Abstract
This work describes how the CTPracticals Moodle module can be used for e-assessment in an introductory course on computer organization, where the practical content consists of the design and simulation of a basic CPU implemented using Logisim, a schematic-based educational tool for the design and simulation of digital circuits. A previous work extended this module to support the verification of codes written in Matlab. This work shows how this feature can be exploited to invoke external tools, i.e. Logisim, not only for the automatic verification of the student submissions, but also for a detailed analysis of the results that improves the assessment. The paper will also present some customization of Logisim that was necessary to improve its batch mode simulation in order to add some useful options that make it generate a richer output format.

Keywords:
Automatic assessment; computer organization; Logisim; Moodle; CTPracticals

1. Introduction
During this last decade, Learning Management Systems (LMS), and in general new technologies, have been adopted very quickly by higher education, and even are identified as a way of reforming it [1]. LMS systems have changed several aspects of the teaching-learning process, advancing towards a more interactive learning, encouraging constructivist pedagogical approaches [2, 3], self-learning, collaborative learning and teamwork [4, 5] which are increasingly valuable skills in the social network era. Furthermore, from the point of view of teachers and instructors, LMSs are helpful to gain tidiness and order while pedagogical activities are packaged in a uniform way across courses.

In the context of Engineering and Computer Science teaching, the CTPracticals module joins together two elements: LMSs, particularly Moodle [6], and the automatic verification of programming-like assignments. Motivations for choosing Moodle (Modular Object-Oriented Dynamic Learning Environment) include its popularity, its modular organization and its free availability. Most present-day educators are used to working with LMSs, platforms that by means of web-based interfaces provide a panoply of activities (forums, chats, surveys, ...) with which a standard course can be set up. Both teachers and students can harness the integration of automatic assessment features into this kind of platforms. Teachers can better organize the lab works, and give a better use...
to the unproductive time spent in checking manually practical assignments, specially in crowded classes [7]. In turn, students benefit from a faster response and they feel more comfortable with an environment they are already familiar with. Automatic assessment and verification of lab assignments has been addressed in numerous works found in the literature [8, 9], including high level language programming [10, 11, 12, 13] and lower levels or even hardware [14, 15].

This work is focused on the usage of the CTPracticals module in the practical component of a basic subject on computer organization taught in the first course of the Computer Science degree, whose lab work consists of the design and simulation of a basic CPU. This is our study case, but there are multiple introductory courses in the Computer Science and Engineering curricula where students have to face the design and simulation of digital circuits and elementary processors. Two topics of discussion about how to teach digital design in first courses are the use of academic versus commercial tools and the use of hardware description languages (e.g. VHDL) versus schematic-based design [16, 17]. Although using commercial design tools with hardware description languages is clearly the best option for high level courses and professional profiles, in authors’ experience this approach may distract students from their learning target for an introductory course, taking into account that current commercial tools are quite complex and not easy neither to learn or to use.

Looking for a simple tool, with a fast learning curve but also complex enough to design simple processors, we have moved to Logisim, a schematic-based system for logic circuit design and functional simulation [18, 19] developed for educational purposes. Insomuch as Logisim can provide textual output in batch mode, it can be invoked directly from one of the verification script languages, TCL or Matlab, supported by CTPracticals. Particularly, Matlab has been preferred, given its good features for data processing and analysis and its simpler syntax. Additionally, as described in next sections, some customization of Logisim has been carried out to improve the Logisim batch mode simulation, adding some useful options and making it generate a richer output format.

This paper is organized as follows. Next section briefly introduces the CTPracticals Moodle module and third section will provide an overview of the course and its practical assignments. Section four will present the improvements incorporated into the program Logisim. Then, section five will describe how the Matlab testers are designed for the automatic assessment of the previously described lab-works. And finally, the two last sections provide, respectively, some results of the experience and brief conclusions.

2. CTPracticals Moodle module

CTPracticals is an e-assessment solution oriented to automatically verify programming-like assignments, which has been developed as an activity of the popular LMS Moodle. It exhibits features and functionality comparable or better than other systems for the automatic assessment of programming assignments [9, 20]. Originally, CTPracticals was created to give support to assignments consisting of VHDL projects, but the module can be adapted to other programming languages [21]. In particular, the module was extended to support Matlab [22], a highly adaptable language and also widely known in the scientific and engineering environments1.

From the developer’s viewpoint, it is feasible to extend the module to other languages apart from VHDL and Matlab. Nevertheless the door is opened to use Matlab as a scripting language for testing. Not only it allows to invoke external tools, but to also analyze the verification results at a quite detailed level. Therefore, the module can be used to verify and assess student projects created with tools like Logisim, as long as such a tool provides its output in a suitable format. Extra modifications of the module are not necessary: by invoking Java as an external application from the Matlab script, Logisim is executed2.

Though next lines in this section will briefly review the main interfaces and functionality of the module, the reader is addressed to [21] and the demonstration site for further details (see section 8). CTPracticals is written in PHP and it is offered as a Moodle module that can be installed in a standard linux-based Moodle

---

1GNU Octave, a Matlab free clone, is also supported.

2In addition to some security mechanisms implemented by CTPracticals, like chroot-ed executions, Matlab scripts are preprocessed to avoid malicious behaviors, avoiding the execution of external commands. An exception to that limitation needs to be introduced to allow executing Logisim through Java. Nevertheless, the command line invoking Logisim needs to match certain regular expression in order to prevent code injection vulnerabilities.
platform. CTPracticals allows users to define activity instances in the course sections, like other Moodle modules (assignments, forums, questionnaires, ...) and its configuration aspects can be controlled through a control block on a side panel of the course page (Fig. 1). By clicking each instance, the user can access the activity itself, whose behaviour varies from students to teachers. The module makes use of three basic objects, that can be managed by means of three tabs on the teacher interface (Fig. 2), the Practical Assignments, the Testers, and the Verifications:

**Practical assignments:** Defined by the teacher, each activity may contain one or several assignments, for which each student team must submit a zip file containing its lab work. Typical attributes of a practical includes: deadline, required files, filename constrains (expressed as a function of the team id), visibility of results, and several others.

**Testers:** Designed by the teacher, the tester defines the key aspects of the verification. It includes the command file containing the verification script, the result file with the reference correct output, auxiliary files used during the verification process and several other control options. The verification script is responsible for launching the code submitted by the students. It may either simply generate an output that is later compared with the correct one, or make a more complex analysis of this output in order to get a finer grading. The teacher will have to decide the tester complexity and the amount of information provided to students. Fig. 2(d) shows the panel for editing every attribute of a tester. A tester script template is sketched in Fig. 3 together with the execution flow of the verification process. The tester attributes in Fig. 2 are related to their use in the script of Fig. 3 by means of labels.

**Verifications:** Launched at the moment of submission, or later by the teacher in batch mode, one verification comprises all the outputs/files generated by the application of a certain tester, together with other important assessment elements. These are: format errors for the submission, the result itself (OK/Wrong), the grade, a textual short feedback, and a detailed verification log. Statistics are also gathered during this process.

As refers to the CTPracticals student interface (Fig. 1), in addition to submitting the works, students can browse information about their practical assignments, like their specifications, deadlines, etc. and the most valuable information: the results of the automatic verification and, consequently, their assessment. As a remarkable feature, the module organizes students in teams, each of which has a unique identifier (team name). This allows a true team work, unlike common Moodle assignments which are submitted and graded on a per-user basis. The team creation process is very interactive and social, because students must negotiate the teams using an invitation system provided by the module. Moreover, forcing the use of team name for defining filenames can mitigate plagiarism (stage ➀ of the template script in Fig. 3).

3. Overview of the course

Computer Technology is a basic course on computer organization delivered in the Computer Science academic plan of the University of Málaga (Spain). The objective of the course is to introduce how the CPU of a computer system works at the register transfer level, covering the gap between the machine level and the assembler language. To this end, the laboratory assignments consist of the design, implementation and simulation of a basic monocycle implementation of the MIPS processor [23].

The chosen tool is Logisim [19], an educational tool for the design and simulation of digital circuits. Licensed under the GNU GPL, Logisim is an open-source and multi-platform application developed in Java. It is also very lightweight, so it can be used by the students even in their underperforming laptops. Logisim projects are stored in XML format so that the storage requirements are low both in the user and the server side. The tool is simple enough to have a quick learning curve, but complex enough to allow the design of a simple processor:

- It includes most of the basic digital circuit elements: inputs, outputs, gates, multiplexors, arithmetic circuits, memory elements, etc. These are both easy to create and modify.

---

3Moodle is developed over the LAMP solution stack: Linux, Apache HTTP server, mysql and PHP.

4Note that although Moodle defines the so called groups, these are used only with organizational purposes (e.g. classrooms).

5Logisim 2.7.1, released in 2011. It requires Java 5 or later.
Fig. 1. (a) A CTPracticals activity in a typical Moodle course page. By clicking the activity, students can browse and submit their work in a zip file (b). The module has associated two side blocks: one for configuration (left) and another with a ranking of teams (right).

- It allows the development of hierarchical designs (top-down methodologies).
- The circuit can be verified all along its implementation (for example, mismatched sizes of buses and other connection errors are reported).

The students lab work consists of three practical assignments. The final score is calculated by weighing each one with 40%, 30% y 30% respectively. These assignments are:

1. Designing a basic CPU. The students will implement a mono cycled MIPS [23], with a reduced instruction set: J, BEQ, ALU, LW, SW. The student is provided with a full specification of the design, the main objective of the assessment being to get acquainted with the simulation tool (Logisim) and how to use it for the implementation, programming and simulation of the processor. The final score is obtained after the execution, on the student’s implementation, of a teacher’s designed program, that will test all the instruction set.

2. Extending the original instruction set. Students will introduce two new instructions: load and store with a relative auto-incremented addressing. The only specification is the behaviour of the instructions. Now, they will have to modify their datapath in order for the new instructions to execute correctly. Besides, the students delivery will include a simple program making use of them: it will have to copy 6 source data memory positions to 6 destination data memory positions. The addresses of both sources and destinations are specified by the teacher. Both the hardware implementation and the program must be correct in order for the lab work to be considered a right one.

3. Optimizing an assembler program with optional further extension of the instruction set. The proposed algorithm is the computation of the sum of the absolute value of the differences between the corresponding elements of two vectors (SAD), operation that is used to measure their similarity. Students will perceive the difficulty of writing this code, though easy, when the instruction set is so reduced, and they will have the freedom of implementing new instructions looking for the best performance. In fact, the grade (from 0 to 10) depends on the total execution time ($T_{EX}$) of the program:

$$grade = \begin{cases} 
0.0, & \text{if there are format errors or the execution result is WRONG} \\
5.0, & \text{if the result is OK and } T_{EX} \geq T_{MAX} \\
10.0, & \text{if the result is OK and } T_{EX} \leq T_{MIN} \\
5.0 + \frac{T_{MAX} - T_{EX}}{T_{MAX} - T_{MIN}} \times 5 & \text{if the result is OK and } T_{MIN} < T_{EX} < T_{MAX}, 
\end{cases}$$

where $T_{MAX}$ (max. execution time) and $T_{MIN}$ (min. execution time) are thresholds defined by the teacher.
An activity instance of the CTPracticals module has been created to allow the automatic verification and scoring of each of the three practical assignments. It appears in the course page together with other materials, such as documents, templates and downloads (see sample course in Fig. 1).

4. Tailoring Logisim to our needs

As aforementioned, Logisim can be invoked from the command line generating a textual output, which makes possible the design of Matlab scripts for the automatic checking of the student’s work. However, certain features of the original Logisim were not well suited for automatic verification from CTPracticals. As Logisim is freely available, it was possible tailoring the source code to our needs. Some useful options were added to make Logisim generate a richer output format:

1. In contrast to Logisim’s native command output, which is very plain (only rows with pin value changes), the new enhanced format contributes valuable data for subsequent analysis: full hierarchy information, signal radix and width, circuit names, and time information (tick count) (see Fig. 3(c)).

2. The simulation results should contain not only the evolution in time of the circuit pins and probes, but also the contents of user specified memory positions. This is important when verifying a program execution (the program writes in the data memory). To inspect memory positions 0, 1, 2, 20, 21 and 22, you may use, when invoked in command line mode, the new parameter:

   \[-\text{ram} \hspace{0.5em} \text{[memory locations to show comma separated]}\]

   For example:

   \[-\text{ram} \hspace{0.5em} 0,1,2,20,21,22\]
Fig. 3. Designing a tester: (a) Verification flow; (b) Skeleton for a verification script; (c) Log output (summarized) of a real execution. The different stages in the script depend on the attributes defined in the tester (Fig. 2). Teacher can perform further checking before launching Logisim, and make some processing on the submitted circuit. After simulation the script must write results according to some CTPracticals conventions (CTP_ prefixed functions in (a)). The log example (c) shows the new output format incorporated into original Logisim: it includes a first column with the clock tick count and allows showing pin and probes recursively as well as RAM locations.

3. In order to simplify the analysis of the results during the verification stage, two new parameters allow to select which pins/probes will appear in the simulation results: -noprobe, which disables specified probes and -nopin, disabling the specified circuit pins.

4. Control of the simulation length is required. Originally, Logisim simulations stop after the activation of the one bit output halt: when performing an unattended simulation, a student error would cause a malfunction of the verification system. The new parameter, -stopat allows to specify the simulation length as a maximum number of clock ticks.

5. To be able of initialize RAM contents. It is important if we want to verify whether the student’s CPU implementation executes the teacher’s test program, and/or the teacher’s data inputs. Logisim, on circuit load or after a reset, always initializes RAM contents to zero. The new behaviour is achieved by associating to RAM memory a text file with the same format used by Logisim for ROM memory, such as:

```
0 5 4 ffffffff ffffffff 3 2 1
```

On circuit reset, and when the simulation is launched on the command line invocation, the file contents are copied to the associated RAM memory. This implied an additional format modification for the .circ files containing the circuit description. A new text field, “Datafile on reset”, will contain the name of the initialization file for each RAM module:

```
<comp lib="4" loc="(220,150)" name="RAM">
  <a name="Datafile on reset" val="datafile.txt"/>
</comp>
```

Also, changes on the Logisim graphic interfaces were required: a new field when visualizing the RAM module properties allows to specify the initialization file (as shown in Fig. 4).
Fig. 3 shows the log results after a tester execution. The output (Logisim_simulation_output=) was obtained using our modified Logisim version (logisim-dac1.1.jar), invoked as:

```
java -jar logisim-dac1.1.jar TestCircuit.circ -tty table -noprobe -nopin -ram 0 -stopat 1000
```

where TestCircuit.circ is the circuit description file, and parameter -tty table is for a command line simulation. Due to parameters -noprobe and -nopin, the simulation output will not contain any probes or circuit outputs, and parameter -ram 0 will include in the output the contents of memory position 0. Due to -stopat 1000 the simulation ends after 1000 clock ticks.

It is also worth mentioning that in addition to the described specific output format oriented to facilitate the automatic verification, Logisim was modified to generate Value Change Dump files (VCD), as it is shown in Fig. 4. This feature allows Logisim to be combined with Gtkwave [24], a free fully featured wave viewer, which completes the student experience with a graphical depiction of the circuit simulations.

5. Using the CTPрактичные модуль to verify the assignments

A tester is but a MATLAB script that should perform the following tasks (Fig. 3):

1. Initial checking:
   - checking the submitted Logisim file (.circ): it should have been implemented using the enhanced version of Logisim. A content search is performed inside the .circ file (a XML-like one) looking for the strings of the names of the files specifying the RAM memory contents. If not found, testing is aborted and an informative error message is produced inside the log file. See 2 in Figs. 3 and 2.
   - when required, the contents of the data and/or instruction memories are changed so that the test data/program designed by the teacher will be used/executed: the corresponding strings are changed in
the .circ file. These files were uploaded by the teacher when creating the tester as auxiliary files. See ➂ in Figs. 3 and 2.

2. Logisim program is invoked with the appropriate parameters for the simulation and collection of the output data required for the future verification and assessment. If execution fails (error code when executing Logisim) this will be reported as a format error in the log file, as well as a copy of the Logisim output. See ➄ in Fig. 3.

3. The output data is corrected comparing against a reference one and the grade is automatically produced. See ➃ and ➅ in Figs. 3 and 2.

Now, we will describe in further detail the particularities of each of the three Logisim testers designed for the course:

5.1. Tester for the practical assignment 1

The content of the instruction memory is substituted with a teacher’s designed program that will test all the specified instructions. But also, it is designed in order to produce enough output information as to allow the automatic tester to inform of which is the incorrect instruction: between every two tested instructions, the program stores in the location 0 of the data memory the last computed value. Logisim will be invoked so that the evolution in time of this 0 memory position will be shown as a part of its output. This output will be compared against the reference one: the log file will inform of which are the incorrect instructions and the grade will be automatically produced as a function of the number of errors. A fragment of a log file corresponding to an incorrect lab-work is shown in 3(c).

5.2. Tester for the practical assignment 2

In this case, none of the contents of the memories are modified. Logisim is invoked so that the contents of the 6 source data memory positions and the 6 destination ones are shown as a part of its output. The grade will be determined after the comparison of the source positions content with the destination ones.

5.3. Tester for the practical assignment 3

As part of the specification, the student program should read the vectors size and their location in memory from fixed memory locations. This makes it possible to change the contents of the data memory, and thus, to use teacher’s defined test vectors. The instruction memory is not changed, as students are encouraged to extend the instruction set searching for a reduction of the execution time of their programs. Logisim will show the contents of the data memory position where the program output will be stored, that will be compared against the reference one. As the Logisim output also includes the cycle count for each reported change (see section 4), when the script finds the correct content in the output memory position, this number will be used as the execution time in order to automatically compute the grade as described in section 3.

As a way of promoting competitiveness, the CTPracticals ranking block has been used (Fig. 1(c)). The ranking orders the student teams according to their grades in the different assignments. The objective is to instill excellence: students will try to improve their CPU by designing new instructions looking for a better performance.

6. Students experience

Several have been the facets brought into play by using CTPracticals as e-assessment tool integrated in Moodle in the dynamic of a Computer Technology laboratory. First, it allows students to work cooperatively in teams. Second, it helps to carry out competitive learning strategies, as it happens with the third practical assignment previously described. For this assignment, a progressive grading and the possibility of multiple retries made teams contend for getting higher marks. This competitive approach would be difficult to implement without an automatic verification tool. Finally, the CTPracticals infrastructure has made easier to move the lab work to a lightweight, full featured and portable digital design tool like Logisim. Its ease of use was highly appreciated by students.
Tables 1 and 2 summarize some figures about the experience. About 3300 submissions were performed by 239 students distributed into five classes. Fig. 5 depicts a breakdown of the results. The inverse bathtub curve of the wrong submissions can be explained because students need to get used to the tools when they start and the specification complexity is slightly increasing. Practical assignment 3 is of special interest. It was very motivating for students, as they could know their position in the ranking. It only showed the team identifiers (not students’ names), giving certain anonymity to prevent possible negative effects. Data from table 2 and Fig. 6 suggest the good students’ attitude towards improving their grade. They faced optional lab work despite the baseline design was the only mandatory to pass this assignment, and the pressure they usually have at the end of the semester. Teachers observations agree that the automatic assessment system has acted as motivation, by providing an effective feedback mechanism that gives promptness and dynamism to the lab work.

| # Students | 239 |
| # Registered teams | 150 |
| # Teams submitting all assignments | 93 (62%) |
| # Teams passing 3rd assignment (grade ≥ 50%) | 78 (52%) |
| # Submissions passing 3rd assignment | 407 |
| # Trials per team for 3rd assignment (avg.) | 4.3 |
| % Teams increasing its grade w.r.t. its first submission | 43.6% |
| Improvement of grade (avg.) | 26.6% |

Table 1. Some figures of the experience for the academic course 2011/12.

<table>
<thead>
<tr>
<th>Number of submissions</th>
<th>Number of teams (% of the total)</th>
<th>% of teams that improve grade</th>
<th>Improvement of grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24 (30.7%)</td>
<td>0 (baseline)</td>
<td>0 (baseline)</td>
</tr>
<tr>
<td>2-3</td>
<td>19 (24.4%)</td>
<td>26.3%</td>
<td>11.9%</td>
</tr>
<tr>
<td>&gt;3</td>
<td>35 (44.9%)</td>
<td>82.9%</td>
<td>29.1%</td>
</tr>
<tr>
<td>Total</td>
<td>78 (100%)</td>
<td>43.6%</td>
<td>26.6%</td>
</tr>
</tbody>
</table>

Table 2. Grade evolution of 3rd practical assignment (processor design) for those teams passing this assignment.

7. Conclusions

This paper has described the application of the Moodle module CTPracticals, developed by the authors, to a computer organization laboratory taught in the first course of a Computer Science degree. Such a module provided Moodle with a new activity class that allows the automatic verification and e-assessment of VHDL/Matlab assignments, as well as an enhanced management of such verifications and other useful features like team work support. As it is an introductory course, Logisim, a simple but full featured educational tool for designing and simulating digital circuit, has been used. With no deeper modification, the existing support in CTPracticals can be used to invoke this simulator, through the existing verification engine for Matlab assignments. The simple syntax and powerful capabilities of Matlab help teachers to design suitable testers and verification scripts. However it was necessary to add some new features to the original Logisim program in order to make feasible the automatic verification process and its use friendlier. As a way of motivating the students, the CTPracticals module has been complemented with a new side block, a ranking that lists the team names ordered by their score. The experience has been very satisfactory for students and teachers, making possible combining team work and soft competitive approaches and bringing a learning-2.0 style to the practical classes.
8. Demonstration site and acknowledgements

A demonstration web site can be visited at: http://guac.ac.uma.es/demo (User: demoteacher, password: 1234). This work has been partially supported by the educative innovation projects PIE08-062 and PIE10-140 granted by the University of Málaga.

References


