WEB-BASED TRAINING SYSTEM FOR TEACHING BASICS OF RT-LEVEL DIGITAL DESIGN, TEST, AND DESIGN FOR TEST

S. DEVADZE, A. JUTMAN, A. SUDNITSON, R. UBAR
TALLINN TECHNICAL UNIVERSITY, ESTONIA

KEYWORDS: Web-Based Teaching, Java Applet, Education Methodology

ABSTRACT: A conception of training system for teaching design and test of digital devices is presented. The system is designed mainly to illustrate RT-level (Register Transfer Level) problems in control intensive digital systems. Such topics as investigation of tradeoffs between speed and HW cost in digital design, RT-level simulation, fault simulation, test generation, built-in self-test (BIST) and other similar are covered by the training system. The system is implemented in a form of Java applet and can be freely accessed over Internet. The latter makes it easy for students even from foreign universities to use this system any time and in any place. The Java applet has built-in multilingual support to ensure easy integration into teaching courses of universities over the world.

INTRODUCTION

Rapid advances in areas of deep-submicron electron technology and design automation tools enabled engineers to design larger, more complex, integrated circuits. Until recently, most electronic systems consisted of one or multiple printed circuit boards, containing multiple integrated circuits (IC) each. Recent advances in IC design methods and technologies allow to integrate these complex systems onto one single IC. These developments are driving engineers toward new System on a Chip (SOC) design methodologies. SOC is seen as a major new technology and the future direction for the semiconductor industry.

On the other hand, the more complex are getting electronic systems the more important become problems of test and design for testability, as costs of verification and testing are getting the major component of design and manufacturing costs of a new product. Today, design and test are no longer separate issues. The emphasis on the quality of shipped products, coupled with the growing complexity of system design, require testing issues to be considered early in a design process. At present, most VLSI and system designers know little about testing. The companies frequently hire test experts to advise their designers on test problems, and they even pay a higher salary to the test experts than to their VLSI designers. This reflects the today's university education: everyone learns about design, but only truly dedicated students learn test [1-3].

Entering the SoC era with its new concepts means teaching more material on high-level and system design together with the same amount of essential basics, that must not be forgotten. In its turn, this means that more information volumes must be fitted into the same time frames. Therefore, new effective teaching concepts must be introduced to avoid poorer coverage of essential topics.

Teaching the basics of digital design and test means teaching a lot of complex connections. Those connected topics have to be explained at first one by one. Then come their dynamic interactions. Traditional teaching methods using desk, overhead or "PowerPoint™" presentations can explain those connections only partially.

After the lecture the dynamic part of the lecture, the connections created by the teacher between different subjects get lost and the students only have a static part of the whole scenario in their notes. After listening to a lecture they can consult only their synopses and try to solve some problems by using a newly learned method as good as they remember. The only accompanying materials students use in most cases are books, scripts etc.

An enhancement of this situation can be reached, if the whole material, containing all scenes of the dynamic process can be accessed via the Internet. The same dynamic content can be used then as during the lecture itself as later by students at home.

In the following a conception and tools are presented to increase the teaching quality in the field of electronics design and test. To illustrate both design and test problems together, we use the same system consisting of several interactive modules. The system supports the possibility of distance learning as well as a web-based computer-aided teaching. The interactive modules are focused on easy action and reaction, learning by doing, a game-like use, and encourage students for critical thinking, problem solving, and creativity.

THE CONCEPT

The teaching system is implemented as a Java applet, which consists of the following parts:

- System model subpanel (Fig. 1)
- Data path

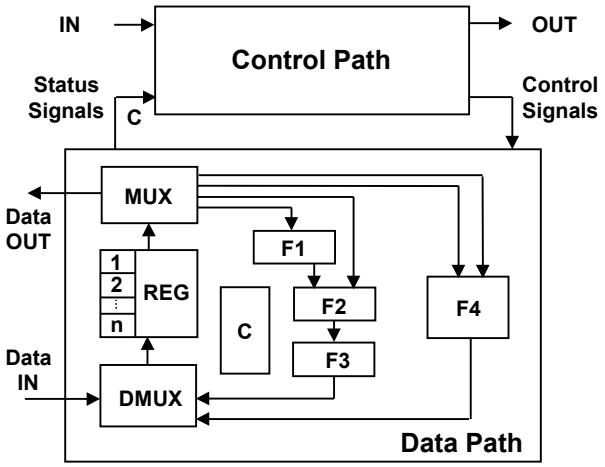

Fig. 1 *System model*

- Control path
- Simulation module
- Fault simulation module
- Test generation module
- BIST module

The conception we describe in the following allows to solve and illustrate many problems related to RT-level control intensive digital design together with test. The range of problems includes (but is not limited to):

- Design of data path and a microprogram (control path) on RT level
- Investigation of tradeoffs between speed & HW cost in the system
- RT level simulation
- Fault simulation
- Test generation
- Design for testability and BIST

a) Data Path Description

Each functional unit F1..F4, MUX, and DMUX has a list of micro-operations unary or binary. It is supported by an RT-level and gate-level models of these microoperations. The RT-level model is needed for high-level simulation by JAVA library subprograms, while the gate-level model is used for gate-level logic and fault simulation. All the microoperations are labeled by a control signal which activates the microoperation. The description of the data path functionality in format "control signal: microoperation" is presented in Fig. 2.

While designing his device (implementing a given algorithm or a function like multiplication, division etc.) a student can select needed microoperations for each unit of data path from the whole set of possible pre-designed microoperations. Each microoperation has a gate-level implementation, and the number of gates determines the cost of the microoperation. By selecting a set of microoperations for the whole data path the student will get also the cost of the data path in the number of gates.

Different architectures can be chosen for implementation of a given function. Students can compare them and find the tradeoffs:

- *M-automaton* (F1 and F3 are selected to be transparent, F4 is disabled, all microoperations are selected only from F2). In this case it is possible to carry out a single microoperation in one clock cycle, the speed is low, but the cost of HW is saved.
- *Sequential IM-automaton* (F1, F2, F3 are enabled, F4 is disabled). In this case the system can carry out maximum 3 microoperations during a single clock cycle if the algorithm gives such a possibility. For example, in division F1 is used for inversion to allow subtraction by the adder in F2, and F3 can be used for shifting of data.
- *Parallel IM-automaton* (all blocks are selected). In this case the system can carry out maximum 4 microoperations during a single clock cycle if the algorithm allows to. For instance, while the three functional units described above perform division, F4 can count clock cycles.

For every chosen architecture, the system calculates the cost of HW. The speed (the number of clock cycles the microprogram needs) can be measured by simulation.

b) Control path

The control path is a microprogrammed controller [4], which implements Mealy FSM (Final State Machine). The controller consists of a microprogram table and an interpreter. The microprogram is developed by the user

MUX	m_{ij}: $B_j = R_i$	i = 1,…,n – Register number; j = 0,1,2,4 – Bus number where B_0 is Data OUT Bus, B_1 is the Bus to F1 etc.
DMUX	d_{ij}: $R_j = B_i$	i = 0,3,4 – Bus number where B_0 is Data IN Bus, B_3 is the Bus from F3 etc. j = 1,…,n – Register number
F1 (F3)	f_{1j} (f_{3j})	unary microoperations like: various shiftings, inverting, counting (+1, -1) etc.
F2	f_{2j}	various binary microoperations (with 2 operands)
F4	f_{4j}	various unary and binary microoperations (with 2 operands); there is a overlay between functions of F4 and the ones of F1, F2 and F3 to allow a parallelization of the given algorithm
C	c_i	a list of Boolean conditions

Fig. 2 *Description of data path functionality*

TABLE 1 The microprogram table

Addr	Next	F1(in)	F2(in)	F4(in1)	F4(in2)	IN	F1	F2	F3	F4	F3(out)	F4(out)	OUT	Input	C1	C2	C3	…
1	2					REG2								A	X	X	X	X
2	3					REG3								B	X	X	X	X
3	4	REG3		REG1	REG2		SHR0			ADDER	REG3	REG1			0	0	X	X
3	3	REG3		REG2			SHR0			SHL0	REG3	REG2			1	0	X	X
3	END												REG1		X	1	X	X
4	3			REG2						SHL0		REG2			X	X	X	X

to realize a given algorithm based on the available (selected in prior) resources of the data path. The user fills in the microprogram table represented as a subpanel of the applet.

The first two columns of the microprogram table (Table 1) represent the address of current microinstruction and the address of the next microinstruction correspondingly. The current microinstruction can be split into several rows in case if its operation depends on the set of conditions C. Then the only proper row will be selected. Columns 3 to 6 correspond to MUX and indicate which register (REG1…REGn) will be multiplexed into which functional unit (F1, F2, F4). Registers where the input data from Data IN (column "Input") will be written are specified in column "IN". The input data are the operands of the implemented algorithm (this will be further discussed in the next subsection).

Columns F1 to F4 stand for a certain microoperation selected for a corresponding functional unit (F1 to F4) in a certain clock cycle. The DMUX section is specified in next two columns. It shows to which register the data from functional units F3 and F4 will be written. Column "OUT" indicates the register which will be redirected to Data OUT. The last columns (C1, C2, …) stand for conditions where the following values must be specified: 0, 1, X (don't care).

In Table 1 an example of algorithm, which performs multiplication of two operands A and B is presented. The result of multiplication is stored in REG1 and fed to the data output. The algorithm is as follows:

$R_2 = A$
$R_3 = B$
While $R_3 \neq 0$
{
  If $(R_3[0]) = 1$
    $R_1 = R_1 + R_2$
  $R_3 = R1(0.R_3)$ // Shift R_3 right by one bit, put 0 at end
  $R_2 = L1(R_2.0)$ // Shift R_2 left by one bit, put 0 at end
}
Data OUT = R_1

c) Simulation

Simulation is carried out at the higher level by using Java subroutines (corresponding to functional units) which are activated by the control signals in the order given in the microprogram table (by checking the condition values). The overall algorithm is the following:

Reset all the *Registers*
Current State = 1
While *Current State* ≠ *END*
{
  Read *Status Signals*
  If (*Current State* = *ADDR*) **and** (*Status Signals* ⊂ *C*)
    Select the *Row* **from microprogram table**
  Else
    Error: *"Simulation Error"*
  Assign function to each enabled *Functional Unit*
  R_{IN} = *Data IN*
  $R_{F3(out)} = F3(F2(F1(R_{F1(in)}), R_{F2(in)}))$
  $R_{F4(out)} = F4(R_{F4(in1)}, R_{F4(in2)})$
  Data OUT = R_{OUT}
  Current State = *NEXT*
}

Simulation can be carried out also at the gate level by using Structural BDDs. The process is also controlled by the data in microprogram table. The simulation data is stored in a subpanel Simulation Results (Table 2).

Column "Test Nr." defines the number of data group from the Test Data Table (described in the next subsection). The clock number is specified in the next column. The simulation data is written in all the other cells of the Simulation Data table at each clock cycle. This data reflects the states of all the registers, outputs of all the functional blocks, data input and output of the device, current states at each clock cycle and condition signals. The simulation data can be used later by the student as the debugging info.

Column "Test Nr." can be neglected during conventional simulation. In that case, we simulate only a

TABLE 2 The simulation data

Test Nr.	Clk. Nr.	R1	R2	R3	R4	R5	R6	R7	F1	F2	F3	F4	IN	OUT	State	C1	C2	C3	C4	C5	C6	C7	C8
1	1	0	3	0	0	0	0	0	0	0	0	0	3	0	2	1	1	1	0	1	0	1	0
1	2	0	3	5	0	0	0	0	0	0	0	0	5	0	3	1	1	1	0	0	0	1	0
1	3	3	3	2	0	0	0	0	2	2	2	3	5	0	4	0	0	1	0	1	0	1	0
1	4	3	6	2	0	0	0	0	0	0	0	6	5	0	3	1	0	1	0	0	0	1	0
1	5	3	12	1	0	0	0	0	1	1	1	12	5	0	3	1	0	1	0	1	0	1	0
1	6	15	12	0	0	0	0	0	0	0	0	15	5	0	4	0	0	1	0	0	0	1	0
1	7	15	8	0	0	0	0	0	0	0	0	8	5	0	3	1	1	1	0	1	0	1	0
1	8	15	8	0	0	0	0	0	0	0	0	0	5	15	0	1	1	1	0	1	0	1	0

single microprogram (formally, with Test Nr. 1). For investigation of test problems (like test generation, fault simulation, etc.) we use the same microprogram repeatedly for several input data. Then each repetition of the microprogram will be regarded as a test with a corresponding number.

Table 2 represents the simulation results corresponding to the example of the multiplication algorithm given in the previous subsection when input operands are numbers 3 and 5.

d) Test generation

A microoperation of one of the units F1, .., F4 can be chosen as a target for testing. It is assumed that the same microprogram will be repeated for a set of operands for the chosen microoperation. The operands must be written into the Test Data Table:

TABLE 3 The test data

No	DA = 1	DA = 2	DA = 3	DA = 4
1	Data_11	Data_21	Data_31	Data_41
…	…	…	…	…
n	Data_1n	Data_2n	Data_3n	Data_4n

Data_1i and *Data_2i* – are the first and the second operand used for the microoperation under test. The microoperation is tested n times by n different sets of 2 (or 1) operands.

For test generation no special automatic means will be provided. Either manually generated functional patterns or randomly chosen patterns (test data) can be used. The efficiency (quality) of these tests can be estimated by fault simulation.

e) Fault simulation

Fault simulation is carried out at the gate level by using Structural BDD model. Faults for the given block are inserted into BDDs. The simulation process is controlled by the data in microprogram table. The target of the fault simulation (a unit, and a microoperation in the unit) are selected by a student and then highlighted. The fault simulation data is reported in the Fault Coverage Table:

TABLE 4 The fault coverage

Nr.	F1		F2		F3		F4	
	f_{1k}		f_{2k}		f_{3k}		f_{4k}	
	C	TC	C	TC	C	TC	C	TC
0								
…	C_i	TC_i	C_i	TC_i	C_i	TC_i	C_i	TC_i
n								

C_i and TC_i – are fault coverage for the current test (microprogram) with data *Data_1i* and *Data_2i* and the total fault coverage for all the tests with all the data in the Data Table up to the current group of *i*. The microoperations f_{ik} are selected from each functional unit as a target for fault simulation.

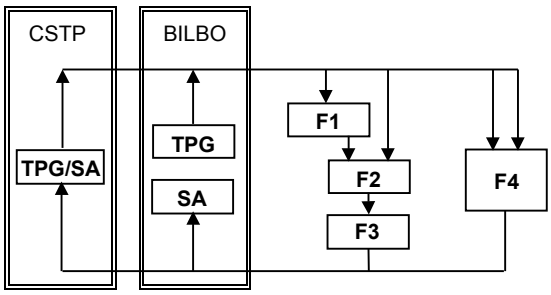

Fig. 3 *Scan-path design*

f) Built-in Self-Test

Usually functional test patterns do not provide a good fault coverage. Therefore scan-path with random test pattern generator is introduced. Special BIST subpanel with BIST results subpanel can be opened for this operation mode. By Scan-Path technology the inputs and the outputs of the combinational blocks in data path are directly accessible by scan-path registers TPG (random test pattern generator), SA (signature analyzer), and TPG/SA (combined TPG and SA) [5]. See Fig. 3.

Two modes may be implemented: BILBO (Built-In Logic Block Observer) mode based on using TPG and SA, or CSTP (Circular Self-Test Path) mode based on using combined TPG/SA scan-path register.

Both modes can be implemented in two ways: different settings for each combinational circuit to be tested, or the same setting for all circuits. The aim of a student's work is to find best settings. For setting the polynomial, the initial state of the TPG, and the number of clocks to be used for test generation, there is a special subpanel.

Again, the targets for testing are microoperations in blocks F1, … F4. Random test patterns generated by the TPG are saved, then fault simulated, and finally the fault coverage is displayed.

THE IMPLEMENTATION

The teaching system is implemented in a form of Java applet. It can be accessed through URL [6]. The Java 1.1 is chosen as the platform for the applet because at the moment it is a de-facto standard and it is included in installation packages of most popular browsers.

Applet interface (Fig. 3) consists of following parts:

- *Schematic View* - panel which contain schematic representation of a design. This panel allows user to define or change some properties of data path of the design. Some components can be enabled/disabled or their functionality can be changed. In the step-by-step simulation mode the results of the simulation are visually demonstrated on the *Schematic View* after each step of the simulation.
- *Microprogram tab-panel* is used to define control path of the system. In the simulation mode this panel shows which part of the microprogram is currently executed.

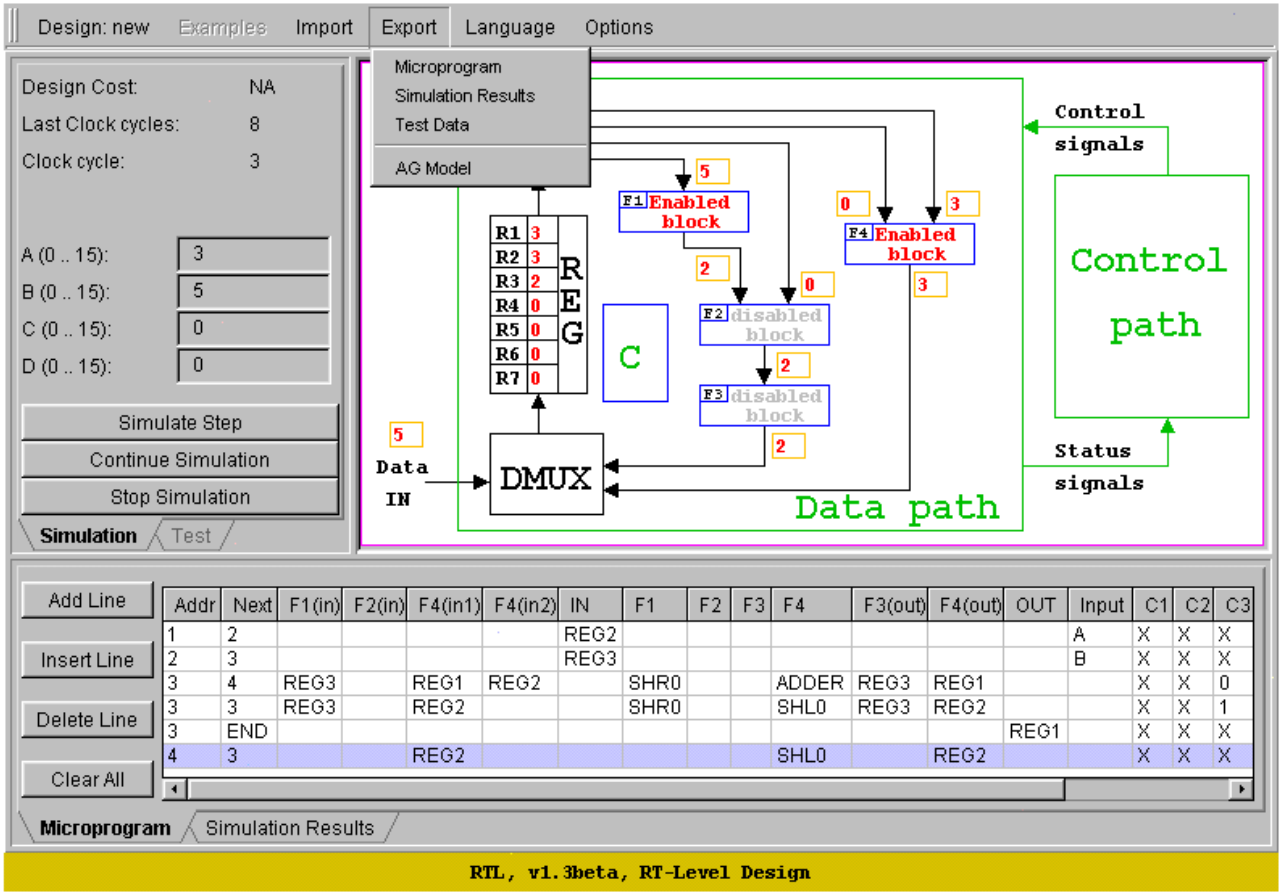

Fig. 3 *Applet window*

- *Simulation tab-panel* and *Test tab-panel* are used to simulate and test design correspondingly. The simulation can be carried out in different ways:
  1) *Step-by-step simulation mode*. In this case each row of the microprogram is executed separately and all the results of this execution (including state of registers and functional blocks, inputs and outputs, status signals, etc) can be viewed directly on *Schematics View* sub-panel. This mode of simulation is useful for illustrating how the design works or for debugging.
  2) *Test mode*. This mode is used to test the design repeatedly with some set of input data. User fills in the *Test data table* in the *Test tab-panel* with the data that will be used in testing. Then the design simulation can be executed for a particular row of test table or for all the rows at once.

  The results of the simulation or test are placed to *Simulation Results tab-panel*.

The user-friendly interface of the applet allows users to start experimenting with the system immediately without requiring a long training. Nearly all operations with the applet (except the specification of input data values) are performed using just the mouse. The help system for the applet is currently under construction. The draft version of this user guide can be accessed through URL [7].

The applet has a flexible design. Should any other RT-level model be used instead of the described one, it must be only specified in a form of text-files. Then it can be loaded just as simple as the original one. The Microprogram table and the Simulation Results table will be automatically reconfigured as well.

The current version of the applet is equipped with several such design templates. New templates can also be added or existing templates can be changed without changing the applet.

Applet has a built-in collection of examples (they also can be extended or modified) that implement different algorithms. They will help users to understand principles of functioning of the system. For connecting the system to other applications as well as providing users with possibility to save the results of their work for further use applet has a capability of exporting results and importing source data. Applet has a built-in multilingual support: its interface can be easily translated to other languages if needed (current language is English).

At the moment the applet is still in its beta version stage. However the most functionality is already implemented. The fault simulation and BIST are the only missed interactive modules planned for future work.

CONCLUSIONS

By the use of web-based media we achieve: presentation of course material independent of place and time, individual learning according to the students‘ own

needs, new forms of communication between teachers and students, up-to-date course material.

The conception presented allows to improve the skills of students to be educated for digital hardware and SOC design connected with test related topics. The system fully reflects the "easy action and reaction" conception which was taken as the major target for its creation. Each field, each functional unit, other modules are click-able and their function can be changed or further adjust-ed. The reaction on this action will be instantly reflected by highlighting and changed colors of selected modules.

On the other hand the tasks chosen for training represent simultaneously real research problems. This provides students with dynamic environment to experiment with and to find interesting solutions for stated problems. The main target of described system is to foster in students critical thinking, problem solving skills and creativity in a real research environment and atmosphere.

The free-access over Internet basis and self-contained nature makes it easy for students even from foreign universities to use this system any time and in any place. The Java applet has built-in multilingual support to ensure easy integration into teaching courses of universities over the world.

THE AUTHORS

S. Devadze, A. Jutman, A. Sudnitson, R. Ubar are with the Computer Engineering Department of Tallinn Technical University, Raja 15, 12618 Tallinn, ESTONIA. E-mail: artur@pld.ttu.ee

Acknowledgements – This work is supported by the Thuringien Ministry of Science, Research and Art (Germany), by EU V Framework project REASON, and by the Estonian Science Foundation (grants No 3658, 4876 and 4300).

REFERENCES

[1] M.L. Bushnell. Increasing Test Coverage in a VLSI Design Course. International Test Conference, Atlantic City, NJ, USA, 1999, p. 1133.

[2] J. Harrington. VLSI Design 101 – the Test Module. International Test Conference, Atlantic City, NJ, USA, 1999, p. 1134.

[3] V.D. Agrawal. Increasing Test Coverage in a VLSI Design Course. International Test Conference, Atlantic City, NJ, USA, 1999, p. 1131.

[4] Armstrong J. R., Gray F. G. Structured logic design with VHDL. *Prentice-Hall*, Englewood Cliffs, 1993, 482 p.

[5] Abramovici M., Breuer M.A., and Friedman A.D. Digital systems testing and testable design. *IEEE Press,* New York, 1999, 652 p.

[6] Teaching system URL: http://www.pld.ttu.ee/dildis/automata/applets/9/

[7] Help URL: http://www.pld.ttu.ee/dildis/automata/applets/9/Help/howto.html