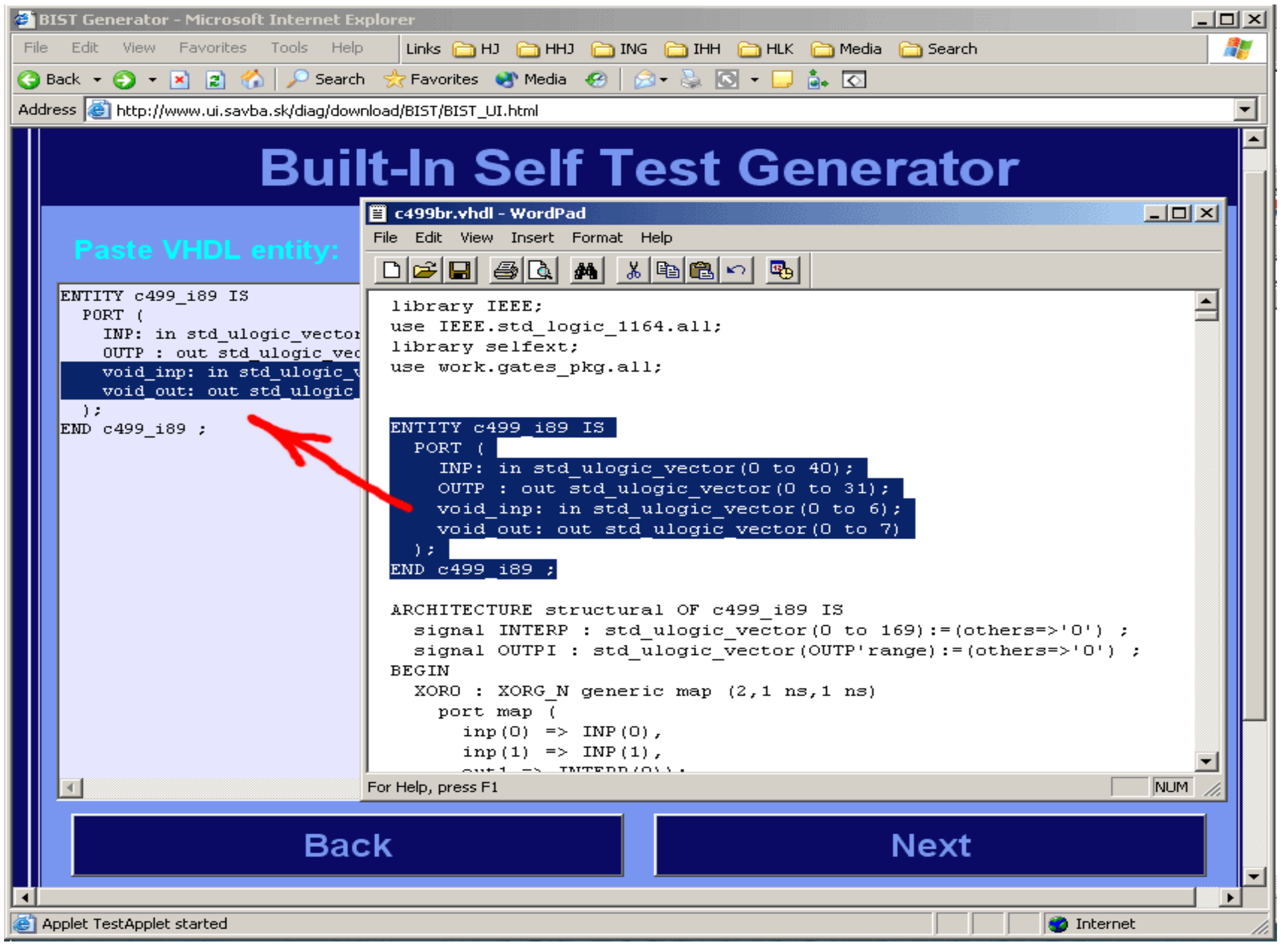
The tool minimizes the number of test patterns in the test set by means of static compaction. The technique implements effective representation of fault matrices by weighted bipartite graphs. The approach contains a preprocessing step for determining the set of essential vectors. Subsequently, implications and a greedy search algorithm are applied. This method offers significantly fast performance in terms of run times.

2.5. Testability analysis

The real cost of a digital product is expressed as: *Cost(Design + Test) < Cost(Design) + Cost(Test)*. It follows from the fact, that the total product cost can be minimized by regarding the design and test of a product as one integral activity rather than the two disjoint unrelated activities. The latter approach is called design for testability (DFT). Among the most promising DFT methods are those aimed at enhancing the testability through adding redundant hardware elements or test-points (additional outputs for observing; inputs for controlling; additional flip-flops in scan-path etc.) to the circuit. The testability analysis tools of the system can be used for finding out where to alter the design to improve the testability. This is done via enumerating untestable or statistically hard-to-test faults, and estimating the controllability, observability and testability characteristics for the nodes of the design.

2.6. Built-in self-test

Different BIST architectures can be simulated and the self-test quality of these architectures can be evaluated. There is a tool, which utilizes a genetic search algorithm for automatically finding good BIST architectures. It is also possible to study the general "store-and-generate" approach, where the whole test sequence will be generated on the basis of a given set of test vectors (i.e. the stored part of the test). All these vectors serve as initial input test patterns for on-line test generation (i.e. the generated part of the test). A Hybrid BIST technique represents an opposite approach, which also partially utilizes deterministic patterns but in the very end of the sequence. This makes it possible to achieve higher fault coverage by shorter test sequence.


BIST Generator - Microsoft Internet Explorer
http://www.ui.savba.sk/diag/download/BIST/BIST_UI.html
Built-In Self Test Generator
Paste VHDL entity:
c499br.vhdl - WordPad
For Help, press F1
Back
Next
Applet TestApplet started
Internet

```
ENTITY c499_i89 IS
  PORT (
    INP: in std_ulogic_vector
    OUTP : out std_ulogic_vec
    void_inp: in std_ulogic_v
    void_out: out std_ulogic
  );
END c499_i89 ;
```

```
library IEEE;
use IEEE.std_logic_1164.all;
library selfext;
use work.gates_pkg.all;


ENTITY c499_i89 IS
  PORT (
    INP: in std_ulogic_vector(0 to 40);
    OUTP : out std_ulogic_vector(0 to 31);
    void_inp: in std_ulogic_vector(0 to 6);
    void_out: out std_ulogic_vector(0 to 7)
  );
END c499_i89 ;

ARCHITECTURE structural OF c499_i89 IS
  signal INTERP : std_ulogic_vector(0 to 169):=(others=>'0') ;
  signal OUTPI : std_ulogic_vector(OUTP'range):=(others=>'0') ;
BEGIN
  XOR0 : XORG_N generic map (2,1 ns,1 ns)
    port map (
      inp(0) => INP(0),
      inp(1) => INP(1),
```

Figure 4. The Web-based BIST tool

3. The Web-based BIST tools

The conception of the Web-based tools is based on multiple learning modules – easily accessible on the Internet and running on mostly used standard browsers. Such Web technologies as Java applets, Java scripts, and Flash are the means for the development and implementation of these tools. The training modules simulate the learning subject in a well illustrative graphical form (in the manner of living pictures) that is self-explanatory, take advantage of learning by doing and involve interaction possibilities (easy action and reaction – click and watch). All the trained methods are explained in the visual and interactive manner in two languages – English and Slovak. Also a set of examples in the database and interactive help are always at disposal.

3.1. A BIST code generator module

This module generates the VHDL description of the BIST components based on their configuration. The components are to be attached to the circuit under test (CUT). The input to the BIST generator is only the VHDL entity of CUT. The user will be able to write or copy his own entity into this tool (Figure 4) or use any of the examples from the prepared database. Further, the user can determine the usage of primary input ports by excluding ports for which BIST TPG should not be used and by specifying clock and reset signals for the BIST control.

Then the user has to select one of the implemented pseudo-random or deterministic TPG techniques based on LFSR type I, LFSR type II, multiple polynomial LFSR, LFSR with reseeding, multiple polynomial LFSR with reseeding, CA or CA with bit-flipping [1,4]. LFSR based TPGs will be generated upon characteristic polynomials and reseeding values. CA based TPGs will be generated upon a deterministic test set using 3 state logic (0, 1 and don't care value).

The output signature compaction is performed by TCR or MISR (selection of the characteristic polynomial is the responsibility of the user). The user can also define a test length, except for CA or CA with bit flipping. In this case, the number of test patterns is defined automatically. Fault-free signature value has to be defined by the user for both compaction techniques. The VHDL description of all BIST components and the VHDL description of CUT with BIST on the top level are the output of this module. The generated VHDL descriptions can be simulated in commercial VHDL simulators and/or synthesized into the logic.

3.2. A BIST learning module

It is a demonstrational module. It will explain the basic BIST structure (Test Pattern Generator - TPG, multiplexer, controller and compactor). The detailed explanation of TPGs and compaction techniques will be the purpose of this module. The method explanation will be done by an interactive animation of the selected and by the user determined TPG (LFSR type I, LFSR type II, multiple polynomial LFSR, LFSR with reseeding, multiple polynomial LFSR with reseeding, CA or CA with bit-flipping) or compactor (MISR, TCR). This module is under development yet.

4. Conclusions

Our paper describes a successful experience in combining separate educational tools developed recently in two different universities: Tallinn University of Technology and Institute of Informatics of Slovak Academy of Sciences.

Based on these tools, we developed an educational workflow aimed at teaching selected topics from the area of self-test of digital electronics.

The resulting environment provides good possibilities for laboratory training and experimental research for students.

5. Acknowledgments

This work was supported in part by the EU Framework V projects REASON and eVikings, and by the Estonian Science Foundation Grants No. 5649 and No. 5910.

References

[1] M.L. Bushnell, V.D. Agrawal, *Essentials of Electronic Testing for Digital Memory and Mixed-Signal Circuits*, Kluwer Academic Publishers, Dordrecht: 2000, p. 690.

[2] M.Aarna, E.Ivask, A.Jutman, E.Orasson, J.Raik, R.Ubar, V.Vislogubov, H.-D.Wuttke, "Turbo Tester - Diagnostic Package for Research and Training", in *Scientific-Technical Journal "Radioelectronics & Informatics".* KNURE. Vol. 3(24), 2003, pp.69-73.

[3] A. Jutman, J. Raik, R. Ubar, "SSBDDs: Advantageous Model and Efficient Algorithms for Digital Circuit Modeling, Simulation & Test," in *Proc. of 5th International Workshop on Boolean Problems*, Freiberg, Germany, Sept. 19-20, 2002, pp. 157-166.

[4] T. Pikula, E. Gramatová, "BIST Architecture Application for Digital Circuits as a Java Applet", In *Proc. of the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Poznan, , 2003, pp. 305-306.

[5] T. Pikula, E. Gramatová, M. Fischerová: "Automatic Design of Cellular Automata for Generating Deterministic Test Patterns," in *Digest the 8th IEEE European Test Workshop*, Maastricht, The Netherlands, 25-28 May 2003, pp. 69-70.

[6] H.-D. Wuttke, K. Henke, "Teaching Digital Design with Tool-Oriented Learning Modules "Living Pictures", in *Proc. of 32nd ASEE/IEEE Frontiers in Education Conference*, Boston, USA, Nov. 6 - 9, 2002, Session S4G, pp. 25-30.

[7] Turbo Tester home page URL: http://www.pld.ttu.ee

[8] Web-based BIST tool home page URL: http://ups.savba.sk/diag/download/BIST/BIST_UI.html

[9] Home page of LeonardoSpectrum synthesis tool: http://www.mentor.com/leonardospectrum/