Solving Digital Logic Assignments with Automatic Verification in SCORM Modules

Markus Damm, Friedrich Bauer, Gerhard Zucker

Vienna University of Technology, Institute of Computer Technology

Key words: *e-learning, virtual lab, automatic task evaluation, SCORM*

Abstract:

This paper presents an application based on Adobe Flash and SCORM within a MOODLE learning environment, which provides individualized assignments for students who learn digital systems. The assignments are evaluated automatically and the result is reported to the MOODLE platform. Since the tasks are solved by the students remotely and unattended, each student needs to get his or her personalized assignment. This is done by selecting one assignment from the pool of available assignments based on the student's unique learner id.

1 Introduction

Teaching digital logic and sequential circuits to bachelor students of electrical engineering has been a long term mission at Vienna University of Technology. For several decades a combination of lecture and tutorial has run successfully and was refined throughout the years. Students were, however, dissatisfied with proper possibilities to practise outside the courses. Of course assignments (including solutions) were provided to students, but it appeared necessary to integrate an improved way of interactive learning. Today the course is augmented with a full-fledged SCORM e-learning module [1] which provides individualized examples that are automatically tested for correctness. The results are taken over from the Moodle e-learning platform [2], which is employed by Vienna University of Technology for all e-learning activities. The core of the SCORM module is LogiFlash [3], an simulator for digital logic and sequential circuits written in Adobe Flash. To allow for cooperation with the Moodle platform additional glue code had to be designed, thus enabling fully automated evaluation of student assignments. The effort for supervisors is nearly independent of the number of students in the course. Since assignments can usually not be created fully automated it is hard to create personalized assignments for more than 200 students. Therefore the SCORM module is able to map a limited pool of assignments to any number of students. Nevertheless, access to other assignments is disallowed, thus the risk of students working on the same assignment is reduced.

An additional tool that students can use to solve their assignments is a state graph, also written in Adobe Flash, which supports state machines when solving assignments with sequential logic.

This paper is organized as follows: In Section 2, we provide information about assignments that were employed in the course and solved by the students; section 3 explains the setup and main components of the SCORM module overall; section 4 sums up the paper and takes a look into future activities.

2 Assignments and test patterns

One class of assignments deals with a finite state machine according to the theoretical background taught in the accompanying first year's lecture. A certain bit sequence on two input lines must be detected, and an output pin has to be set when the bit sequence has been detected successfully. Figure 1 shows as an example the detection of the bit sequence E_1E_0=10,01,01, which results in setting the output A to 1, whereas A stays 0 otherwise. To solve this problem, the student has to transfer the problem into a finite state machine, set up a state encoding, compute the (minimized) state transition and output functions, and finally realize and test the resulting circuitry with LogiFlash. That is, the student experiences all stages of circuit design and synthesis using a straightforward example.

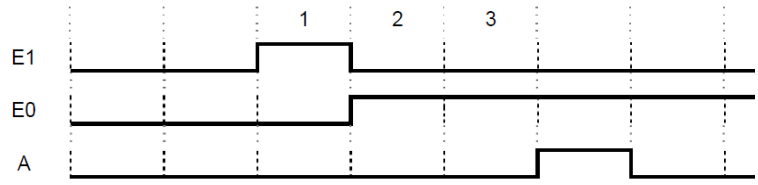

Figure 1: Example input-output sequence

This type of base task allows us to easily create a large number of variations of the assignments in the pool by simply changing the significant bit streams and the level of the output pin. Thus, we get many different assignments of the same complexity level (e.g. the number of states required is always 4), which is important to ensure equal treatment of all students.

To verify the correctness of the student's solution, a test pattern has to be generated. Since the student can use any state encoding of his liking, the test pattern can only involve the input bits and the output bit. For the same reason, we cannot assume the automaton under test to be in the start state when resetting the state memory to 0, since this might not be the encoding of the start state. Therefore, the test pattern starts with an input sequence which sets the automaton to the start state, provided it is constructed correctly. For this initial test pattern, the resulting output is ignored. After that, the test pattern consists of input-output pairs, where at each step the test pattern input is applied to the automaton and it is verified if its output matches the output of the test pattern. The test pattern is designed such that it includes test paths through all state transitions of the automaton. This approach gives the student freedom of how to solve the task with respect to state encoding and logic minimization, since only the input-output behaviour is tested. This leads to an automatic evaluation of logic circuits beyond the usual "multiple choice" techniques.

This evaluation is done by comparing the behaviour of the automaton to specific test patterns. Since all problems have a similar structure, the test patterns that allow the software to verify the correctness of the solution (see section 3.2) can be generated nearly automatically and do not have to be entered manually by teaching staff.

3 Components and Technical Realization

This section describes the components used within the SCORM module as well as its overall design. Especially, it is explained how to make individual task assignments in Subsection 3.3.

3.1 State Graph Construction

The state graph is a tool that has been designed to give students additional support to solve their assignments. It shows three different views of the state machine simultaneously: the state diagram, the truth table and the animated block circuit.

The students can use this tool to draw the state graph easily and intuitively. Since all assignments have four states, the students do not need to draw the states but only the transitions between them. To add or modify a transition, the students only need to click on the transition's position and then can select among adding/modifying/removing this transition for any specific combination of the inputs E_1E_0.

Each transition has a corresponding entry in the truth table (e.g. one column). When modifying any transition in the state diagram, the truth table is updated automatically. If no transition exists yet, the truth table's column remains filled with "X" (Don't Care) in red color. This makes it very easy to see if any entries are still missing.

On the other hand, the truth table itself is also editable by simply clicking on the entry that has to be changed. If any entry is modified, the state diagram is updated automatically. So the truth table and the state diagram contain consistent data at all times.

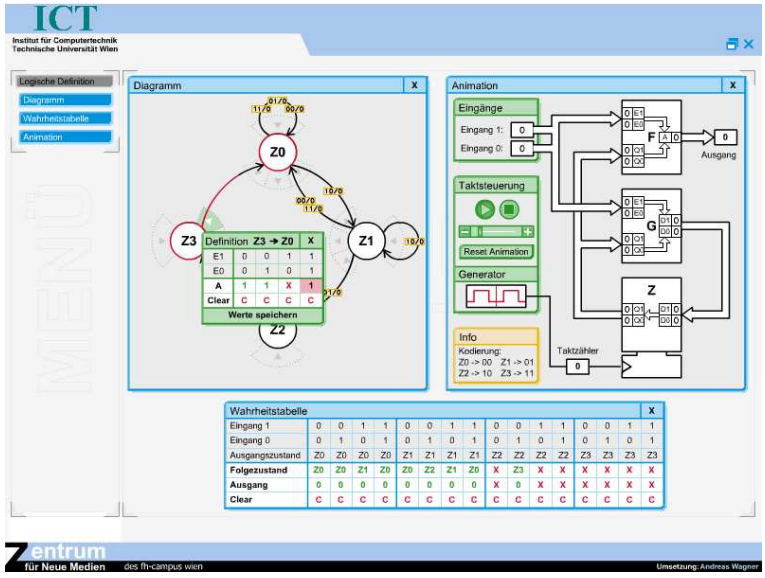

Figure 2: state graph construction tool

It makes sense to start the animated block circuit after completing the state diagram and truth table. A virtual clock generator is started (the frequency is adjustable), forcing the state machine to switch to a new state at every positive clock edge. Then, the input variables can be set to 0 or 1, respectively. Depending on the old state and the values of the input variables, the new state is prepared which can be watched at the animated block circuit. If the truth table has not been filled out completely, it might happen that the next state cannot be calculated correctly. In this case, the animated block circuit will provide the user with an exception message. A clock edge counter completes the animated block diagram, showing the exact number of transitions performed.

3.2 LogiFlash

LogiFlash [3, 4] is a graphical logic simulator implemented with Adobe Flash. The user can construct arbitrary logic circuits out of several stateless logic gates (e.g. AND-gates, multiplexers, adders) and stateful components (e.g. flip flops, registers), and simulate them. Also, it is possible to construct custom components out of circuits. Circuits can be stored in a custom XML format. LogiFlash offers also a simple API to be used within other Flash applications, e.g. to couple the simulation of a circuit with the animation of the corresponding KV-diagram or state graph. LogiFlash development started at the department for Computer Science at the Goethe University, Frankfurt-Main/GER and continues today at the Vienna University of Technology.

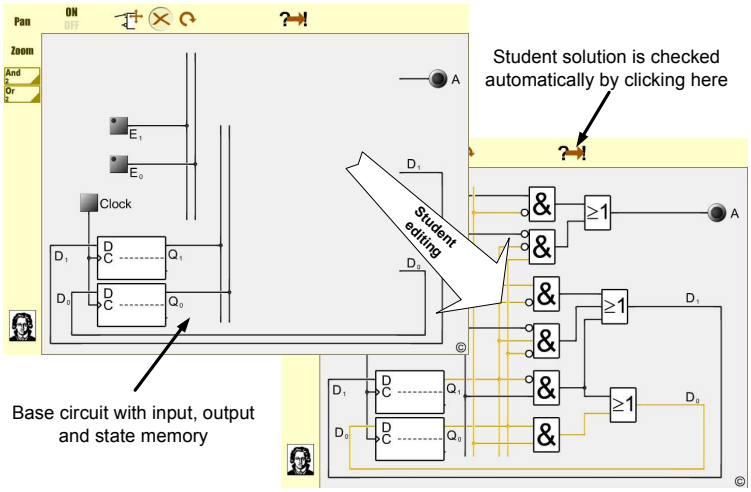

Figure 3: Constructing and checking circuits with LogiFlash

Among its many features, a certain ability is most valuable for e-learning applications: It is possible to define and process test patterns to verify automatically if a circuit has a certain desired behavior. For combinatorial circuits, this comes down to matching the circuit's beha-

vior against a table of outputs for all possible input combinations. For sequential circuits, this is more complex, as already described in Section 2. Fortunately, LogiFlash is capable of processing also these more complex test patterns, including an initial stage where the output is ignored. In LogiFlash, test patterns are stored together with circuits as XML. For the case at hand, the test pattern were stored together with a base circuit consisting of the state memory, the inputs and the output (see Figure 3).

For the tasks to solve by the students, only AND and OR gates were allowed. Therefore we made a modified version of LogiFlash with restricted features which resulted in a simplified user interface (see Figure 3). Also, additional code had to be added for the SCORM integration.

3.3 Architecture of the SCORM Module

The SCORM module is divided into four parts (see Figure 4): After a general introduction, a flash module (the task description loader) reads the student's learner id and loads, by applying a hash function on the id, one out of 20 different PDF files containing different task description. Note that these concealment measurements would have not been necessary if the loader would simply load one out of 20 HTML pages, which is also possible, since in this case the file name of the respective page is hidden. We decided, however, to use PDF files anyway since they provide other advantages, e.g. better printability.

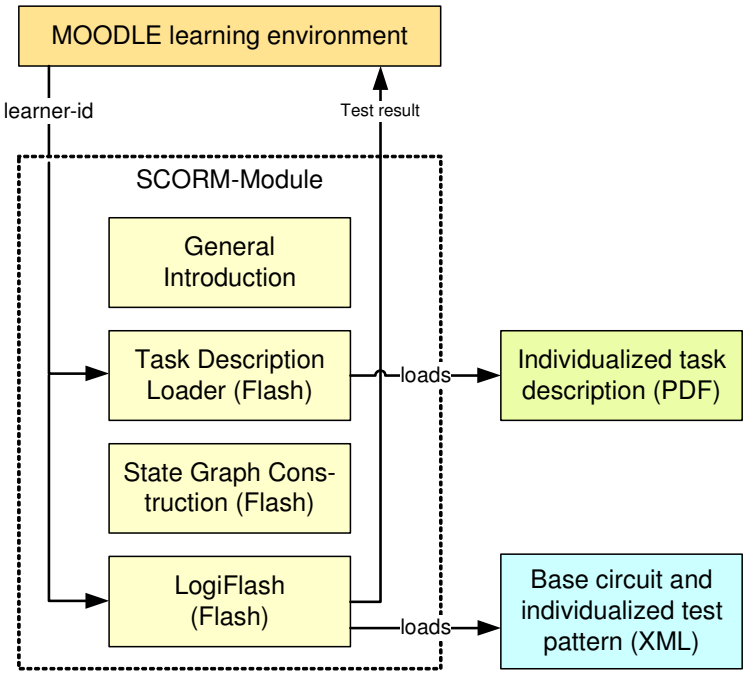

Figure 4: The SCORM module

This loading mechanism is the only task of this flash module, since individualized tasks are not foreseen by the SCORM standard per se. Note that this part could have been implemented by other means, e.g. using JavaScript. Flash was only chosen since we used it extensively anyway.

Since we did not want the students to easily find out if they have the same assignments, and since they might learn the name of the PDF file (depending on the browser used), we made about 30 versions of the same file with random names, using another hash function. Therefore, it appeared to the students as if they had truly individual assignments.

The third part is the state graph construction tool described in Section 3.1. This part is not dependant on the learner id, and also not essential to solve the task described in the assignment, but is intended to provide a help to the students. If the tool is not used, the students basically have to perform the same actions with paper and pencil.

The forth part is the LogiFlash module for solving the assignment, which now loads the general base circuit for the task together with the appropriate test pattern fitting to the individual assignment. In this case, the students do not learn the respective file names of the XML files loaded, therefore there is no need for concealment like it was with the PDF files.

If the automatic task verification process evaluates the solution to be correct, the completion status of the module is set to "completed". For the communication between Flash and SCORM, the pipwerks ActionsScript 2.0 SCORM API wrapper was used [5]. Since the

MOODLE platform at the Vienna University of Technology is directly coupled with the university's electronic grading system, no further actions by the teacher are required.

4 Conclusion and future work

This paper presents a novel approach to use individualized and automatically verifiable tasks in teaching for technical computer science. While the type of task presented is standard in computer science teaching, the individualized presentation and the involved automaticity is very innovative, and was rewarded with the E-learning Award 2009 from the Vienna University of Technology's E-learning Centre [6]. Recently the development of LogiFlash has been made accessible to the public by introducing LogiFlash as an Open Source project at [7]. It is expected to draw attention to the project by many Flash programmers who help to extend the current functionality of the application.

References:

[1] Advanced Distributed Learning: SCORM Overview. http://www.adlnet.gov
[2] http://moodle.org/
[3] Damm, M., Klauer, B. & Waldschmidt, K. (2003). LogiFlash - A Flash-based Logic-Simulator for educational Purposes. In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2003 (pp. 748-750).
[4] LogiFlash Examples and Download: http://www.ti.cs.uni-frankfurt.de/wwr/
[5] http://pipwerks.com/lab/scorm/
[6] http://elearning.tuwien.ac.at/index.php?id=442
[7] http://logiflash.sourceforge.net/

Author(s):

Markus Damm, Dipl. Math.
Friedrich Bauer, Dr. techn.
Gerhard Zucker, Dr. techn.
Vienna University of Technology,
Institute of Computer Technology
Gußhausstraße 27-29/384
1040 Vienna
{dam|bauer|zucker}@ict.tuwien.ac.at