Enhancing VHDL Learning through a Light-weight Integrated Environment for Development and Automated Checking

Akash Kumar, Rajesh C. Panicker and Ashraf Kassim
Department of Electrical & Computer Engineering, National University of Singapore, Singapore
Corresponding author email: akash@nus.edu.sg

Abstract—**The development environments for Hardware Description Languages (HDLs) are essentially meant and designed for highly trained professionals/ engineers and as such are not suitable for use as an introductory tool for students learning HDLs. With students adopting a variety of operating systems, there is a need for a light-weight and cross-platform environment. Further, such a development environment for students should be able to provide some feedback about the functional correctness of their program. In this paper, we describe an open-source environment for developing and simulating VHDL programs on the client side, and a server side application for automated checking of submissions. The client has been developed for three desktop operating systems – Windows, Linux and MacOSX. The server application runs on Linux. The client allows students to develop and simulate VHDL programs. They can also submit programs to a remote server for automated verification. The environment – client and server, has been used for two semesters at the National University of Singapore to provide an enhanced learning experience to the students in a first year course on digital fundamentals.**

Index Terms—**Integrated development environment, VHDL, Simulation, Automated checking.**

I. Introduction

Very high speed integrated circuit hardware description language (VHDL) is one of the two most popular hardware description languages, the other being Verilog. VHDL is widely used in academia and industry. It was developed by US department of defense and standardized by the IEEE (standards 1076-1987, 1076-1993, 1076-2008). It is a strictly typed, formal language syntactically similar to ADA. VHDL is widely taught as a part of the first or second course in digital design in undergraduate curricula of electrical / computer engineering. Hence, teaching community has been developing several tools and techniques for enhancing the students' learning experience of VHDL [1], [2]. VHDL is also widely taught as a part of more advanced courses such as computer architecture [3], [4], and increasingly, as an alternate language for describing systems in other electrical / computer engineering courses [5].

Today's electronic systems, ranging from digital audio systems to complex computers, are substantially realized using digital logic. At National University of Singapore's Department of Electrical and Computer Engineering, EE2020 Digital Fundamentals is a first course that introduces fundamental digital logic, digital circuits, and programmable devices to the students [6]. This course provides students with an understanding of the building blocks of modern digital systems and methods for designing, simulating and realizing such systems. The emphasis of this module is on understanding the fundamentals of digital design across different levels of abstraction using hardware description languages and lab exercises.

In fundamental courses, students are taught how to describe and simulate simple circuits using VHDL. However, most courses use bulky integrated development environments (IDEs) such as Xilinx Webpack or Altera Quartus as a tool for editing and simulating designs. For example, the most basic version of Xilinx ISE, the Webpack has a download size of more than 6GB and an installed size (depending on the selected features) close to 12GB. Moreover, it can be installed only on Windows and Linux platforms; thus it is not truly a multi-platform software. The increasing MacOSX user base is unable to use it without a virtual machine running a supported operating system (OS). While it is very feature rich and supports synthesis etc., such additional functionalities are probably unnecessary at the entry level. The large feature set may in fact, increase the learning curve for students in using the software. The software is typically required only for 6-8 weeks; only students who take advanced modules requiring VHDL need this software in the future. Hence, we have developed an IDE utilizing only open-source tools – jEdit, GHDL and GTKWave. The IDE is light-weight and has an installer of size less than 20MB. It is truly multi-platform and runs on MacOSX too. Further, since it is based on flexible editor jEdit, its functionalities can be easily enhanced through Java-based plugins.

For efficient learning of programming, it is very important for students to receive immediate feedback about the correctness of the program. While a (good) compiler is able to identify most syntax errors, extensive simulation is needed to ensure that the program works correctly for all scenarios. Sometimes, even though students think that they have the correct answer for all scenarios, the timing (which is very important in hardware design) may be incorrect. Manual checking of code and resulting simulation waveforms is not feasible for large classes. An automated checker has therefore been developed that allows students to submit the code to a server and receive feedback on the correctness of the code.

Several tools for automated checking / evaluation for teaching programming have been reported in the literature [7]–[11]. Comprehensive reviews on this topic can be found in [9] and [10]. However, automated checking in VHDL has received only limited attention. Gutierrez *et al.* [2] developed an automated VHDL checking and collaboration system where students can submit their designs to *Moodle* (an e-learning platform [12]) for automated checking. This was further extended to a computer architecture module, where students simulated a complete simplified CPU written in VHDL [4]. In this paper, we describe a light-weight application and an automated VHDL source checking system, which is targeted to enhance the learning experience of students specifically in a course on digital fundamentals. It broadly includes the following functionalities:

- A light-weight cross-platform open-source IDE for editing, compiling and simulating VHDL programs.
- A client (integrated into the IDE) – server system to do automated checking of VHDL programs.
- An interface for the teaching staff to update the database of problems whose solutions can be automatically checked.
- A logging system to keep track of student submissions, which can be useful in programming tests.

The rest of the paper is organized as follows. Section II provides details of the IDE that has been built using various open source tools into a nice package. Section III gives more information on client and server components required for the automated checking feature. Section IV presents relevant results to evaluate the efficiency of the developed IDE and the automated-checker. Section V concludes the paper and provides directions for future works.

II. IDE Implementation

There are a number of development environments available for designing circuits using a hardware description language (HDL). However, most of these are commercial tools that need to be purchased [13], [14]. Further, since they are aimed at developing commercial designs, most of the available features are often not necessary in an introductory level course. For basic HDL development, students only need to edit, compile and simulate simple programs, typically contained within a single file. In the following subsections, we provide further details about the individual components put together in the light-weight IDE.

A. Editor

The text editor should support syntax highlighting for VHDL, allow custom plugins and run on multiple platforms. jEdit [15] is a mature programmer's text editor which is feature rich and easy to use. It is released as a free software with full source code, provided under the terms of the GPL 2.03. Some of the relevant features include:

- Written in Java; runs on all desktop OSs.
- Built-in macro language; extensible plugin architecture, allowing custom plugins to be developed easily.
- Auto indent and syntax highlighting for a lot of languages including VHDL.
- Support for code-folding and word-wrap.

Figure 1 shows the jEdit editor with a custom plugin designed for VHDL compilation, simulation, etc. The editor area shows a VHDL code snippet with highlighted syntax. The custom plugin is docked below the editor area, with user interface (UI) controls for performing various tasks. Since the editor and the plugin is based on Java, it works cross-platform without the need for re-compilation.

B. Compiler and Simulator

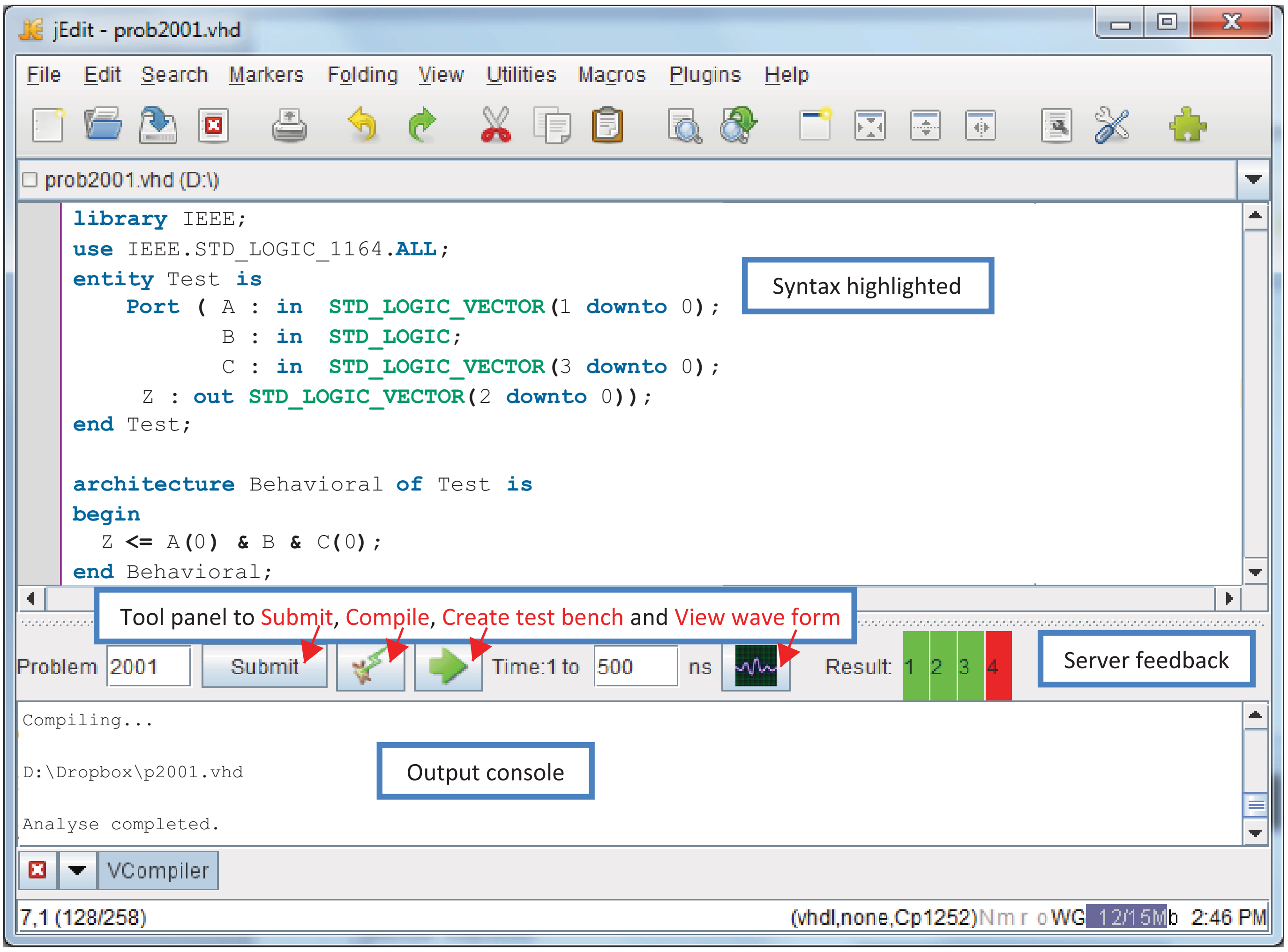
The compiler and simulator should also be light, open-source and cross-platform. GHDL [16] meets these requirements and is hence ideally suited for our purpose. It allows the user to compile and execute VHDL code directly. It has several commands allowing the user to analyze, elaborate and run VHDL code / testbenches with various options. Though it is a command-line tool making it powerful, it can often be hard to use for a beginner for that reason. In our IDE, this tool is invoked using the UI controls, essentially making GHDL transparent to the user. Upon simulation, a value change dump (VCD) file is produced.

C. Waveform Viewer

The VCD file is a text file with the values of signals at various time points in the simulation. For easy visualization of results, we need a program which shows this information graphically. GTKWave is an open-source GTK+ based wave viewer which runs on Unix, Windows and MacOSX [17]. It supports several file formats including standard VCD files. It is an interactive tool allowing features to be invoked using command-line options. These are again made transparent to the user through the use of plugin UI controls. As shown in Figure 2, we can select the signals to be displayed, their radix and zoom-level. Moving the cursor about the signals displays the value of the signal at particular time instant.

D. Plugin

The plugin integrates the compiler, simulator and the waveform viewer with jEdit, and handles the communication with the server. The visual interface of the plugin has three main components – 1) compilation, testbench generation and simulation controls; 2) output console; and 3) server interactivity controls. GHDL is invoked using proper command-line options to analyze the code upon pressing the 'compile' button. Syntax errors, if any, are displayed in the output console.


jEdit - prob2001.vhd
File Edit Search Markers Folding View Utilities Macros Plugins Help
prob2001.vhd (D:\)
Syntax highlighted
Tool panel to Submit, Compile, Create test bench and View wave form
Problem 2001 Submit Time:1 to 500 ns
Result: 1 2 3 4
Server feedback
Output console
VCompiler
7,1 (128/258)
(vhdl,none,Cp1252)Nm r o WG 12/15Mb 2:46 PM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Test is
    Port ( A : in  STD_LOGIC_VECTOR(1 downto 0);
           B : in  STD_LOGIC;
           C : in  STD_LOGIC_VECTOR(3 downto 0);
      Z : out STD_LOGIC_VECTOR(2 downto 0));
end Test;

architecture Behavioral of Test is
begin
  Z <= A(0) & B & C(0);
end Behavioral;
```

```
Compiling...

D:\Dropbox\p2001.vhd

Analyse completed.
```

Fig. 1. Integrated development environment with the plugin docked at the bottom

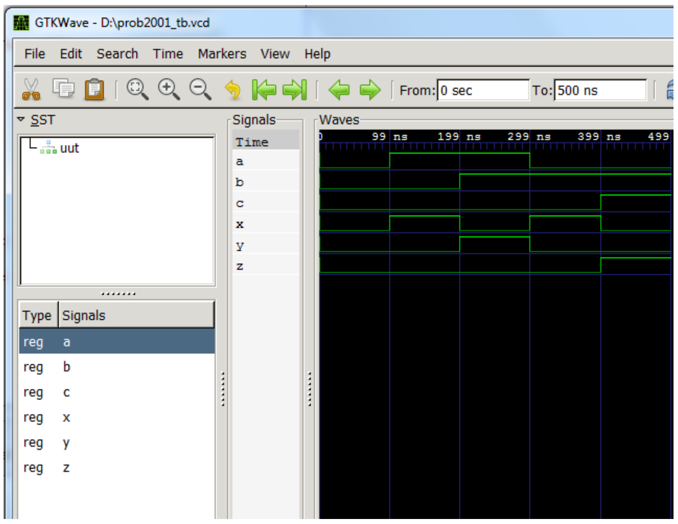


Fig. 2. The GTKWave window showing an output waveform

'Create testbench' button can be used to create a testbench template automatically. The various input / output ports are identified by parsing the entity in the VHDL unit under test, and an appropriate architecture is created. To complete the testbench code, the user just needs to fill in the stimuli (test pattern) to be applied. A sample testbench generated by the plugin for the code given in Fig. 1 is shown in Fig. 3.

The simulation is performed through the 'simulate' button after entering the duration for which the simulation needs to be performed. This invokes GHDL which runs the testbench, generating a VCD file, followed by an automatic invocation of GTKWave. The statuses and messages from various tools can be viewed in the output console. More details on the communication with the server is presented in Section III.

The three open-source components as well as the plugin needs to be configured properly for correct functionality. We have created custom installers which does this automatically for Windows, Linux as well as MacOSX so that the users can get the IDE to run out of the box. It should be noted that while jEdit, being Java based runs on all the platforms without re-compilation, other components as well as the installer have separate binaries for each platform.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY hh_tb IS
END hh_tb;

ARCHITECTURE behavior OF hh_tb IS
   COMPONENT Test
   PORT(
      A: in STD_LOGIC_VECTOR(1 DOWNTO 0);
      B: in STD_LOGIC;
      C: in STD_LOGIC_VECTOR(3 DOWNTO 0);
      Z: out STD_LOGIC_VECTOR(2 DOWNTO 0));
   END COMPONENT;

   signal A: STD_LOGIC_VECTOR(1 DOWNTO 0) := (others => '0');
   signal B: STD_LOGIC := '0';
   signal C: STD_LOGIC_VECTOR(3 DOWNTO 0) := (others => '0');
   signal Z: STD_LOGIC_VECTOR(2 DOWNTO 0) := (others => '0');
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: Test PORT MAP (A => A, B => B, C => C, Z => Z);
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      -- insert stimulus here

      wait;
   end process;
END behavior;
```

Fig. 3. Listing of the sample testbench created automatically for the program shown in Figure 1

III. Automated Checker

In order to provide quick feedback to students, the IDE provides a feature to check the correctness of selected problems automatically by submitting them to the server. In the following subsections, we provide further details about the client (an IDE feature) and server that have been developed for automated checking of VHDL programs and associated tasks. Details of a problem-database and management of submissions is also provided.

A. Server Connector

The developed IDE plugin supports automated checking. Once the program has been compiled, students can submit it after indicating the corresponding problem identifier. The server compiles the submitted program, carries out the simulation using the input test cases for the problem and compares it with the corresponding output cases. For each test case, a response is sent to the client indicating the correctness. Figure 1 shows the response for problem 2001, where the program produced correct output for three out of the four test cases. The plugin supports a maximum of ten input cases per problem (although this can be changed easily if necessary). More details about input size and format are given in Section III-C.

B. Automated Checking Server

The server for automated checking of VHDL programs was inspired by the various online judging systems for programming contests [18], [19]. Such online judging systems accept the user solution to a problem and compares the output produced against a set of input cases. Since there are usually no unique answers in programming, it is not possible to directly compare the code to a model solution. Instead, the program has to be compiled and tested across multiple cases for correctness. This part of compilation, checking and sending the response to the client is handled by a server which is described below.

In order to handle multiple requests from students to the server, the server platform adopted was the Apache Tomcat [20], an open source implementation of the Java Servlet and JavaServer Pages technologies with the specifications developed under the Java Community Process. A Java Servlet is a Java programming language class used to extend the capabilities of servers that can be accessed by a host application via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. Thus, it can be thought of as a Java Applet that runs on a server instead of a browser. JavaServer Pages (JSP) is a technology that enables dynamic generation of web pages based on HTML, XML, or other document types. JSP is similar to PHP, but it uses Java programming language.

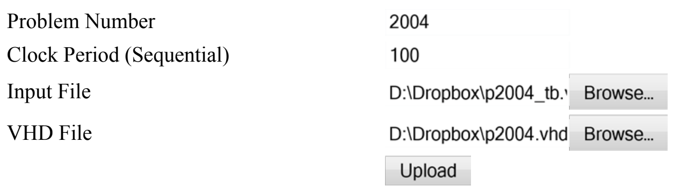
Two servlets are written for the server – *teacher servlet* and *student servlet*. The teacher servlet handles addition of new problems to the system by managing the VHDL files and input files associated with the test cases. More details about this feature is provided in Section III-C. The student servlet handles VHDL file submissions by students. It compiles their program and checks the output produced for corresponding input cases and sends the response to the client. The servlet also takes care of organizing the files on the server for logging purposes. More details on this feature are provided in Section III-D.

C. Problem Database

An interface is provided to upload new problems and solutions to the server. This is intended to be used by teachers to add new problems for students to solve. The interface shown in Figure 4 has the problem identifier (i.e., problem number), the clock period/duration (default is 100ns), input cases (for evaluating / comparing submissions) and the problem solution which is used to generate the model output. The three options for upload to the problem database are:

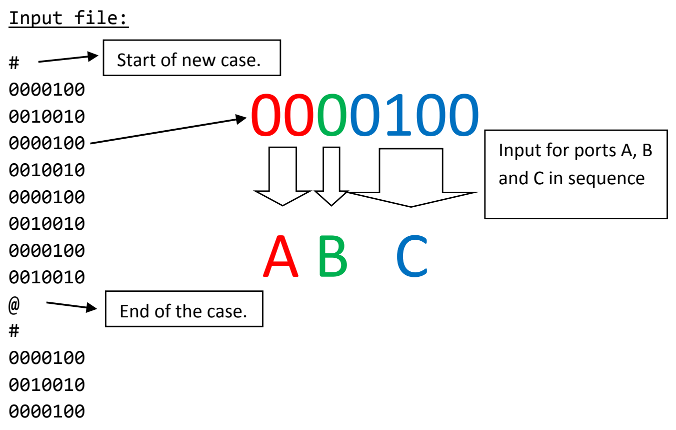
1) A combinational VHDL file without any input file or clock period.
2) A combinational VHDL file with an input file. (Clock is not applicable for a combinational circuit.)
3) A sequential VHDL file with an input file and clock period.

Once a new problem is submitted, the teacher servlet creates a folder for the problem. This folder contains the input cases and the model VHDL answer. The format of the file containing



Problem Number	2004
Clock Period (Sequential)	100
Input File	D:\Dropbox\p2004_tb.\ Browse...
VHD File	D:\Dropbox\p2004.vhd Browse...

Fig. 4. Webpage for uploading new problems on the server

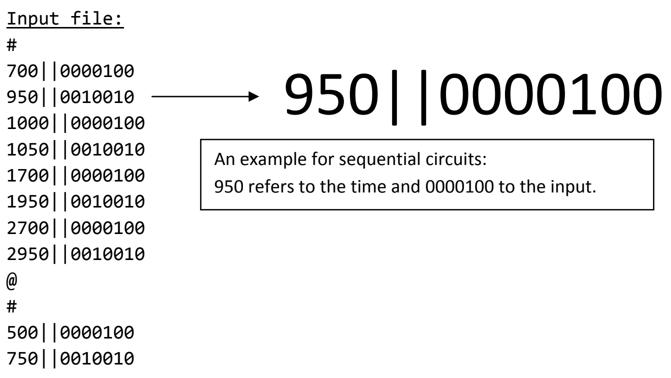


```
Input file:
#
0000100
0010010
0000100
0010010
0000100
0010010
0000100
0010010
@
#
0000100
0010010
0000100
```

Fig. 5. Sample input file for providing test cases for combinational circuits

input cases depends on whether the problem requires combinational or sequential description. Figure 5 shows a sample input file for a combinational circuit. While a maximum of ten input cases are supported, each test case can be as long as desired. Each test case is enclosed between '#' and '@' as shown in the figure. The order of values on a line is determined by the order of input signals declared in the model VHDL file. For the example shown in Figure 5, there are three inputs – A, B and C which are 2-bit, 1-bit and 4-bit wide respectively. If no input file is provided, all possible input combinations are used as a single test case.

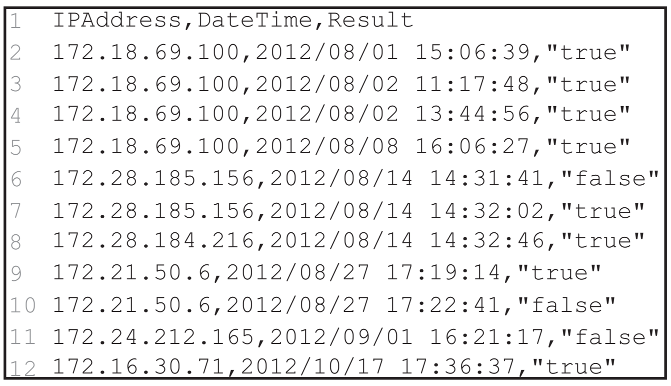
Figure 6 shows a sample input file for a sequential circuit. Each line in this file indicates the time at which the inputs get their designated value, followed by the value itself, demarcated using '||'. A separate webpage showing the problems students can try out along with their identifiers is maintained, as shown in Figure 7.



```
Input file:
#
700||0000100
950||0010010
1000||0000100
1050||0010010
1700||0000100
1950||0010010
2700||0000100
2950||0010010
@
#
500||0000100
750||0010010
```

950||0000100
An example for sequential circuits:
950 refers to the time and 0000100 to the input.

Fig. 6. Sample input file for providing test cases for sequential circuits



```
IPAddress,DateTime,Result
172.18.69.100,2012/08/01 15:06:39,"true"
172.18.69.100,2012/08/02 11:17:48,"true"
172.18.69.100,2012/08/02 13:44:56,"true"
172.18.69.100,2012/08/08 16:06:27,"true"
172.28.185.156,2012/08/14 14:31:41,"false"
172.28.185.156,2012/08/14 14:32:02,"true"
172.28.184.216,2012/08/14 14:32:46,"true"
172.21.50.6,2012/08/27 17:19:14,"true"
172.21.50.6,2012/08/27 17:22:41,"false"
172.24.212.165,2012/09/01 16:21:17,"false"
172.16.30.71,2012/10/17 17:36:37,"true"
```

Fig. 8. Listing of a log file with IP address, time and the result of submissions

D. *Submission Management*

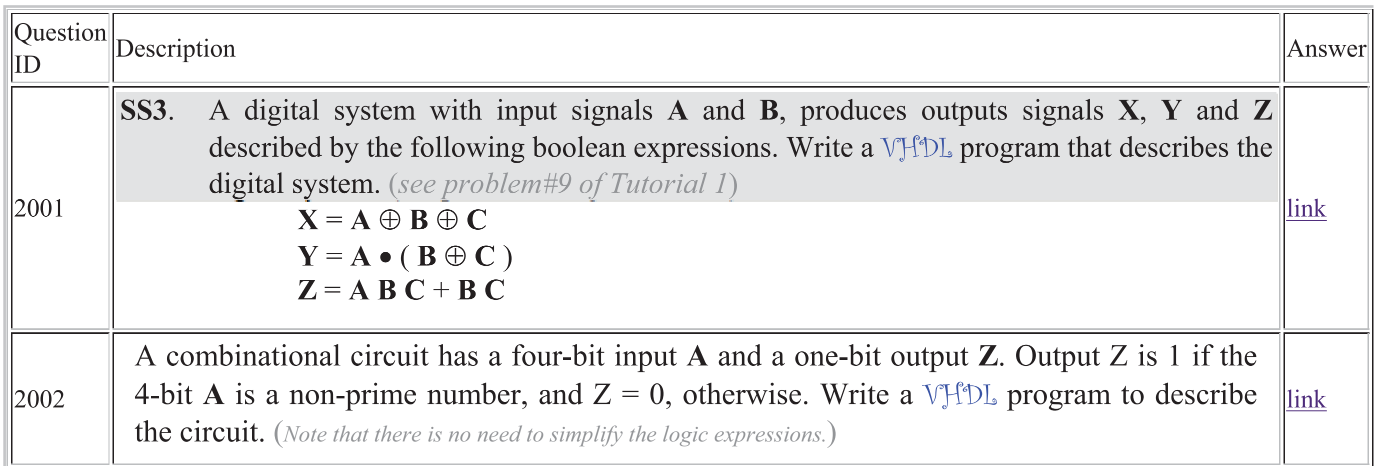
When a submission is received by the server, the student servlet copies them into an appropriate folder as indicated by the problem identifier. Another folder with the IP address of the client is also created. For each submission, the server creates a new sub-folder with the time-stamp. The servlet then compiles the submitted file with the model testbench and returns the result to the client. All submissions are thus logged in the system. Figure 8 shows a sample log file. The log file stores the IP addresses of the submissions, their date and time stamps as well as the results. This serves two main purposes – one is for debugging, and the other, more important use is in an examination setting where students are expected to submit VHDL programs for a particular problem. Since the files stored are rather small text files, it is efficient with respect to the storage space required.

IV. Evaluation and Results

In this section, we present major observations on a survey conducted to evaluate the usability and usefulness of our tool. Most questions were evaluated on a Likert scale from 1 to 5, with 5 being the most desirable. Table I shows the various survey questions and the average of the responses. Most users found the installation process rather easy with our custom installers for different platforms. None of the users found it difficult to install with 50% users finding it extremely easy to install.

Most users (63% with a score of 4 and above) felt it was important to have a simple VHDL editor since existing tools are very bulky. Close to three-quarters of respondents found it easy (score of 4 or 5) to use the editor. The automated checking feature was received very well with close to 90% respondents feeling a strong need for it.

A similar number of people felt that such a simple editor and automated checking feature can motivate and enhance the learning process. Some useful suggestions to improve the interface were also received during the feedback which will be incorporated in the future versions of the tool.



Question ID	Description	Answer
2001	**SS3**. A digital system with input signals **A** and **B**, produces outputs signals **X**, **Y** and **Z** described by the following boolean expressions. Write a VHDL program that describes the digital system. *(see problem#9 of Tutorial 1)* $\mathbf{X = A \oplus B \oplus C}$ $\mathbf{Y = A \bullet (B \oplus C)}$ $\mathbf{Z = A\,B\,C + B\,C}$	link
2002	A combinational circuit has a four-bit input **A** and a one-bit output **Z**. Output Z is 1 if the 4-bit **A** is a non-prime number, and Z = 0, otherwise. Write a VHDL program to describe the circuit. (*Note that there is no need to simplify the logic expressions.*)	link

Fig. 7. Sample problems with descriptions and problem identifiers

TABLE I
SURVEY RESULTS OF THE DEVELOPED IDE

Question	Avg. Score
How would you rate the installation process?	4.1
How important is it to have a light-weight IDE?	3.8
How usable is the provided editor?	3.9
How important is to have automated checking feature?	4.4
How usable is the automated checking feature?	3.9

V. CONCLUSIONS AND FUTURE WORK

We have created (i) a light-weight open-source IDE for editing, compiling and simulating VHDL programs and (ii) an automated system for verification of VHDL programs submitted by students. The IDE is cross-platform and installers for the three most popular platforms – Windows, Linux and MacOSX are provided. The software are available for download at [21].

As mentioned in Section III-D, we plan to explore the possibility of using the IDE for the programming test conducted as a part of continuous assessment for EE2020. This will make the administration of the test easier, and the evaluation more objective. Since the accuracy of the program can be checked instantly, the evaluation workload of teaching assistants will be reduced. Further, in the current system, the webpage where the problems are hosted has to be edited manually. Making it automated would greatly ease the administrative workload – perhaps a text file with the problem description can be uploaded together with the sample program and test cases. We also intend to collect more feedback from students and make appropriate modifications to enhance features and usability.

ACKNOWLEDGEMENTS

We would like to thank Ong Jing Jun and Doan Viet Tiep for their help in programming the system. We would also like to thank CDTL (Centre for Development in Teaching and Learning), NUS for supporting this project through the Teaching Enhancement Grant (TEG).

REFERENCES

[1] A. Wu, “Interactive learning toolbox for logic synthesis with VHDL,” in *Proc. IEEE Intl. Conf. Microelectronic Sys. Edu.*, 1997, pp. 77–78.
[2] E. Gutirrez, M. A. Trenas, J. Ramos, F. Corbera, and S. Romero, “A new moodle module supporting automatic verification of VHDL-based assignments,” *Computers & Education*, vol. 54, no. 2, pp. 562 – 577, 2010.
[3] T. C. Huang, R. Melton, P. Bingham, C. Alford, and F. Ghannadian, “The teaching of VHDL in computer architecture,” in *Proc. IEEE Intl. Conf. Microelectronic Sys. Edu.*, 1997, pp. 133–134.
[4] M. Trenas, J. Ramos, E. Gutierrez, S. Romero, and F. Corbera, “Use of a new moodle module for improving the teaching of a basic course on computer architecture,” *IEEE Tran. Edu.*, vol. 54, no. 2, pp. 222–228, 2011.
[5] F. Azcondo, A. De Castro, and C. Braas, “Course on digital electronics oriented to describing systems in VHDL,” *IEEE Tran. Indus Elec.*, vol. 57, no. 10, pp. 3308–3316, 2010.
[6] A. A. Kassim, S. A. Kazi, and S. Ranganath, “A web-based intelligent learning environment for digital systems,” *International Journal of Engineering Education*, vol. 20, no. 1, pp. 13–23, 2004.
[7] M. Amelung, K. Krieger, and D. Rosner, “E-assessment as a service,” *IEEE Tran. Learn. Tech.*, vol. 4, no. 2, pp. 162–174, 2011.
[8] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, “On automated grading of programming assignments in an academic institution,” *Computers & Education*, vol. 41, no. 2, pp. 121–131, 2003.
[9] K. M. Ala-Mutka, “A survey of automated assessment approaches for programming assignments,” *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005.
[10] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, “Review of recent systems for automatic assessment of programming assignments,” in *Proc. 10th Koli Calling Intl. Conf. Computing Education Research*. New York, NY, USA: ACM, 2010, pp. 86–93.
[11] C. Daly and J. Horgan, “An automated learning system for Java programming,” *IEEE Tran. Edu.*, vol. 47, no. 1, pp. 10–17, 2004.
[12] Moodle. (2013). [Online]. Available: https://moodle.org
[13] Xilinx. (2013). [Online]. Available: http://www.xilinx.com
[14] Altera. (2013). [Online]. Available: http://www.altera.com
[15] jEdit. (2013). [Online]. Available: http://www.jedit.org/
[16] GHDL. (2013). [Online]. Available: http://ghdl.free.fr/
[17] GTKWave. (2013). [Online]. Available: http://gtkwave.sourceforge.net/
[18] UVa. (2013) Universidad de Valladolid Online Judge. [Online]. Available: http://uva.onlinejudge.org/
[19] SPOJ. (2013) Sphere online judge. [Online]. Available: http://www.spoj.com/
[20] Tomcat. (2013). [Online]. Available: http://tomcat.apache.org/
[21] A. Kumar. (2013) VHDL IDE Download page. [Online]. Available: http://www.ece.nus.edu.sg/stfpage/eleak/teaching.html