Experimenting with Web-Based, Personalized Homework Assignments

Daniel H. Linder
Mississippi State University

Abstract

In our senior-level computer architecture course at Mississippi State University, we are experimenting with using a Web interface to distribute, debug, and collect VHDL programming assignments which are unique to each student. Using a standard Web browser, students interact via forms with a remote computer that runs their VHDL simulations, checks the results, and returns debugging information in the form of HTML tables. This presents a simpler interface to the students than the full VHDL CAD tools and eliminates issues associated with tool distribution and licensing. The personalized nature of the assignments also helps ensure that all students work through the assignments in detail. We have found that the fixed format of the returned simulation results does not interfere with code debugging and that the immediate feedback to the students on the success or failure of their simulations strongly motivates them to work until their simulations are correct.

I. Introduction

The growing importance of hardware description languages such as VHDL for the simulation and synthesis of modern digital systems makes these languages an essential component of a computer engineering curriculum. However, programming assignments in VHDL or other exercises that use CAD tools can be difficult to add to a course for several reasons. CAD tools often have complex interfaces that take time to learn. An assignment using a CAD tool can quickly turn into a struggle with tool usage rather than a reinforcement of lecture concepts. Licensing can be another problem. Many students prefer to work on their own PCs instead of coming to a centralized lab, but CAD tool vendors may be reluctant to make an expensive tool available on all student machines. Finally, if every student has the same assignment, interactions among students can degenerate from discussing concepts to merely exchanging details of how to get the assignment working. When this happens, many students obtain only a superficial understanding of the issues involved in the assignment.

To help address these problems, we are experimenting with using the Web for the distribution, debugging, and collection of VHDL programming assignments[1]. In our senior-level computer architecture class, a student can use a standard Web browser for the compilation and simulation of VHDL code for specific hardware modules. Code is submitted through an HTML form interface and the results of the simulation are returned via an HTML table. This simple interface allows students to avoid tool complexities and concentrate on code development. Since the VHDL compilation and simulation tools run on a university server, there is no need for installing and licensing the tools on student computers. In fact, the Web interface allows students to work on their assignments from virtually any computer—including those not powerful enough to run the tools even if

a license were available. Our VHDL Web system also generates a personalized assignment for each student. The assignments for the different students all share a common structure, but their details differ. Hopefully, this will make each student work through a particular version of the assignment.

Many educators have also experimented with the automatic generation and grading of personalized homework assignments and quizzes. Several non-Web-based examples include Circuit Exerciser[2], CAPA[3], CircuitTutor[4], and CHARLIE[5]. These efforts have reported improved student grades, greater student effort on assignments, and increased teacher productivity. With the growth of the Internet, automated assignment systems are now exploiting the ubiquitous interface provided by a Web browser. Examples of Web-based systems include Mallard[6], Carnegie Mellon Online[7], and microprocessor courses at Case Western Reserve University[8]. Using the Web eliminates the need to distribute and install custom software packages to access the system. Also, students are already familiar with how to use the browser interface.

Our VHDL exercises are an attempt to apply the concept of Web-based, personalized assignments to problems involving programming and CAD tools. The next section examines several of the assignments that have been successfully incorporated into our system. The remaining sections discuss the Web-based implementation, the results, and possible future work.

II. Personalized VHDL Homework Assignments

From the student's perspective, our VHDL programming assignments have four steps:

1. Request a personalized assignment specification
2. Create/edit VHDL code on a local computer
3. Compile and simulate code on a remote computer
4. Submit the final code for grading

Students may repeat steps 2 and 3 many times while debugging. Steps 1, 3, and 4 are accomplished via our Web interface.

In step 1, a student requests a personalized assignment via an HTML form that asks for the student's university ID. When the form is processed, an HTML page with a mixture of tables and text is returned that specifies the details of the assignment. This specification is saved on the remote computer so that it may be retrieved at any time using the same form. The assignment specifications for the different students look similar, but their details vary. For example, our instruction decoder assignment varies instruction attributes such as opcode, field positions, and addressing conventions. The ALU assignment changes the size of the requested adder and the carry-lookahead unit. The pipelining assignment varies the number of stages in the execute and memory sections of the pipeline. The assignment specifications ask the student to create a behavioral or structural architecture for one or more VHDL entity declarations. For example, the pipelining assignment requests behavioral architectures for the forwarding and stalling units of the specified pipeline.

In step 2, a student may use any local computer and editor to create the VHDL code. When it is time to test the code, the student performs step 3 by entering the local file names for the VHDL

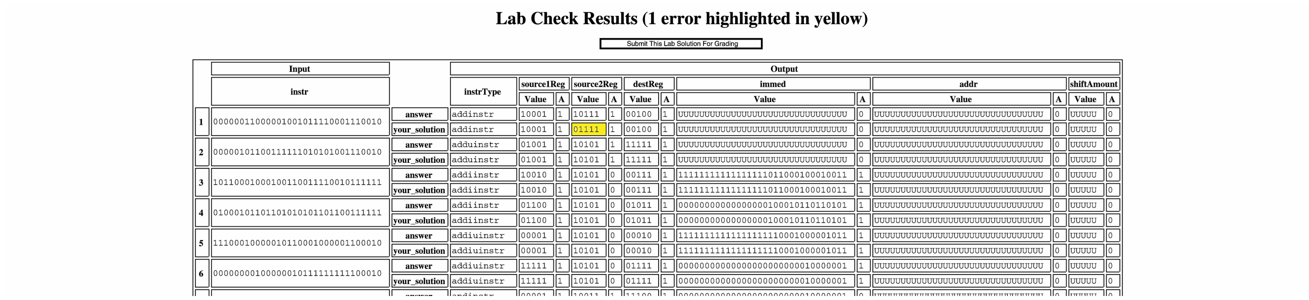
code files into another HTML form. When the form is processed, the code is transferred to the remote server machine for compilation and simulation. If compilation errors are found, an HTML page is returned that shows a list of the errors. If the code successfully compiles, it is simulated and the HTML page returned from the form's processing shows a combination of two sets of simulation data. One set consists of the results from simulating the student's code. The other set shows the correct simulation results. Differences between the student's results and the correct simulation are highlighted so that the student can quickly see where the behavior of the submitted code was wrong.

Regardless of the simulation results, a student can complete step 4 by pressing a form button on the HTML page returned from step 3. This submits for grading the last code that was simulated. In our present system, the number of edit-compile-simulate cycles is only limited by the student's patience. There is no penalty for the amount of debugging required. There are, of course, penalties for code that does not compile, does not simulate properly, or is not submitted by the due date. Students can resubmit their solutions if they realize how to fix remaining problems (the last submission is graded).

The format of the HTML page returned to the student in step 3 is critical to the success of an assignment. There must be sufficient information to debug the code, but there cannot be so much that it overwhelms the student with details. When students run the VHDL simulation tools directly on workstations in our computer lab, they have access to the full interface. They can examine signals and variables throughout a design and interact with a simulation as it proceeds. With the Web-based system, an entire simulation runs with a fixed set of test vectors in a batch mode and data is returned only on a preselected set of signals. A major question at the beginning of this project was how students would perform under these limitations.

As an introduction to VHDL, our instruction decoder assignment asks the student to define logic for parsing a 32-bit instruction into its field values based on the instruction's class. Figure 1 shows an example portion of the simulation output returned to the student. On the left side of the table is a list of instructions that was automatically generated when the student requested the assignment. During the simulation, the instructions are applied to the student's logic and the output values are captured and displayed on the right side of the table. For each instruction, the right side shows two rows of data. The first row is the answer and the second row is the student's solution. The yellow field indicates a difference between the student's results and the answer. In this assignment, the VHDL code does not need internal signals or state variables and the students reported no difficulty in using the table results to debug their logic blocks.

In our ALU assignment, the student must define a behavioral model for an individual bit and then assemble a structural model for an ALU using carry-lookahead units. A table similar to Figure 1 is returned to the student—the left side shows the input operands and the right side shows the output result of the ALU. It was clear that the students would have preferred access to the values on the internal signals of their structural models. This is difficult to provide without putting restrictions on how the students name and organize signals. Nevertheless, the students were quite successful with this assignment. They could temporarily change their code to use the provided outputs for examining signal values other than those for which the outputs were intended.



Lab Check Results (1 error highlighted in yellow)

Submit This Lab Solution For Grading

	Input: instr		Output: instrType	source1Reg Value	A	source2Reg Value	A	destReg Value	A	immed Value	A	addr Value	A	shiftAmount Value	A
1	00000011000001001011110001110010	answer	addinstr	10001	1	10111	1	00100	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
		your_solution	addinstr	10001	1	01111	1	00100	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
2	00000101100111111010101001110010	answer	adduinstr	01001	1	10101	1	11111	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
		your_solution	adduinstr	01001	1	10101	1	11111	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
3	10110001000100110011110010111111	answer	addiinstr	10010	1	10101	0	00111	1	11111111111111111011000100010011	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
		your_solution	addiinstr	10010	1	10101	0	00111	1	11111111111111111011000100010011	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
4	01000101101101010101101100111111	answer	addiinstr	01100	1	10101	0	01011	1	00000000000000001000101101101 01	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
		your_solution	addiinstr	01100	1	10101	0	01011	1	00000000000000001000101101101 01	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
5	11100010000010110001000001100010	answer	addiuinstr	00001	1	10101	0	00010	1	11111111111111111110001000001011	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
		your_solution	addiuinstr	00001	1	10101	0	00010	1	11111111111111111110001000001011	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
6	00000000100000010111111111100010	answer	addiuinstr	11111	1	10101	0	01111	1	00000000000000000000000010000001	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0
		your_solution	addiuinstr	11111	1	10101	0	01111	1	00000000000000000000000010000001	1	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU	0	UUUUU	0

Figure 1. Example Output for the Instruction Decoder Assignment

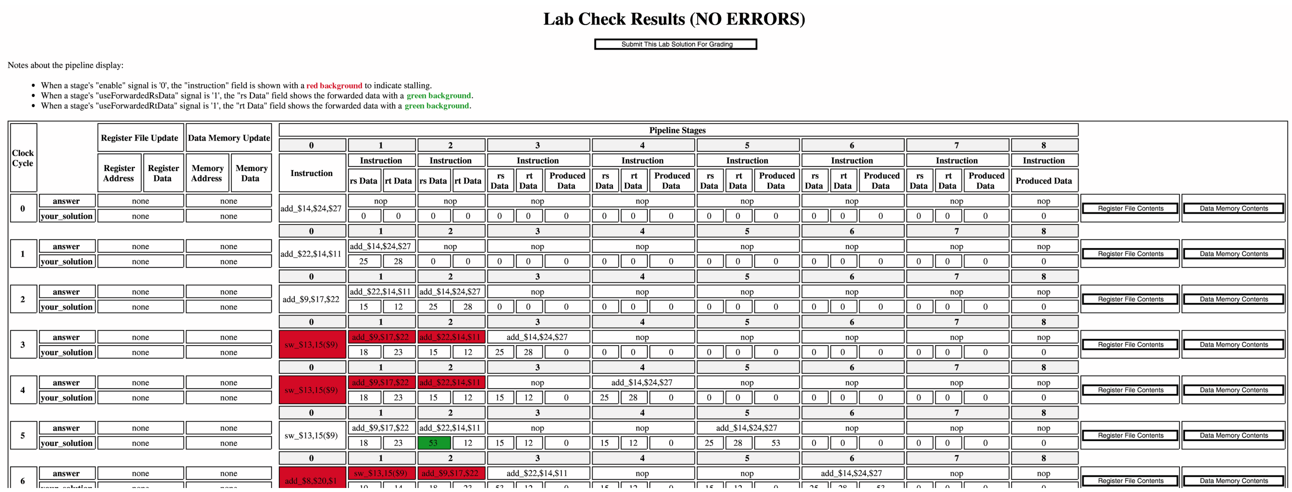


Figure 2. Example Output for the Pipelining Assignment

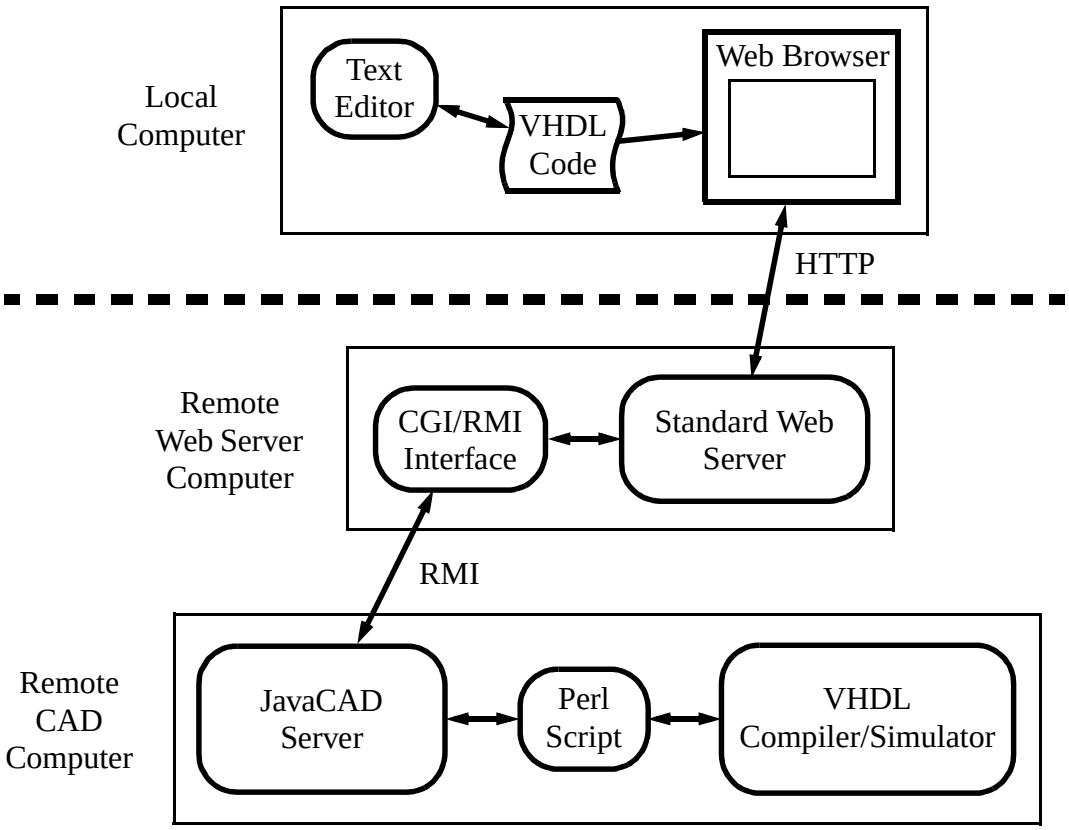


Figure 3. Implementation Components

Our pipelining assignment presented the greatest challenge in terms of returning simulation results to the students. In this assignment, the students must add stalling and forwarding logic to a processor with a 9-stage pipeline. There are many I/O signals for the stalling and forwarding logic blocks and a simple listing of their values would create a bewildering display of numbers. Figure 2 shows an example portion of the output returned to the student. The left side shows the updates to the registers and memory while the right side shows the flow of instructions through the student's pipeline. The signals for each stage of the pipeline were condensed using several techniques. The signals that indicate what instruction is in a stage and the registers it uses are displayed by converting them into a simple assembly language representation. Color is used to represent some of the key signal values controlled by the student's code. A red background for an instruction indicates that the stage is stalled. A green background for a data value indicates that the data was forwarded from some other part of the pipeline. Finally, JavaScript is used to hide values that are seldom used. At the extreme right of the table are JavaScript buttons that pop up dialog boxes containing listings of the current register and memory values.

III. Web-Based Implementation

Figure 3 shows the software and hardware components involved in processing the HTML forms used in our Web-based, personalized assignments. The "Local Computer" is whatever machine the student uses to edit VHDL code and run a Web browser. The "Remote Web Server Computer"

is a machine that the university uses for running a standard Web server. The “Remote CAD Computer” is a machine that is licensed by the university to run the CAD tools needed by the assignments. Both remote computers can be the same machine, but keeping them separate allows the remote CAD computer to be more secure.

The local computer needs only enough processing power to run a Web browser and editor. When the student submits a form to compile and simulate VHDL code, the browser reads the code from the file system on the local computer and transfers it over the Internet to the remote Web server computer. No special capabilities are required by the Web server running on this machine; it needs only the ability to execute common gateway interface (CGI) scripts. The VHDL code is transferred to the remote CAD computer using Java’s remote method invocation (RMI) protocol. Thus, a CGI/RMI interface was created with Perl and Java to allow a CGI script execution to make an RMI request.

The remote CAD computer runs a custom Java program called the JavaCAD server. We created this server to make batch-mode CAD processes remotely accessible over the Internet. The JavaCAD server can be called from an HTML form using the CGI/RMI interface mentioned above or directly from a Java client program via a standard RMI request. We use the HTML forms for the VHDL assignments and the direct RMI requests for a variety of other projects. When the JavaCAD server receives an RMI request, it checks the parameters for security problems and allocates temporary space for the execution of the requested CAD tool. Then, it spawns a Perl script that controls the details of the particular batch-mode CAD process. If the process does not complete its work within a given time limit, the process will be automatically killed. The JavaCAD server also manages resources by reclaiming temporary disk space and limiting the number of outstanding processes.

Each VHDL assignment requires the development of two custom Perl scripts called by the JavaCAD server. The first script is used when a student requests a personalized assignment (step 1 discussed in Section II). The work of this script involves the following:

1. Create the HTML specification of the assignment
2. Select the test vectors for simulations of the assignment
3. Generate the solution VHDL code
4. Compile and simulate the solution VHDL code to obtain the answers for comparison with simulations of the student’s code

When this script finishes, the HTML specification of the assignment is returned to the student. The second custom script that must be developed is used when a student wants to compile and simulate code (step 3 discussed in Section II). The work of this script involves the following:

1. Compile the VHDL code supplied by the student and return errors if necessary
2. Simulate the code with the selected test vectors and collect the output results
3. Generate the HTML table comparing the results of the student’s code to the answers obtained when simulating the solution code

Even though the details of the scripts change from assignment to assignment, the basic pattern of the code stays the same and this helps reduce the programming effort. The most time-consuming part of developing a new assignment is not the writing of the custom scripts but is the design of what the assignment will involve, how it will differ for the individual students, and how the simulation output will be displayed.

IV. Results

Although the full, interactive simulator interface has greater flexibility, our HTML table approach seems to work well for these introductory assignments. We had been concerned about the practicality of debugging a programming assignment over the Web, but the students adapted quickly and many expressed a preference for the Web interface. This is probably demonstrated most clearly by the small number of students who resorted to using the VHDL tools directly on a lab workstation—even though many students had prior experience with the tools. In some ways, the HTML tables are actually superior to the direct simulator interface. The tables displayed for the pipelining assignment condense a large amount of information that would be more difficult to comprehend as simulator waveforms. Of course, students specializing in this area must eventually learn how to control the CAD tools themselves, but we have other courses that focus on tool usage. The Web interface helps ease this course's diverse set of students into VHDL programming. (The fall '98 class included 48 students drawn from the undergraduate and graduate programs in computer, electrical, and computational engineering as well as computer science and physics.)

As other educators have noticed, immediate feedback on errors seems to highly motivate students to work until they achieve a perfect score. Nearly every student completed our VHDL assignments correctly before the due date; only about 10% of the students submitted the assignments late or with errors. As might be expected, the students performing the best on the in-class tests consistently completed their assignments within a few days while the other students usually took more than a week. Since our system time stamps the code submissions, it would be possible to factor into the grade the length of time required to complete an assignment in order to reward the students who turn in their solutions early.

From the instructor's perspective, the Web-based system improves the efficiency of answering individual questions and grading. A directory structure accessible to the instructor holds the last code files that a student submitted along with the code's simulation results. In response to an email or phone call from a student, the instructor can quickly look at the same material the student is viewing. At the end of the assignment, the submitted code is still manually examined to help build an impression of how the various students are doing, but the automatic simulations and comparisons save a lot of repetitive work. This allows the instructor to concentrate on giving advice to help improve the programming skills of the students.

V. Future Work

To support more complex assignments in the future, improvements are needed in the way displays of simulated results are created and organized. Even generating an HTML table can require some tedious programming. Thus, Perl/Java libraries that provide object-oriented abstractions for the creation of displays with tables, block diagrams, and waveforms would be very useful. Also,

condensing simulation results onto a single HTML page is more challenging as the complexity of the assignments grow (for example, the results page for the pipelining assignment has become quite large). The JavaCAD server has the ability to allocate and manage temporary space for a collection of Web pages that hold information about the results of a request. By exploiting this feature, more complex Web page structures describing simulation results could be built. That is, a relatively small page could be returned that contained links to more detailed information on other pages. Also, more JavaScript could be used to display information only when it is requested.

Although this paper discussed VHDL exercises, a similar approach might be useful for other kinds of programming assignments. This would be especially true if the goal of the assignments is to quickly introduce the basics of a language without having to deal with a development tool's learning curve or the issues associated with the tool's distribution and licensing. Even if there are no problems with using the development tools, a Web-based testing system could complement the tools by providing more comprehensive feedback and possibly unique displays of the code's behavior.

Bibliography

1. See http://www.ece.msstate.edu/~linder/Courses/EE4713/labs/ (use social security number 111223333 to experiment with the assignments).
2. A. Yoshikawa, M. Shintani, and Y. Ohba, "Intelligent Tutoring System for Electric Circuit Exercising," *IEEE Transactions on Education,* vol. 35, pp. 222-225, August 1992.
3. E. Kashy et al., "CAPA—An integrated computer-assisted personalized assignment system," *American Journal of Physics*, vol. 61, pp. 1124-1130, December 1993.
4. B. Oakley, "A Virtual Classroom Approach to Teaching Circuit Analysis," *IEEE Transactions on Education,* vol. 39, pp. 287-296, August 1996.
5. D. Barker, "CHARLIE: A Computer-Managed Homework, Assignment and Response, Learning and Instruction Environment," *Frontiers in Education Conference,* pp. 1503-1509, 1997.
6. M. Swafford et al., "Mallard™: Asynchronous Learning in Two Engineering Courses," *Frontiers in Education Conference*, pp. 1023-1026, 1996.
7. D. Rehak, "Carnegie Mellon Online: Web-Mediated Education," *Frontiers in Education Conference,* pp. 1510-1516, 1997.
8. F. Merat and D. Chung, "World Wide Web Approach to Teaching Microprocessors," *Frontiers in Education Conference,* pp. 838-841, 1997.

DANIEL H. LINDER
At Mississippi State University, Daniel H. Linder is an assistant professor in the Department of Electrical and Computer Engineering and is also associated with the NSF Engineering Research Center for Computational Field Simulation. His current research interests include computer architecture and electromagnetic simulation. He received a Ph.D. in Electrical Engineering from Mississippi State University in 1994.