A Web-Based Visualization and Animation Platform for Digital Logic Design

Abdulhadi Shoufan, Zheng Lu, and Sorin A. Huss

Abstract—This paper presents a web-based education platform for the visualization and animation of the digital logic design process. This includes the design of combinatorial circuits using logic gates, multiplexers, decoders, and look-up-tables as well as the design of finite state machines. Various configurations of finite state machines can be selected to define the machine type, the state code, and the flip-flop type. Logic minimization with the K-map approach and the Quine McCluskey scheme is also supported. The tools, denoted as DLD-VISU, help students practice related topics in digital logic design courses. Also, instructors can use the tools to efficiently generate and verify examples for lecture notes or for homework problems and assignments. DLD-VISU was designed relying on a thorough investigation of related pedagogical aspects to define appropriate interactive graphical processes. The decision for a web-based solution, on the one hand, was motivated by making the tools available, portable, expandable, and at the same time transparent to the user. On the other hand, the advocated approach enables instructors to define access rules for their students to assure that students cannot use the tools to solve assessed homework problems or assignments before submission deadline. DLD-VISU supports self-assessment and reflects the student learning process using learning curves. The proposed platform was evaluated both in form of students' feedback as well as by analyzing the impact of using the tools on students' performance.

1 Introduction

Digital logic design (DLD) is a core course in several undergraduate programs including electrical engineering, computer engineering, and computer science. DLD is usually taught in the first or second year. This poses special difficulties to students for four reasons. First, DLD is a comprehensive course with diverse topics that must be covered to enable students to attend advanced courses such as computer architecture and embedded systems. Second, a DLD course is rich in new concepts, theories, and approaches, which are not part of school education, as a rule. Thus, students face these topics without any or with very limited background. Third, solving DLD problems manually is error-prone because of working with large numbers of 1's and 0's. For instance, the state table used to design a small finite state machine with four binary-coded states, four input signals, and two output signals, has 64 lines and 10 columns with a total of 640 zeros and ones. Flipping any of these bits by mistake may lead to an FSM circuit, that doesn't meet the specification. Finding this kind of error is tedious and its correction may require a start from the beginning. This doesn't only cause frustration but also deters many students from trying to solve advanced problems for practicing. Fourth, an essential aspect in digital logic design is to learn how different design alternatives result in different non-functional properties of the digital circuit. For instance, to investigate the effect of the FSM type, the state code, or the flip-flop type on the performance or on the gate usage of a finite state machine, different design alternatives for the same specification should be generated and compared. Given the complexity of generating one design alternative, it is obvious that a comprehensive evaluation of design alternatives is almost impossible.

Digital logic design has long been addressed in education literature. Several papers can be found that focus on DLD course construction [1], [2], [3], the usage of commercial tools and hardware description languages (HDL) for learning DLD [4], [5], and employing programmable logic to enhance the effectiveness of DLD learning process [6], [7]. Using HDL and commercial design tools as learning technologies is very useful. However, these tools operate on two ends of the design process and hide internal design steps, which form the core learning outcomes in a typical DLD course. For instance, a commercial synthesis program can read an FSM specification in form of a state diagram or HDL code and generate the corresponding circuit. The question "How does this generation works", which is in the center of a DLD course, is not answered by these tools. Individual contributions on the visualization and animation for digital logic design can be found in the literature. This related work will be discussed in Section 2.

In this paper we present a web-based tool, denoted as DLD-VISU, for the visualization and animation of different DLD topics. The main contribution of this tool to academic learning consists in a consequent exploitation of relevant pedagogical theories in conceiving the graphical animation process. These theories known as Epistemic Fidelity, Cognitive Constructivism, Dual Coding, and Individual Differences, specify the fundamental requirements for a successful animation solution in the field of education.

Currently, DLD-VISU supports the design of combinatorial circuits using logic gates, multiplexers, decoders, and look-up tables. The input function can be entered as a

- *A. Shoufan is with the Department of ECE, Khalifa University, Abu Dhabi, UAE. E-mail: abdulhadi.shoufan@kustar.ac.ae.*
- *Z. Lu and S.A. Huss are with the Department of Computer Science, Integrated Circuits and Systems Lab, Technische Universität Darmstadt, Darmstadt, Germany. E-mail: {zheng_lu, huss}@iss.tu-darmstadt.dee.*

Manuscript received 20 Mar. 2014; revised 22 Aug. 2014; accepted 3 Sept. 2014. Date of publication 11 Sept. 2014; date of current version 16 June 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TLT.2014.2356464

1939-1382 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Boolean function or as a truth table and can be minimized using K-map or Quine-McCluskey algorithm. Additionally, finite state machines can be synthesized step by step starting from a state diagram. All the design steps are performed under students' interaction, so that intermediate values can be verified. Different FSM configurations regarding the machine type, the state code, and the flip-flop type can be selected to test various design alternatives. DLD-VISU enables instructors to set access rules that define when students can access which topic. This is important for instructors who use graded homework problems and assignments as assessment tools.

The paper is structured as follows. Section 2 reviews the related work and details the contribution of DLD-VISU. Section 3 describes the design concepts behind DLD-VISU. Section 4 details the included learning topics. Section 5 describes some implementation aspects. Section 6 details the tools evaluation and Section 7 concludes the paper.

2 Related Work and Paper Contribution

In this section we review related work on the visualization and animation of digital logic design in the order of their relevance to our work. Then we describe the innovation of DLD-VISU and compare it with related work.

Stanisavljevic et al. presented recently a system for digital logic design and simulation (SDLDS) [8]. The system consists of three modules for design, simulation and evaluation. The design can be carried out either starting from a formal description or by instantiating and connecting library modules. Combinatorial circuits as well as finite state machines of both Moore and Mealy types are supported. A Boolean function is entered as a truth table. The system then creates the canonical forms of the function and draws the corresponding circuits using AND and OR gates as well as inverters. Additionally, the K-map approach is used to create the minimized functions and draw the optimized circuits. A finite state machine can be designed starting from state table that is entered by the user.

In [9] a Windows-based tools suite, denoted as WinLogiLab, is presented that supports interactive learning of the design of combinatorial and sequential logic circuits. The suite includes tools for number presentation and conversion, the design of combinatorial circuits with logical gates, Boolean function minimization using K-map and Quine-McCluskey approaches, and the specification and simulation of general purpose finite state machines.

HADES is a versatile simulation and visualization platform for computer architecture based on Java applets [10]. HADES offers interactive simulations on the gate level and it includes a state diagram editor. The user can select the machine type, specify the inputs and the outputs of the machine, and draw the state diagram. In [11], the authors use the Flash technology to generate what they call Flash notes for DLD topics such as the basic logic gates, simplifying logical circuits, flip-flops.

Several simulators were presented for educational pruposes. For instance, LOG is a digital simulator for UNIX, which was originally developed at UC Berkeley in the 1980s as a tool for teaching logic design. LOG is available as part of the CHIPMUNK package [12]. Logisim is a Java-based simulator of digital systems consisting of gates and flip-flops [13]. LoGen uses dynamic HTML and PHP to generate and simulate logic circuits on the gate level [14]. Other simulators available online include smartsim [15] and easysim [16].

DLD-VISU is a tool for the visualization and animation of digital logic design. Its main contribution consists in the definition of a theoretical learning framework and the application of this framework to a wide range of DLD topics, as depicted in Fig. 1. The framework essentially relies on the findings in the field of algorithm animation to generate graphical processes for learning DLD. The motivation to this approach is that most DLD topics can be described in an algorithmic way. This is not only valid to the main design processes for combinatorial and sequential logic, but also to several intermediate steps such as function minimization using the k-map or the Quine-McCluskey scheme, Boolean function implementation using multiplexers, decoders, NAND gates, or NOR gates. Even individual actions within these intermediate steps can be sophisticated enough for the computer, so that an algorithmic description is justified. One example for that is finding all the prime implicants in a k-map. The biggest challenge for any DLD visualization tool is to find an appropriate abstraction level for the graphical processes that contributes to the learning process effectively. This strongly relies on the instructor's experience in teaching DLD topics and her or his awareness of the level of details most appropriate for topics' presentation. This complies with the Epistemic Fidelity theory as one of the five fundamental theories that we adopted to build the DLD-VISU theoretical framework. This framework will be detailed in the next section. We are not aware of any learning technology that uses a similar approach to produce graphics or graphical processes for DLD visualization.

On the topical level, DLD-VISU focuses on the synthesis of digital logic rather than on simulation. Thus, it is mostly related to SDLDS [8] and WinLogiLab [9]. DLD-VISU supports various design aspects that are not covered by these two solutions as can be seen in the comparison given in Table 1. However, we should emphasize that the main strength of DLD-VISU is not its topic coverage but the way these topics are presented based on the proposed framework. For instance, DLD-VISU is the only solution that addresses the k-map minimization in the methodic way that starts with determining all the prime implicants, identifying the core implicants, and writing the minimized function as a sum of all core implicants and as many prime implicants as necessary. These steps are addressed explicitly and with user interaction and self-assessment. Another example is the implementation of Boolean functions using NAND gates. DLD-VISU is the only tool that supports all possible ways to enter the function, to minimize it, and to implement it using AND and OR gates. Then, the AND and OR gates are replaced by their NAND equivalent circuits and the redundant inverters are removed to obtain the final circuit.

Furthermore, as a web solution DLD-VISU provides two functions that are of high relevance to the educational process as will be detailed in the next section, see

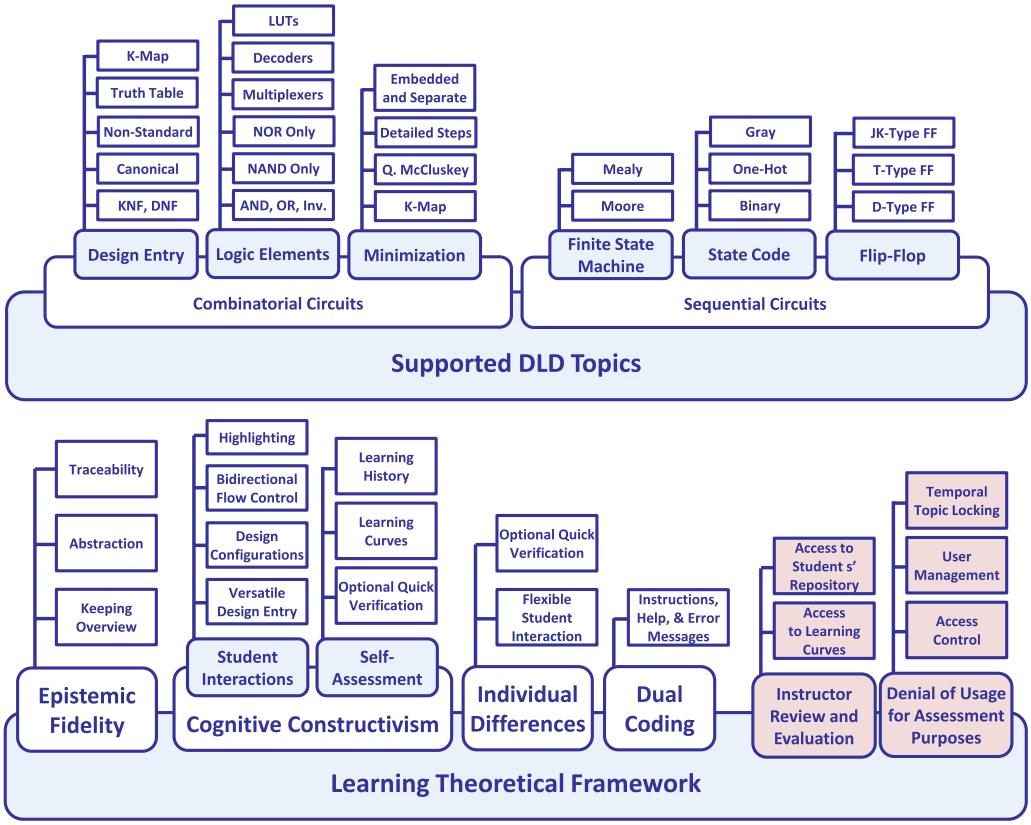

Fig. 1. DLD-VISU theoretical framework and covered topics.

TABLE 1
Comparing DLD-VISU with SDLDS and WinLogiLab

Topic	WinLogiLab	SDLDS	DLD-VISU
Formal entry of Boolean function	Yes	No	Yes
Specifying the function form (DNF, KNF, non-standard, canonical)	No	No	Yes
Design entry as a truth table	Yes	Yes	Yes
Design entry as a K-map	Yes	No	Yes
Minimization using K-Map	Yes	Yes	Yes
Minimization using Quine-McCluskey	Yes	No	Yes
Implementation using AND, OR gates and inverters	Yes	Yes	Yes
Methodical implementation using NAND gates	No	No	Yes
Methodical implementation using NOR gates	No	No	Yes
Methodical implementation using multiplexers	No	No	Yes
Implementation using decoders	No	No	Yes
Implementation using LUTs	No	No	Yes
Design of Moore and Mealy FSMs	No	Yes	Yes
FSM entry as a state diagram	Yes	No	Yes
Selecting or editing binary state code	No	Yes	Yes
Selecting or editing one-hot state code	No	Yes	Yes
Selecting or editing Gray state code	No	Yes	Yes
FSM design using D-type flip-flops	No	Yes	Yes
FSM design using T-type flip-flops	No	Yes	Yes
FSM design using JK-type flip-flops	No	Yes	Yes
Minimizing the state and the output equations	No	Yes	Yes
Drawing the FSM state diagram	No	Yes	Yes

Fig. 1. First, instructors can lock some topics for periods where students need to deliver related homework solutions. For that a user management system was introduced that enables instructors to register their students and to define appropriate access rules. Second, instructors can access students' learning curves and history. Accessing students' learning curves helps instructors evaluate the learning level of students. The students' history can be used as a repository by the instructor for the purpose of review and assessment.

Fig. 2. FSM flowchart.

3 DLD-VISU Design Concepts

DLD-VISU underlies a pedagogical design concept as well as a web design concept. Both will be detailed in the following.

3.1 Pedagogical Design Concept

Visualization and animation have long been used as means for knowledge transfer. In university education, animation gained special attention in learning algorithms since the 1970's. The effectiveness of algorithm animation, however, has always been arguable [17]. Hundhausen et al. identified four theories about the effectiveness of algorithm animation [18]. The *Epistemic Fidelity* theory relies on the assumption that humans construct in their heads symbolic models for the physical world and use these models for their reasoning and action. Visualization and animation aim at a graphical representation of an expert's mental model. The effectiveness of using animation depends on the fidelity of the match between the graphical representation and the expert's mental model. The *Cognitive Constructivism* theory asserts that passive viewing of the graphics may be not very helpful. Instead, an active engagement in the visualization process is essential to construct own understanding. The *Dual Coding* theory assumes that coding the information both as graphics and text promotes the most efficient and robust knowledge transfer. The *Individual Differences* theory assumes that students benefit from visualization differently and this depends on their cognitive abilities and learning style.

DLD-VISU was designed taking these four theories into consideration as detailed in the following. The first two theories will be detailed in two separate sections. The last two theories will be described together in one section.

3.1.1 Epistemic Fidelity

Designing a combinatorial or sequential circuit is a dynamic process with various steps that operate on different data. Our mental model for understanding this design process relies on three principles: keeping overview, abstraction, and traceability. Keeping overview is essential to find oneself in the whole design process. Abstraction is a key principle to focus on the current design step and not to be confused by unnecessary details. Traceability helps students understand the transition from one step to another and identify the origin of each Boolean term or value and each logic component generated through the design process.

Keeping Overview. DLD-VISU enables students to keep track of the learned topic by displaying a *progress chart* for the design process at the top of the animation window. The progress chart shows all the steps of the design process and highlights the current design step. Fig. 2 shows the progress chart of the FSM design process with selecting the state code as the current step. Some steps in the progress chart are complex and consist themselves of several steps. In such a case a *progress sub-chart* appears under the main chart upon arriving at the corresponding step. For instance, the progress chart for implementing a combinatorial circuit includes four main steps. The implementation step herein is complicated when a multiplexer realization is desired. In this case, a progress sub-chart appears to show the details of function expansion according to Shannon's' scheme, see Fig. 3.

Abstraction. DLD-VISU supports abstraction by selecting an appropriate granularity for each animation step on the one hand, and by presenting only necessary data in each step, on the other. Too fine-grained steps would require more clicks, which may cause inconvenience. Too coarse-grained steps, in contrast, would demand that more data is presented in the animation window. This may lead to crowded pages and confusion. DLD-VISU displays in the animation window as much information as necessary for understanding the current design step. During the creation of the state table in FSM design, for instance, the state diagram is displayed so that students can verify each line in the state table by investigating the state diagram. However, during the derivation of the state equations, the state diagram is not displayed anymore because state equations are produced from the state table.

Traceability. Traceability is highly important for learning digital logic design. Remember that solving DLD problems is error-prone as students have to deal with a large number of Boolean terms and values. DLD-VISU enables students to identify the source of each term or value either by a stepwise generation of these terms and values, by highlighting using colors, or both. For example, when the prime implicants are derived from the K-map depicted in Fig. 10, students can trace how each implicant is determined not only with the aid of colors but also by a gradual display of the minterm groupings in the map and the related implicants.

3.1.2 Cognitive Constructivism

The Cognitive Constructivism theory assumes that the user should be engaged in the animation process to attain the desired learning outcome. Passive viewing is not helpful, as a rule. DLD-VISU is an interactive tool that engages the student at different stages of the design process. User interactions in DLD-VISU can be categorized into two groups as detailed in the following.

Design and animation interactions. Four interactions can be identified in this group:

1) The design entry is the student's responsibility. The design can be entered as a Boolean function in different forms, as a truth table, as a K-map, or as a state diagram.

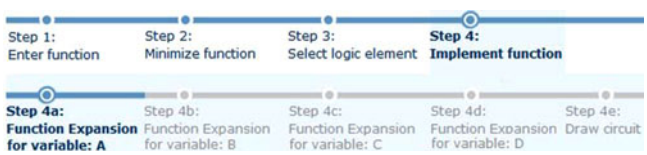

Fig. 3. Progress sub-chart example.

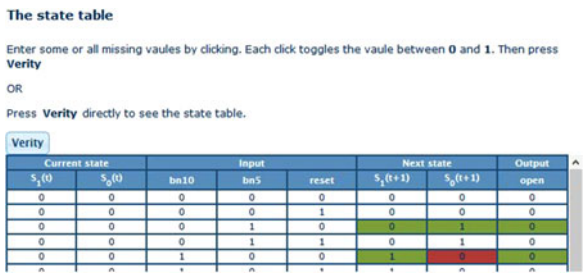

Fig. 4. Self-assessment while creating the state table.

2) The student controls the design flow by selecting different configurations. For combinatorial logic design, the user can select the design entry method, the logic style, and the minimization approach. In the design of FSMs, the user can select the machine type, the state code, and the flip-flop type.
3) The student controls the animation process using forward and backward buttons.
4) The student can use the mouse pointer to highlight some graphical elements and show their belonging for the sake of traceability.

Self-assessment interactions.[1] Animation is a "divide and rule" approach that enables students to understand the design process by investigating intermediate values and final results. Given that, the question arises: How can students make the best of an animation tool like DLD-VISU? It is obvious that a passive observation of the values produced in the internal steps is pointless. Rather, students should develop in their minds or on a piece of paper an idea about the values expected from any step before running it and then compare these values with the actual values produced by the animation program. We believe that this kind of self-assessment is essential to make benefit from any animation solution. To facilitate self-assessment, DLD-VISU provides two techniques, that will be explained using the example of creating the state table within the design of a finite state machine:

1) The state table is used to specify the next state and the output functions. Instead of displaying a full table directly, DLD-VISU prompts the student to enter some or all the values followed by a click on a button called "Verify", see Fig. 4. Then, the state table is displayed with all the correct values. The values that were entered by the student are highlighted by a green or a red background depending on whether the entry was correct or wrong, respectively. Values which were not entered by the student are displayed without background color.
2) The rate of correct entries in this step is registered in what we call the *leaning curve*. This curve captures the success rate in doing some step over the time. Fig. 5 shows that the student has edited a state table five times with an increasing success rate. Note that the different points on the learning curve do not necessarily belong to same state table. When the mouse cursor is moved over a point of the curve, additional information about that point is shown in a rounded rectangle, as can be seen in Fig. 5. Students may use the learning curves to evaluate their learning progress and to manage their learning time more efficiently by focusing on topics with poor learning curves.

1. In the current version, the self-assessment is built-in for K-map and Quine-McCluskey minimization, for state code entry, and for the creation of the state table in the FSM design.

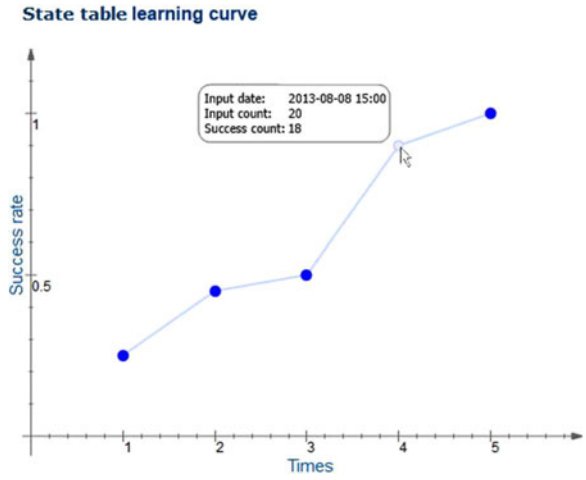

Fig. 5. Learning curve for state table setup.

DLD-VISU includes a menu called *My History* with two items: *My Learning Curves* and *My Animations*. The learning curves are stored under *My Learning Curves* automatically. Upon completing an animation process, students can save all the design steps under *My Animations*. This item includes a table with reference to saved designs. The student can reload any of these designs, e.g., for the purpose of comparison with other design alternatives. Fig. 6 shows an example for an animation history table.

3.1.3 Dual Coding and Individual Differences

The Dual Coding theory assumes that the graphical visualization must be supported by text to achieve effective learning. DLD-VISU makes use of this theory and provides the users with succinct texts. The text can include an instruction to do some action, a help statement, or an error message. The Individual Differences theory assumes that students benefit from visualization differently depending on their cognitive abilities and learning styles. DLD-VISU does not address this point directly as it has no user-specific actions. However, through the ability to control the animation process and to trace the design steps, as well as the ability to enter the design by hand, students can design circuits with different complexity levels and can control the animation process in a way that suits their cognitive abilities. For instance, a student can

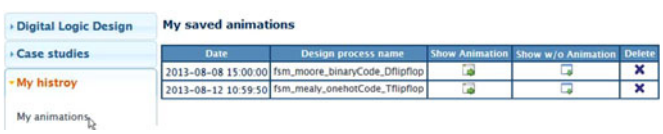

Fig. 6. My animations history example.

Topic		Lock Period	
		From	To
Combinatorial Logic	Logic gates		
	Multiplexers	2013-05-01	2013-05-08
	Decoders	2013-05-09	2013-05-12
	Look up tables		
	LUT s+ MUXs		
	K-map minimization		
	Quine-McCluskey minimization		

Fig. 7. Access control rules example.

work on a K-map with three or four variables depending on her or his learning stage.

3.2 Web Design Concept

The design of DLD-VISU as a web solution was driven by our intention to attain the typical advantages of a web application, on the one hand. These advantages include availability, portability, and update-ability. Students can access DLD-VISU from any computer with an internet connection using any web browser. DLD-VISU can be updated on the server transparently to the user.

On the other hand, the proposed solution is intended to accompany students taking a DLD course. We are aware that some instructors do not welcome a learning technology that enables students to generate problem solutions. Students may use such technology to solve homework problems and assignments. To tackle this problem, DLD-VISU is enhanced with a special *Access Control System* (ACS) that differentiates between three types of users: Instructor, Student, and Guest.

An instructor uses the web to register. Besides some personal data, an instructor must enter his professional email address and website. The system administrator verifies the data usually through online search and sends a confirmation along with log-in data to the instructor. An instructor can then log in and perform one of following:

1) An instructor can register one or more students that attend her or his course using the web. For this purpose the student name and email address must be entered. Upon receiving the request, the system automatically adds the students to the database related to the corresponding instructor and sends log-in data to them.
2) To assure that students cannot use DLD-VISU to generate solutions for graded homework problems or assignments, the instructor can lock different topics using a list that we call the *Access Control List*, see Fig. 7. The upper entry in the figure, for instance, shows that students cannot use DLD-VISU to design combinatorial logic using multiplexers in the time period from the first to the 8th of May 2013.
3) An instructor can run all the animation functions of DLD-VISU.

A student willing to use DLD-VISU has to send an inquiry to her or his instructor. After getting log-in data, students can log in and use the animation tools according to the access control rules defined by the instructor. This indirect registration approach aims at reducing the administrative overhead which would be needed to verify that some student really attends the course of a specific instructor.

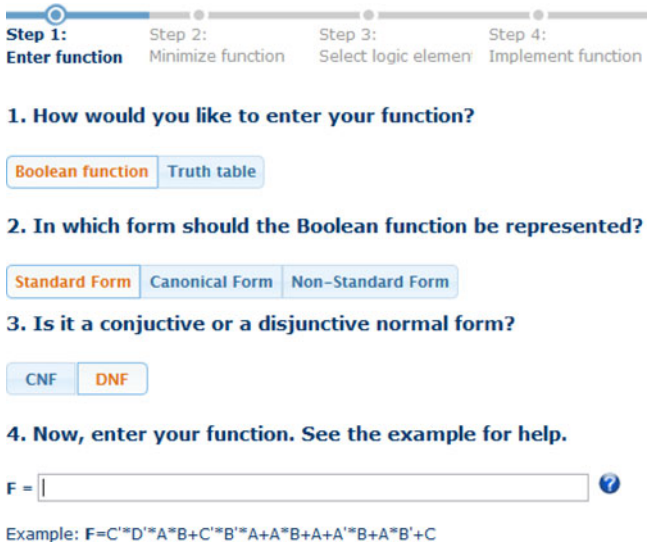

Fig. 8. Design entry as a boolean function.

Users who wish to use DLD-VISU without registration or log-in can access the tools as guests. A guest, however, can only run some built-in case studies.

4 Visualization and Animation in DLD-VISU

This section details the visualization and animation processes supported in the DLD-VISU Beta version. This includes the design of the combinatorial logic and finite state machines. Boolean function minimization using K-map or Quine McCluskey algorithm is a part of the combinatorial logic design but can also be invoked separately.

4.1 A&V of Combinatorial Logic Design Process

A combinatorial logic circuit may be implemented using gates, multiplexers, decoders, or look-up tables. DLD-VISU partitions the design process of combinatorial circuits into four steps, see the progess chart at the top of Fig. 8.

4.1.1 *Function Entry*

The student can enter the function either in a Boolean form or as a truth table. Additionally, if K-map minimization is used separately, students can fill in the K-map directly. When a Boolean function is selected the student is requested to choose the standard form, the canonical form, or the non-standard form. If a standard form is selected the student can select the disjunctive normal form (DNF) or the conjunctive normal form (CNF). Then the Boolean function can be edited using the same format given in the last line as an example, see Fig. 8. Similarly the student can select the canonical form (sum of minterms) and enter the function as a list of minterm numbers such as $F = \Sigma m(0, 1, 4, 9)$. The non-standard form can include sums of products as well as products of sums. Questions 2, 3 and 4 in Fig. 8 appear dynamically depending on the answer given to each previous question. This helps students focus on one question at one time point, which is in line with the abstraction concept of DLD-VISU. Note that requesting students to answer the above questions aims at making them familiar with the terminology of Boolean algebra.

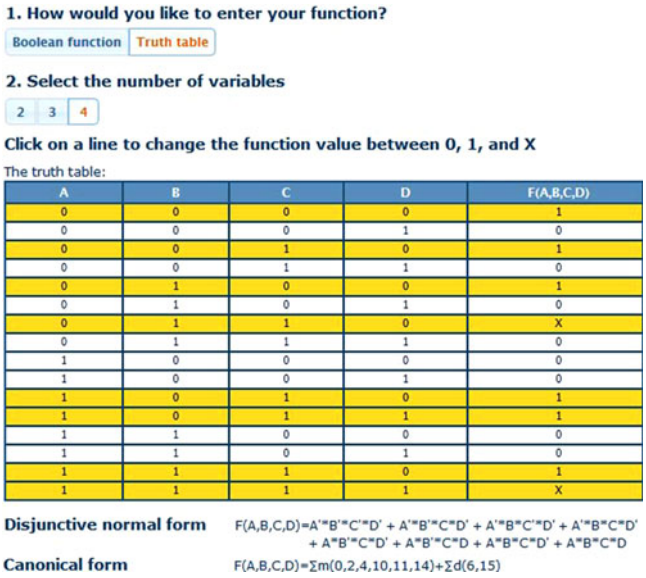

Fig. 9. Design entry as a truth table.

If a truth table is chosen to enter the function, then the student is requested to enter the number of variables. Then, a corresponding truth table is displayed in which all output values are 0 by default. The student can click on the desired table line to switch the corresponding output value between 0, 1, and X (Don't care). While editing the truth table the corresponding Boolean function represented in the disjunctive normal form and in the compact canonical form appears under the table automatically, see Fig. 9.

4.1.2 Function Minimization

The minimization of logical functions is a main learning outcome in any DLD course. Students should be kept aware that a circuit, which implements a function with less components, is more economic, more reliable, shows better performance, and consumes less power. Minimizing logical functions using Boolean algebra is only practical for small functions with limited number of variables. The K-map approach is popular for functions of up to four variables. The Quine McCluskey algorithm is more appropriate for functions of more variables. DLD-VISU supports both approaches as will be detailed in the following.

In the minimization step the student is requested to select the minimization method. Depending on this selection a progress sub-chart is displayed, see Fig. 10.

K-Map minimization. In the first step of K-map minimization, the map is set up with the function minterms as depicted in Fig. 10. Following, all the prime implicants are determined. For self-assessment, students are able to edit the implicants and verify their entries. The same applies to the next step for determining the core implicants. Fig. 10 shows the animation window after completing all the minimization steps, for space reasons. This figure depicts that the student has entered three out of four prime implicants in step 2, whereas the third entry is wrong. In steps 3 and 4 the student did not make any entries.

A prime implicant as a Boolean term is displayed in the same color as the outline color of the corresponding

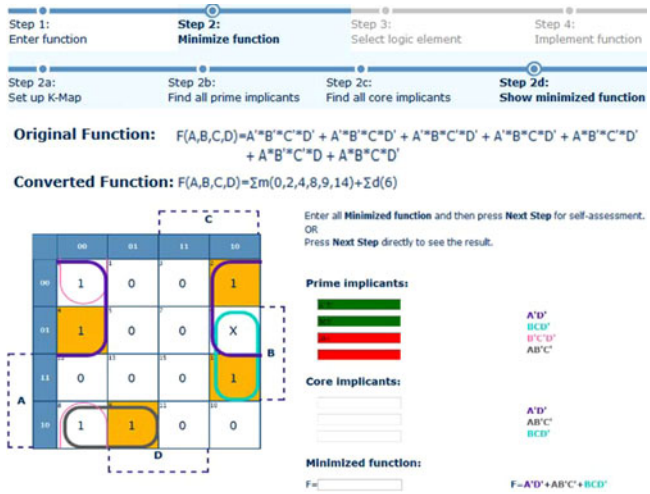

Fig. 10. K-Map minimization using prime and essential prime implicants.

grouping on the map. The squares with orange background in the map highlight the minterms that are covered by only one prime implicant, thus, justifying why this implicant is a core implicant. For instance, the minterm AB'C'D is only covered by the prime implicant AB'C'. This is why AB'C' is a core implicant which is highlighted on the map by a bold outline. Prime implicants stay with a thin outline such as B'C'D'.

Quine-McCluskey minimization. For functions of more than four variables, identifying the implicants and the core implicants using K-map can get tedious and error-prone. The Quine-McCluskey method (QMC) is a tabular search-based approach which makes it more suitable for minimizing large functions. QMC proceeds as follows. The first step consists in finding all the implicants and is itself a multi-step procedure as depicted in the example of Fig. 11. In an initialization step, the function minterms are grouped according to the number of 1's and listed in blue color in the second column of the table. This grouping according to the number of 1's accelerates the search for adjacent minterms that follows in a successive way. The adjacent minterms found in the second column are listed one by one in a new column (Step 1), where the different bit is replaced by "-".

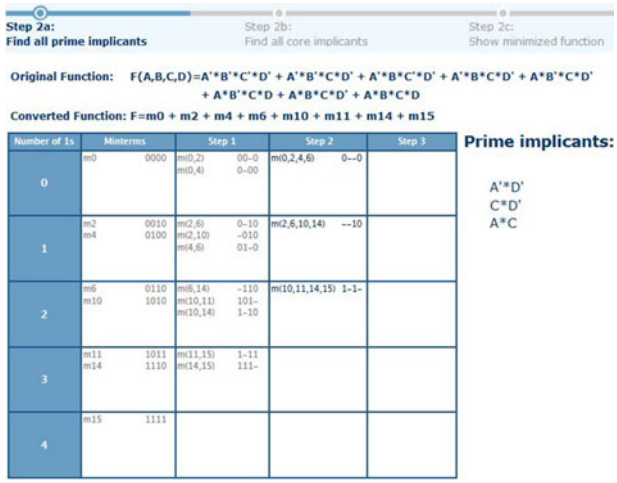

Fig. 11. Minimization with Quine McCluskey: Finding prime implicants.

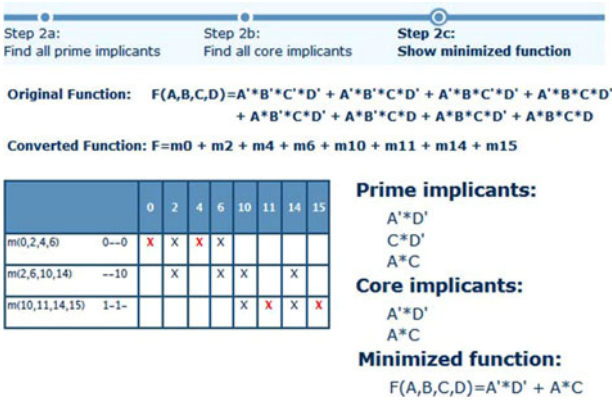

Fig. 12. Minimization with Quine McCluskey: Finding core implicants.

Any minterm, for which an adjacent minterm is found, is marked by a gray color. Upon completing the list in Step 1, this list is searched for adjacent minterms, which are inserted in a new column (Step 2). For that the sign "-" is treated as a third bit value.

This procedure is repeated iteratively until no adjacent minterms are available. The implicants correspond to the remaining entries with blue color. In the second step the core implicants are determined. This is done using a matrix that shows which minterms are covered by each implicant. In Fig. 12, for instance, the minterms m_0 and m_4 are only covered by the implicant A'D'. That is why this implicant is a core implicant. This is indicated by highlighting the check sign "X" in the corresponding columns. In the last step the minimized function is set up as the sum of all core implicants and as many implicants as necessary to cover all the minterms of the function.

4.1.3 Selecting Implementation Style

DLD-VISU supports the design of combinatorial circuits with gates, multiplexers, decoders, as well as with look-up tables and multiplexers which is typical in modern FPGAs. Students can select one of these alternatives as can be seen in Fig. 13. When logic gates are selected, students can also specify the type and the size of the gates to be used.

4.1.4 Implement Function

In this step, the minimized function is mapped to the combinatorial circuit using the selected components.

Implementation with gates. The implementation with AND and OR gates is straightforward. Fig. 14 shows the

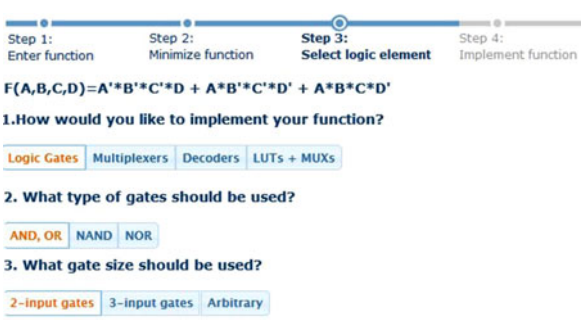

Fig. 13. Selecting the Implementation Logic Elements.

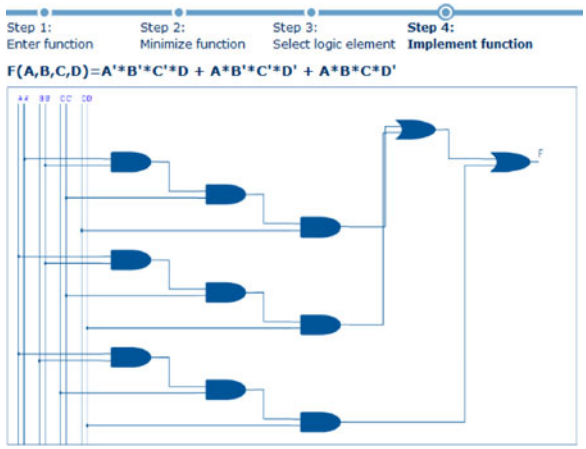

Fig. 14. Implementing a combinatorial circuit with AND and OR.

implementation of the function $F(A,B,C,D) = A'B'C'D + AB'C'D' + ABCD'$ with two-input AND and OR gates.

The implementation with NAND or NOR gates is more complex and accomplished as a four-step process. This process is explained in the following for the same function above using two-input NAND gates.

1) First, the circuit is implemented with AND and OR gates only. The resulting circuit will be the same as the one given in Fig. 14.
2) Each AND gate is replaced by its equivalent NAND circuit, which consists of two NAND gates. The first NAND gate takes the inputs. Its output is connected to both inputs of the second gate that realizes an inverter. Similarly, each OR gate is replaced by its equivalent NAND circuit which consists of three NAND gates. Fig. 15 shows the result of this step.
3) In the next step, all the redundant inverter pairs are found and marked as seen in Fig. 16. A redundant inverter pair consists of two consecutive inverters.
4) Finally, the marked redundant inverter pairs are deleted as depicted in Fig. 17.

Implementation with multiplexers. If multiplexers are selected to implement the function, the Boolean function is first expanded according to Shannon's scheme. The expansion relies on using the function variables as control signals for the MUXs. The MUXs' data inputs either stem

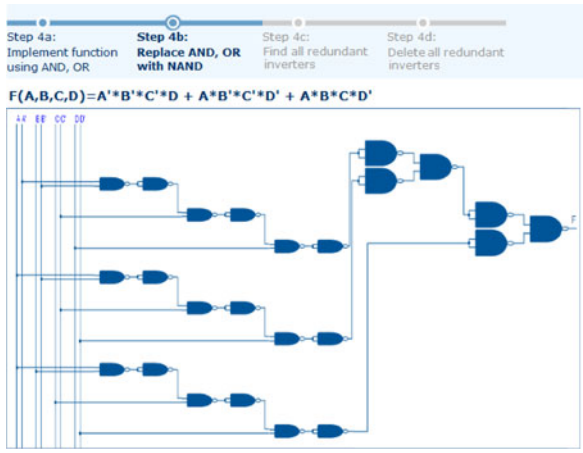

Fig. 15. Replacing AND and OR gates by their NAND equivalents.

Fig. 16. Identifying redundant inverters.

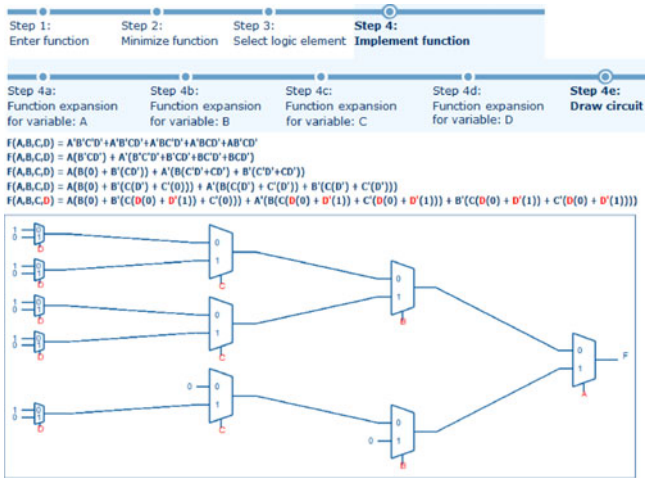

Fig. 18. Implementing a combinatorial circuit with multiplexers.

from the outputs of previous MUXs or are connected to the constant values 1 or 0. Currently only 2-1 MUXs are supported by DLD-VISU. The function expansion according to Shannon is displayed step by step followed by drawing the corresponding multiplexer, see Fig. 18.

Implementation with decoders. DLD-VISU supports implementations with 3×8 and 4×16 decoders. If the function has two or three variables, the circuit is built up using one 3×8 decoder. If the function has 4 variables, the circuit can be implemented either using one 4×16 decoder or two 3×8 decoders. Functions of five variables are implemented using two 4×16 decoders. Fig. 19 shows how DLD-VISU generates a decoder circuit for a four-variable function using two 3×8 decoders.

Implementation with look-up tables and multiplexer. DLD-VISU supports implementations with 16-bit lookup tables. Additionally, five-variable functions can be implemented using two look-up tables and one multiplexer. The goal of this implementation form is to introduce students to SRAM-based field-programmable gate arrays (FPGA).

4.2 A&V of Sequential Logic Design Process

DLD-VISU visualizes the design of finite state machines in ten steps as was shown in the progress chart in Fig. 1.

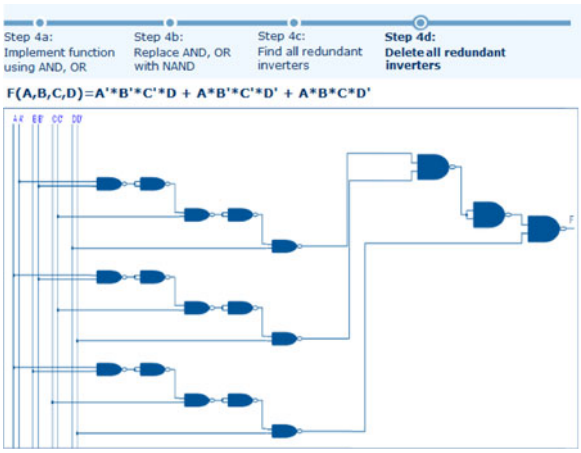

Fig. 17. Deleting redundant inverters.

Both Moore and Mealy machines are supported with binary, Gray, and one-hot state code as well as D-type, T-type, and JK-type flip-flops. Thus, a total of 18 design alternatives can be visualized. For space reasons, however, the design steps described below takes only one of these design alternatives into consideration. This design alternative is a Moore machine, with binary-coded states and D-type flip-flops.

4.2.1 State Diagram Entry

After selecting the FSM type, students can specify the machine behavior as a state diagram. Fig. 20 shows an example for an edited state diagram of a simple Moore machine. Four buttons are available to enter the state diagram. The properties of a state or a transition can be edited in the boxes right to the state diagram.

State and transition properties differ depending on the machine type. In a Moore machine, for instance, the state has two properties: the state name and the output signals that should be active in that state. Additionally, one state can be marked as the start state, which the machine returns to after reset. A transition has just one property, which is the Boolean expression that represents the condition for that transition. The Boolean expression can be the name of a single input signal or a complex expression with several input signals. To build such an expression in DLD-VISU, the characters "!", "*", and "+" are used for inverting, AND, and OR operation, respectively.

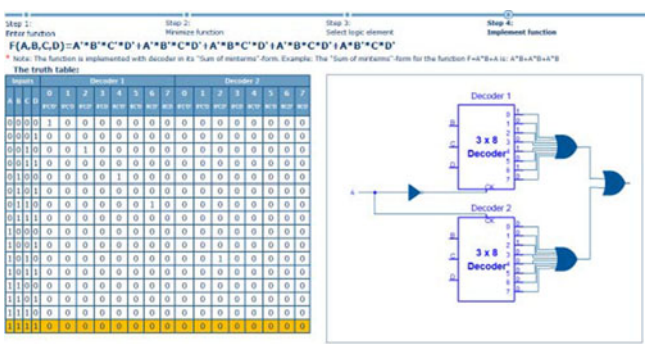

Fig. 19. Implementing a combinatorial circuit with decoders.

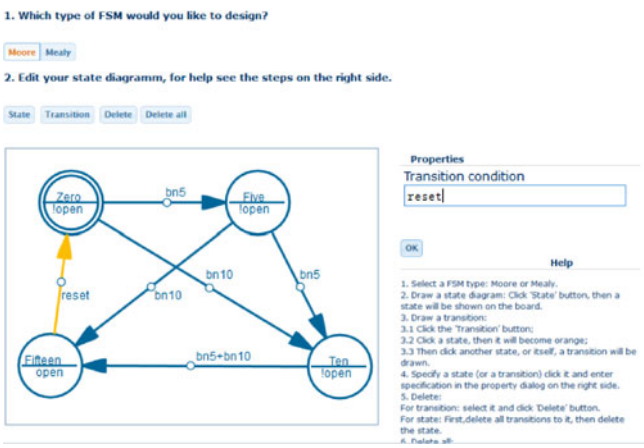

Fig. 20. Editing state diagram for FSM design.

4.2.2 State Code Selection

The state code has an important effect on the behavior and the resource usage of a FSM. One-hot code, for instance, enables a relatively simple input and output logic. The result is a higher-performance FSM, however, at the cost of flip-flop usage. With DLD-VISU students can select and enter the state code. Consequently, the state diagram is regenerated, where the state names are replaced by the state code, see Fig. 21. The correct state code table is also displayed so that the student can map the state code to the state names. With the aid of background color, the student can assess which entries are correct or wrong.

4.2.3 State Table Creation

From the state diagram with encoded states, the state table can then be created as seen in Fig. 22. This figure also shows that the student has entered some values for the next state and the output signals. Correct and wrong entries are marked by a green or red background color, respectively.

For traceability, the state diagram is also displayed in this step. In Fig. 22, however, we cut out the state diagram, for space reasons. From the following step on, the state diagram will not be displayed anymore because the next steps are accomplished based on the state table only. This complies with the abstraction principle DLD-VISU is built on.

4.2.4 State Memory Design

This step simply consists in selecting the type of the flip-flops that should be used to implement the state memory. The number of generated FFs depends on the number of states and the state code as can be seen in Fig. 23 for the example FSM. The FSM general architecture shown in this

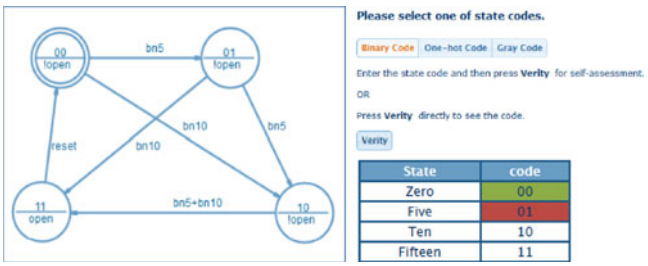

Fig. 21. State coding.

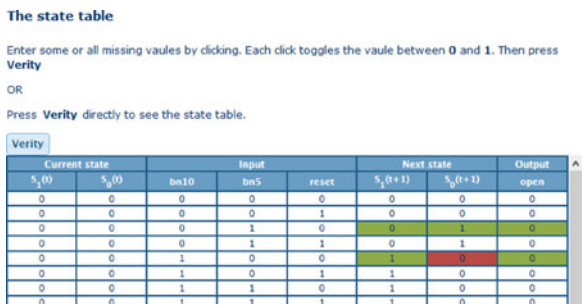

Fig. 22. State table creation.

figure is displayed to keep overview of the current design step by highlighting the corresponding component. In the following, however, this architecture diagram will not be shown again, for space reasons.

4.2.5 Design of Combinatorial Input Logic

The design of the input logic circuit is accomplished in three steps (steps 5, 6, and 7 in the FSM progress chart, see Fig. 2):

In the first step the state equations are derived from the state table. A state equation describes the next state as a function of the current state and the input signals, see Fig. 24. Using the mouse cursor, a table line and the corresponding term in the equations can be highlighted for the sake of traceability. Note that the column related to the output signal is omitted in the table of Fig. 24. This is because output signals do not affect the input logic. DLD-VISU hides output signal columns in this step for the purpose of abstraction.

In the next step the state equations are minimized. The minimization is performed using the Quine Mc-Cluskey algorithm implicitly, i.e., without animation. FSM design is an advanced topic in digital logic design and it is assumed that students at this stage are familiar with the design of combinatorial logic including minimization.

Lastly, the combinatorial input circuit corresponding to the minimized function is displayed gate by gate under highlighting the corresponding term in the minimized state equations for traceability, see Fig. 25.

4.2.6 Design of Combinatorial Output Logic

The design of combinatorial output logic is visualized in three steps of the progress chart (steps 8, 9, and 10). These steps are highly similar to the construction of the input logic. Therefore, only the final circuit is shown here for brevity, see Fig. 25. In the first step, the output equations are

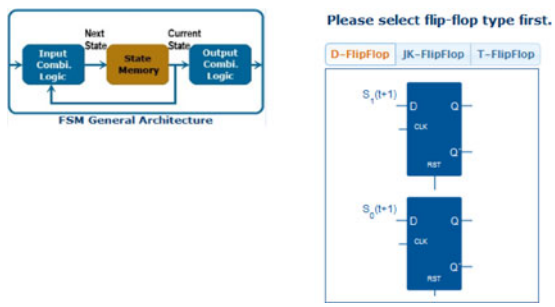

Fig. 23. State memory design.

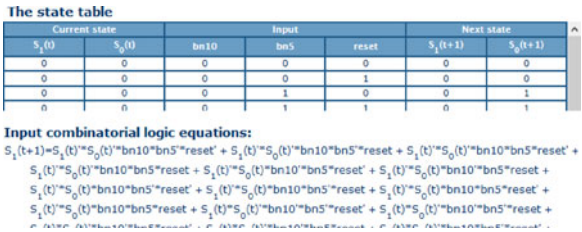

Fig. 24. Generation of state equations.

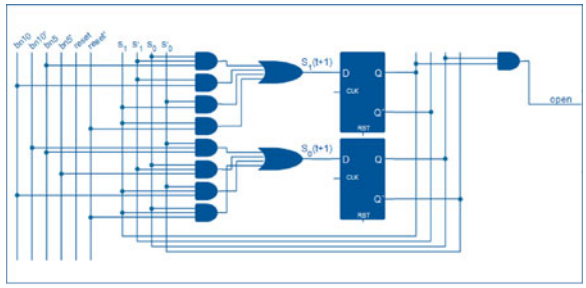

Fig. 25. Output circuit.

derived from the state table. An output equation describes an output signal as a function of the current state in Moore machines. In Mealy machines, an output signal is a function of both the current state and the input signals. During deriving the output equations, therefore, DLD-VISU hides the next-state columns in the state table for Mealy machines as well as the input signal columns for Moore machines. In the next step the output equations are minimized. Finally, the combinatorial output circuit is displayed gate by gate to obtain the overall FSM circuit.

5 Implementation

DLD-VISU was implemented using the integrated environment SSH2, whereas SSH2 stands for Struts 2.0, Spring, and Hibernate. Struts 2.0 is a framework to develop JAVA EE web applications supporting the MVC pattern. Using this pattern, the Model, i.e., the application logic, is separated from the View, i.e., the HTML page presented to the user, and the Controller which is responsible for the data exchange between the view and the model. Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Finally, Spring is an application framework and a container for inversion of control (IoC) that is used to manage Java objects via dependency injection. Spring reduces the coupling between the Controller and the Model.

The interaction between the client and the server in DLD-VISU shall be illustrated using the sequence diagram depicted in Fig. 26. This diagram relates to the state coding step depicted in Fig. 21 (without self-assessment) and shows how the state code table is generated in nine steps.First, when the user clicks on “Binary Code”, JavaScript generates a request in the form of an action named “getBinaryCode. action” and sends it to the server using the Ajax technique and the jQuery library. Ajax enables a web application to send data to the server and to get the response without refreshing the current page. The action parameters include the ID of the state diagram under consideration and the name of the callback function written in JavaScript. The callback function is responsible for the display of the table in the browser. DLD-VISU uses Tomcat as a web server. Upon receiving the request, Tomcat calls the Struts 2 filter that intercepts the request. Struts 2.0, then looks up the class name “stateCodeAction” corresponding to the action label in the configuration file “struts.xml” and sends the class name to Spring. Spring, then, looks up the actual action class “com.org.action.StateCodeAction” in the configuration file

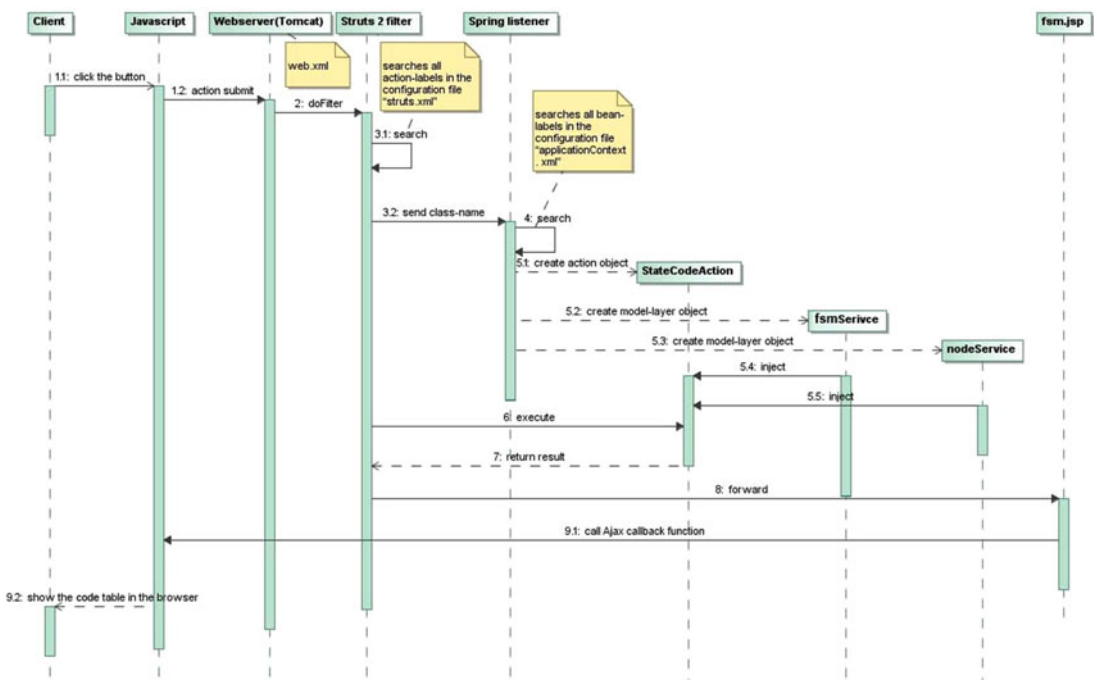

Fig. 26. Client-server interaction to create a code table in the FSM design process.

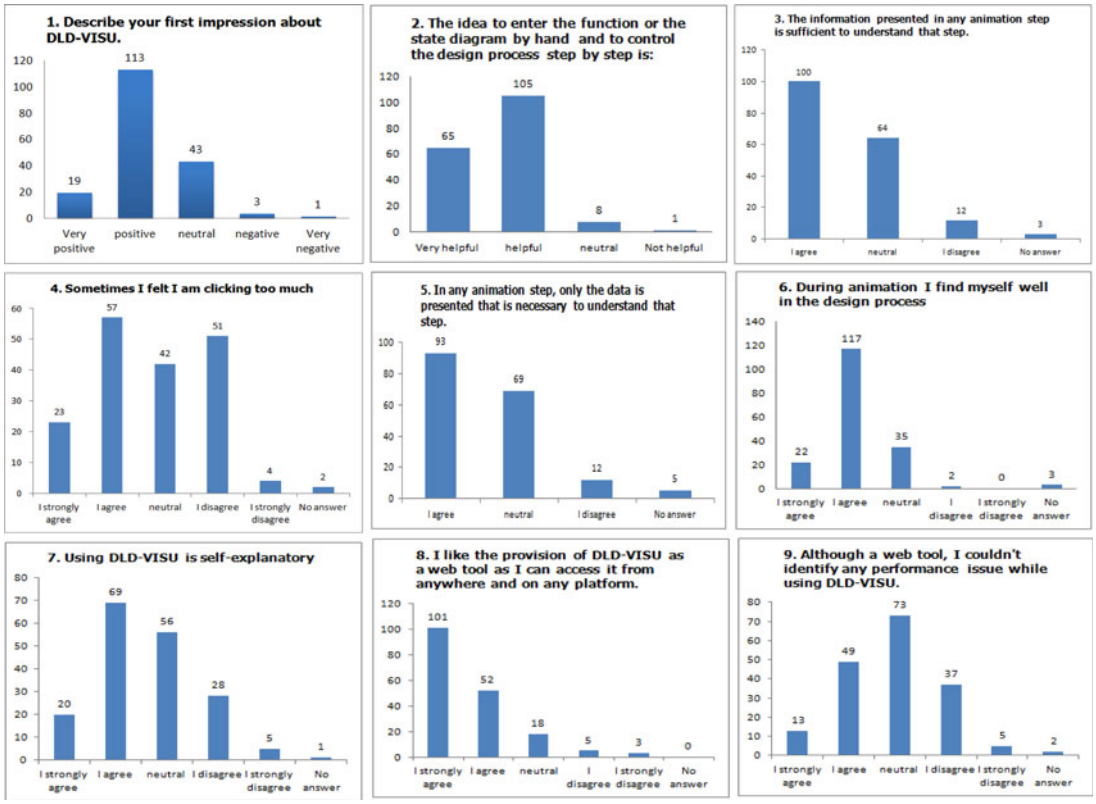

Fig. 27. Students feedback results.

"applicationContext. xml". Spring creates a new action-object "StateCodeAction", and two model-layer objects "fsmSerivce" and "nodeService". The latter two model objects are injected into the action object. Struts executes the method of the new action object. In this example the method is "getBinaryCode". The model returns the execution result to Struts, which is the code table in this case. Struts looks up the success tag in the file "struts.xml" and forwards the results to the JSP file specified in the success tag. The browser gets the result from the JSP and calls the Ajax callback function that updates the part of page with the code table.

The view layer of the MVC pattern runs in the browser to receive user entries and to generate the animation based on the data sent by the server. For that *JavaScript* and the *JSXGraph* library are utilized. *JSXGraph* is a cross-browser library that is implemented in JavaScript completely. *JSXGraph* supports both the Web Vector Graphic Language *VML* (only Microsoft Internet Explorer before Edition 9.0) and the Vector Markup Language *SVG* (other popular browsers including Firefox, Chrome and Safari).

6 DLD-VISU Evaluation

6.1 Students' Survey

This section presents the results of a survey conducted with 179 students at Technische Universität Darmstadt against the end of the semester, in which they took the DLD course.

Recall that DLD-VISU was realized based on four pedagogical theories, that describe the requirements for an effective animation solution. In the previous sections we tried to highlight how these theories were mapped to concrete aspects in the presented software solution. The goal of students' feedback is to see how far this mapping served its purpose by asking questions related to their experience with DLD-VISU.

For instance, Question 5 in Fig. 27 indirectly addresses the abstraction concept in DLD-VISU. The answers to this question shows that 54 percent of the students agree that in an animation step only the data is presented which is necessary to understand this step. Almost 39 percent gave, however, a neutral answer to the same question. The last percentage was high enough for us to review our solution and to improve its abstraction feature at several points, which is an essential objective of the feedback system. The traceability principle is addressed in Question 3 and students gave similar answers as to the abstraction question. Almost 78 percent of the students seem to have found their way through the animation steps according to the answers to Question 9. This indicates that DLD-VISU helped them keeping the overview of the design process. Remember that keeping overview, abstraction, and traceability are the three principles, which reflect our instructional model.

Question 2 implies that students must be active in the animation process by entering the design specification and controlling the design process. Recall that the Cognitive Constructivism theory presumes user interaction for an effective animation tool. Interestingly, almost all the students found this kind of interaction as helpful or very helpful.

Almost half the students found that DLD-VISU is self-explanatory according to Question 7. This confirms that the textual instructions and messages were sufficient for this part of students to use DLD-VISU without extra help. Adding text to the graphic is essential for effective animation according to the Dual Coding theory. The Individual

Differences theory assumes that students benefit from visualization differently depending on their cognitive abilities and learning styles. Again, DLD-VISU does not define user-specific actions but students, for instance, can try designs at diffirent difficulty levels that fit their learning abilities. The granularity of the animation step (e.g., should the entries of a K-map diplayed all at once, line by line, or square by quare?) is an important parameter that may be set by the students according to their learning level. The need for such an improvement in DLD-VISU can be seen from the answers to Question 4. While 45 percent feel that the animation steps are too fine-granular because of the need to click frequently, 30 percent have an opposite opinion.

Question 1 relates to the first impression which is highly important for the acceptance of any e-learning tool. We believe that, besides its advantage in the learning process, the clear layout and the uncomplicated operation of DLD-VISU have contributed to the positive or very positive impression which 73 percent of the students had. Question 8 shows that most students welcome on-line learning due to various advantages including the flexibility and the lack of the need to download and install software on the own computer. Only 37 percent of the responsees experienced a performance issue in using DLD-VISU as a web solution according to Question 9.

6.2 Impact on Students Performance

The survey presented in the previous section gives an indirect evaluation of the effectiveness of DLD-VISU. At that time the self-assessment system was not implemented yet. Currently, the learning curves that record the self-assessment results provide an efficient way to evaluate DLD-VISU.

6.2.1 Evaluation in Fall 2013

In the previous semester, DLD-VISU was presented in the classroom to 38 electrical and computer engineering students taking the DLD course at the sophomore level. Interested students were asked to request access data by sending an email to the instructor. Nineteen students were registered. However, out of these students only seven accessed and used DLD-VISU. This low usage may be attributed to the fact that we presented the tool very late in the semester, specifically in the last week before the final exam. Each of the seven students completed the K-map minimization process for 4.8 functions in average. The overall success rate for all students and all trials is 79.74 percent. In the final exam all the students had to apply K-map minimization as a problem part with 5 marks. We found out that DLD-VISU users solved this final-exam problem part with an average of 4.49 marks compared to 3.89 marks that non DLD-VISU users achieved in average. Thus, DLD-VISU users performed in this task 15.4 percent $(100 * (4.49 - 3.89)/3.89)$ better than the others. However, this improvement cannot be seen as a net gain: we compared the overall performance of the students and found out that the DLD-VISU users have a higher average, however only by 5.1 percent.

6.2.2 Evaluation in Spring 2014

In this semester, we conducted an experiment to gain more data on the effectiveness of DLD-VISU by answering the following question: Given a specific minimization task, what is the advantage of using DLD-VISU to perform this task over the traditional manual approach? This question is especially challenging: If a student starts solving the problem using DLD-VISU, then she or he would get immediate feedback and see the correct solution so that a succeeding manual solution of the same problem would not help in assessment. Even if the student starts with the manual solution, she or he would gain experience with the function so that solving it with DLD-VISU would be easier. For a valid and fair comparison, therefore, we completely avoided that a student solves the same problem twice.

For an in-depth understanding of the effectiveness of DLD-VISU, furthermore, we gave minimization problems of different difficulty levels. For that we specified the concept of difficulty level as follows.

Difficulty Level. The difficulty level of the function minimization problem using K-map increases with the following factors.

1) The number of prime implicants N_{PI} in the K-map. When N_{PI} increases, then the probability of overlooking one or more prime implicants is higher.
2) The difference between the number of prime implicants and the number of core implicants, which we refer to as D_{PI-CI}. A higher D_{PI-CI} value indicates the availability of more overlapped prime implicants and, thus, the availability of more than one optimal function. In contrast, if D_{PI-CI} is zero, then there is only one optimal function.

Based on this understanding, we specify the difficulty level using the heuristic formula $DL = N_{PI} + D_{PI-CI}$, or $DL = 2N_{PI} + N_{CI}$, where N_{CI} is the number of the core implicants.

Experiment execution. The experiment was conducted with 49 students who had already took or were taking the DLD course at the Technische Universität Darmstadt and at Khalifa University, respectively. The experiment participants at both universities were familiar with the K-map minimization approach and had a good understanding of the related concepts minterm, implicant, prime implicant, and core implicant. The students were asked to bring their laptops to the class room and to make sure they have access to the internet. One day before we registered the students in the DLD-VISU system and we divided them into two groups (Group A and Group B) with almost comparable average performance in the course. At the start of the experiment we reviewed the K-map approach and demonstrated DLD-VISU for the first time. Students were tutored step-by-step how to use DLD-VISU to minimize Boolean functions and we made sure that all the students run one example successfully, which was first detailed on the white board. The K-map review and the DLD-VISU demo took almost 10 minutes. Each student had to minimize eight functions (named F_1 to F_8), whereas the index in the function name corresponds to its difficulty level. These functions are

1) $F_1 = \Sigma m(0,3) + \Sigma d(1,2)$
2) $F_2 = \Sigma m(0,1,2,3,13,14) + \Sigma d(12,15)$
3) $F_3 = \Sigma m(0,1,2,3) + \Sigma d(9,11)$
4) $F_4 = \Sigma m(4,5,14,15) + \Sigma d(7,13)$

TABLE 2
Performance Out of 100 Marks & Processing Time for Groups A and B

	Approach	No. Participants	Performance	Processing Time
Group A	Manual	25 Students	80.14	39.92 min
Group B	DLD-VISU	24 Students	93.79	32.13 min
Difference			13.65	7.79 min
Advantage of using DLD-VISU			17.03%	19.5%

5) $F_5 = \Sigma m(4, 5, 7, 13, 14) + \Sigma d(9, 11, 15)$
6) $F_6 = \Sigma m(0, 1, 2, 7, 8, 9, 13) + \Sigma d(10, 11)$
7) $F_7 = \Sigma m(0, 2, 5, 10, 12, 15) + \Sigma d(8, 13)$
8) $F_8 = \Sigma m(0, 2, 10, 12, 13, 15) + \Sigma d(5, 8)$

The 25 students of Group A had to minimize these functions manually. The 24 students of Group B used DLD-VISU to do the same. The functions had to be processed in the order of their indexes. Manual solutions had to be delivered on paper. DLD-VISU solutions are evaluated automatically and grades (success rates) are registered in the learning curves, as it was detailed in Section 3.1.2. The time amount needed to complete the manual solution or the DLD-VISU solution by each student was recorded.

Results and analysis. Students had to determine all the prime implicants and core implicants before writing the minimized function. The manual solutions were graded using the same pattern used in DLD-VISU to determine the success rate. Table 2 shows the average performance of both groups as well as the average time that was needed to complete the respective task. The table shows that DLD-VISU users completed the task in 19.5 percent less time and performed almost 17.03 percent better than non DLD-VISU users. Given the sample size of 49 students and the constraints, under which the experiment was conducted, these values show a clear advantage in using DLD-VISU. Remember that the participants had no previous experience with DLD-VISU in contrast to the manual solution. This doesn't only make the experiment results more significant, but it also shows the ease of operation of the proposed tools.

Another aspect that can be investigated is the student performance in relation to the difficulty level. Fig. 28 shows the advantage of using DLD-VISU as a function of the difficulty level. The curve points in this figure were determined using the same formula used to calculate the advantage of using DLD-VISU in Table 2, however, for each question separately. For instance, the average performance of DLD-VISU users (Group B) in minimizing F3 was almost 14 percent better than the performance of Group A. In spite of two exceptions, the trendline of this function shows that the advantage of using DLD-VISU increases with the difficulty level. The effect of the increasing difficulty level seems to be compensated by the improved learning level with every trial due to the immediate feedback that students get from DLD-VISU. We believe that this feature is especially important for the acceptance of DLD-VISU. The two exceptions F5 and F7 may be associated with the way the experiment was executed and with the selection of these functions itself. As mentioned before, F5 was processed directly after F4. However, F5 had accidentally a large similarity with F4, so

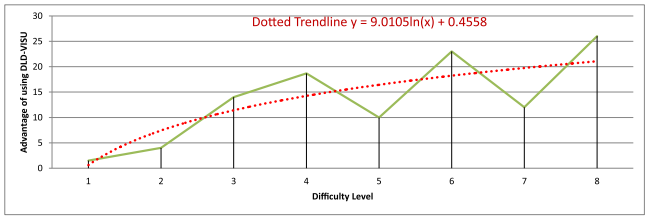

Fig. 28. Advantage of using DLD-VISU against difficulty level.

that both groups performed well in minimizing F5. However, it should not be excluded that these exceptions were —at least partially— caused by issues in the definition of the difficulty level itself. Evaluating this aspect needs more data, which will be a part of future work.

7 Conclusion

DLD-VISU was built based on a long experience in teaching digital logic design and a critical view of effective methods to present its different topics. The proposed software solution is a reflection of this experience under consideration of important theories related to the effectiveness of algorithm animation. We believe that this approach is of general value in the field of learning technology and can be applied beyond digital logic design. The presented results in terms of students' survey and experiments showed the effectiveness of the tools. Informal discussions with the students who tested the tools revealed strong enthusiasm for DLD-VISU. Also, several instructors whom we talk to welcomed the idea of DLD-VISU especially the detailed presentation of the FSM design process. In addition to expanding its functionality to include simulation and further topics, the long-term plan is expand DLD-VISU to a course-based animation platform for different courses.

References

[1] S. Steele and G. Balazs, "A course sequence for the teaching of digital systems," *IEEE Trans. Educ.*, vol. 9, no. 4, pp. 198–201, Dec. 1966.
[2] O. Adamo, P. Guturu, and M. Varanasi, "An innovative method of teaching digital system design in an undergraduate electrical and computer engineering curriculum," in *Proc. IEEE Int. Conf. Microelectron. Syst. Educ.*, 2009, pp. 25–28.
[3] J. Amaral, P. Berube, and P. Mehta, "Teaching digital design to computing science students in a single academic term," *IEEE Trans. Educ.*, vol. 48, no. 1, pp. 127–132, Feb. 2005.
[4] G. Puvvada and M. Breuer, "Teaching computer hardware design using commercial cad tools," *IEEE Trans. Educ.*, vol. 36, no. 1, pp. 158–163, Feb. 1993.
[5] H. Abidin, M. Kassim, K. Othman, and M. Samad, "Incorporating VHDL in teaching combinational logic circuit," in *Proc. 2nd Int. Congr. Eng. Educ.*, 2010, pp. 225–228.
[6] Y. Zhu, T. Weng, and C. Cheng, "Enhancing learning effectiveness in digital design courses through the use of programmable logic boards," *IEEE Trans. Educ.*, vol. 52, no. 1, pp. 151–156, Feb. 2009.
[7] A. Zemva, A. Trost, and B. Zajc, "A rapid prototyping environment for teaching digital logic design," *IEEE Trans. Educ.*, vol. 41, no. 4, p. 342, Nov. 1998.
[8] Z. Stanisavljevic, V. Pavlovic, B. Nikolic, and J. Djordjevic, "Sdldssystem for digital logic design and simulation," *IEEE Trans. Educ.*, vol. 56, no. 2, pp. 235–245, May 2013.
[9] C. Hacker and R. Sitte, "Interactive teaching of elementary digital logic design with winlogilab," *IEEE Trans. Educ.*, vol. 47, no. 2, pp. 196–203, May 2004.
[10] N. Hendrich, "A java-based framework for simulation and teaching: Hades-the hamburg design system," in *Proc. Microelectron. Educ.*, 2000, pp. 285–288.

[11] N. H. Yusof and R. Hassan, "Flash notes and easy electronic software (EES): New technique to improve digital logic design learning," in *Proc. IEEE Int. Conf. Elect. Eng. Inf.*, 2011, pp. 1–6.
[12] Chipmunk Documentation. (2014, Sep. 19). [Online]. Available: http://www.cs.berkeley.edu/~lazzaro/chipmunk/describe/log.html
[13] C. Burch, "Logisim: A graphical system for logic circuit design and simulation," *J. Educ. Resources Comput.*, vol. 2, no. 1, pp. 5–16, 2002.
[14] S. Kubisch, R. Rennert, H. Pfueller, and D. Timmermann, "Logengeneration and simulation of digital logic on the gate-level via internet," in *Proc. 1st IEEE Int. Conf. E-Learn. Ind. Electron.*, 2006, pp. 46–51.
[15] Smartsim. (2014, Sep. 19). [Online]. Available: http://smartsim.org.uk/downloads/manual/smartsim_user_manual.pdf
[16] Easysim. (2014, Sep. 19). [Online]. Available: http://www.research-systems.com/easysim/easyman.pdf
[17] M. Tudoreanu and E. Kraemer, "Balanced cognitive load significantly improves the effectiveness of algorithm animation as a problem-solving tool," *J. Vis. Lang. Comput.*, vol. 19, no. 5, pp. 598–616, 2008.
[18] C. Hundhausen, S. Douglas, and J. Stasko, "A meta-study of algorithm visualization effectiveness," *J. Vis. Lang. Comput.*, vol. 13, no. 3, pp. 259–290, 2002.

Abdulhadi Shoufan received the Dr-Ing degree from the Technische Universitt Darmstadt, Germany, in 2007. Currently, he is an assistant professor of information security and ECE at Khalifa University, UAE, and he leads the Security Hardware Group at the Center for Advanced Security Research Darmstadt (CASED), Germany. His research areas include embedded security and cryptographic hardware as well as engineering education.

Zheng Lu received the BE degree from Beijing Union University in 2001, and the MSc degree from Technische Universität Darmstadt, Germany in 2009. Since 2010, he is working toward the PhD degree at Technische Universität Darmstadt. His research areas include digital logic design process and web-based interactional education platform.

Sorin A. Huss received the Dr-Ing degree in electrical engineering from Technische Universität München, Germany, in 1982. He worked in the industry from 1982 until 1990 in different positions at AEG Aktiengesellschaft in Ulm, Germany. Since 1990, he has been a full professor in the Computer Science Department of Technische Universität Darmstadt, Germany. He was one of the founders and directors of the CASED Center for Advance Security Research Darmstadt. His current research interests are in the areas of embedded system design methodology, HW/SW architectures for IT security, and car-to-car communication systems.