INTEGRATING VHDL INTO A FIRST COURSE IN LOGIC DESIGN

Peter Walsh
Department of Computing Science
Malaspina University-College
Nanaimo, B.C., V9R 5S5 Canada
and
Department of Computer Science
University of Victoria
P.O. Box 3055, Victoria, B.C., V8W 3P6 Canada

Abstract

Present industry practice has created a high demand for systems designers with knowledge and experience in using hardware description languages. Many universities offer this type of training in advanced digital engineering courses. In this paper, we describe our experience in integrating VHDL into a first course in logic design.

1 Introduction

CSCI 355 is a course in Digital Logic and Computer Organization offered annually at Malaspina University-College in partnership with the University of Victoria. Course prerequisites include CSCI 261 Computer Architecture and Assembly Language Programming and CSCI 161 Computer Programming II (CS2 in the ACM curriculum). Topics include combinational circuits, sequential circuits and an introduction to systems level digital design. The course extends over fourteen weeks with three hours of lectures per week and nine two-hour laboratory sessions.

Material covered in the laboratory sessions serves to reinforce the understanding of the material covered in the lectures. In the past, students were introduced to breadboarding and designed and implemented systems using SSI, MSI, LSI devices. Our primary goal was to modernize the laboratory component of the course by replacing breadboarding with modeling using the hardware description language VHDL. Our secondary goal was to teach VHDL from a software engineering perspective.[1] To this end, we emphasize design verification through systematic testing and implementation to specification.

[1] Our students are declared computing science majors and have been exposed to key software engineering principles (contained in CSCI 355's prerequisite courses).

Section 2 describes our approach to systematic testing using *cew*, a component exerciser workbench. Section 3 describes our approach to quality control using acceptance tests. Acceptance testing using *cew* forms the corner stone of our quality system. Section 4 introduces texts and tools we considered for our course. Section 5 describes our experience running the course for the first time and section 6 contains concluding remarks.

2 Systematic Testing

Current practice is to model a design component in VHDL and then test the design's behavior using a testbench, which applies test cases and reports deviations of actual behavior from expected behavior. We use *cew*, a workbench for developing test suites for VHDL, C++ and Java [1]. Testing VHDL components with *cew* involves a special interface to the component under test (CUT), a scripting language and a driver generator. *cew* is oriented towards highly automated testing, where the driver generates the inputs, runs the tests, and checks the outputs for correctness.

Figure 1 overviews a testbench for a JK flip flop using *cew*. The I/O signals of the CUT are connected to test pins within the testbench. A call based testing interface is developed to provide a testers abstraction of the CUT. The syntax of a test-case in *cew* is *cew_Ntest(trace, actval, expval)*. A *trace* is any manipulation of the CUT through the testing interface or through the test pins. The *actval* is an expression that is evaluated after the trace. It's value is taken as the "actual value" of the trace. *expval* is the value that *actval* is expected to have.

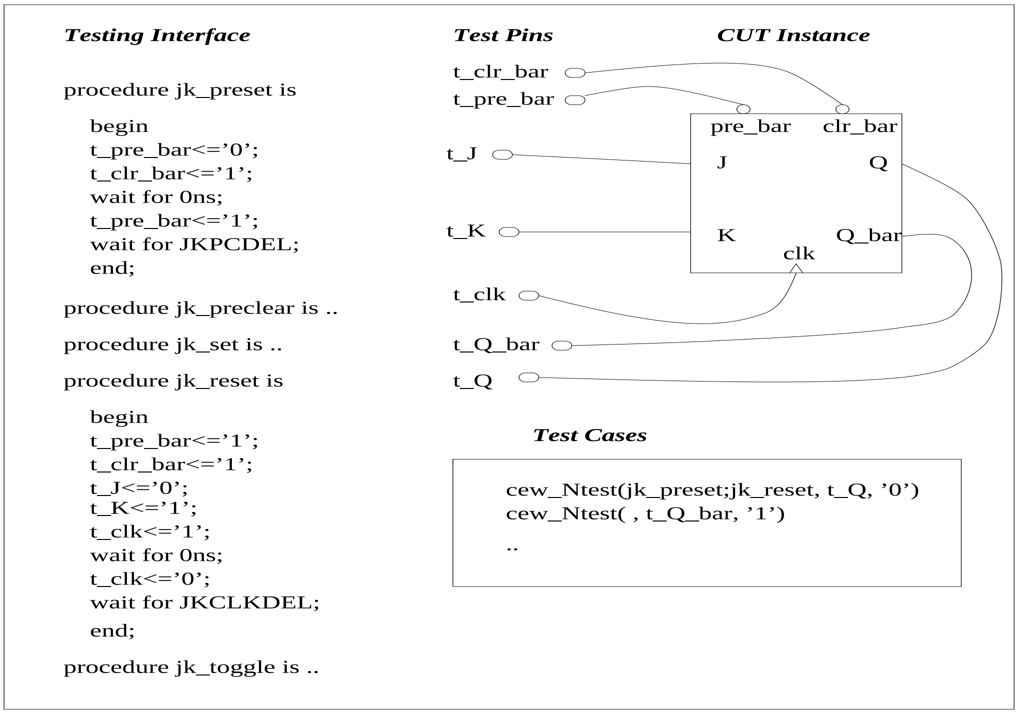

Figure 1: Testbench Overview for a JK Flip Flop

cew operates in one of three modes *viz*, `Interactive`, `Interactive_On_Failure` and `Batch`. `Batch` mode is used for regression testing. In `Batch` mode, *cew* produces a listing detailing failed test cases and summary statistics. The other two modes allow for interactive testing. Figure 2 shows a *cew* interactive test script for the JK flip flop testbench. The script produces a menu that can be used to debug the CUT or the test cases. The syntax of a menu item is *cew_Menu_Item(sel_char, prompt, action)*. *sel_char* is used to select a menu item and *prompt* describes the action associated with a menu item. On selection, a menu item's *action* is executed. This results in one or more calls to routines in the testing interface or direct manipulation of the CUT through the test pins.

The interactive tester is executed after the execution of every test case in `Interactive` mode and after the execution of a failed test case in `Interactive_On_Failure` mode. For example, Figure 2 shows the response from the JK flip flop testbench when the test case `cew_Ntest(jk_preset, t_Q, '0')` is executed in `cew_Interactive_On_Failure` mode.

```
cew_Start_Menu
  cew_Menu_Item(a, PRESET, jk_preset);
  cew_Menu_Item(b, PRECLEAR, jk_preclear);
  cew_Menu_Item(c, SET, jk_set);
  cew_Menu_Item(d, RESET, jk_reset);
  cew_Menu_Item(e, TOGGLE, jk_toggle);
  cew_Menu_Item(f,
    DUMP (the values of t_Q and t_Q_bar),
    write(message, "The value of t_Q is ");
    write(message, t_Q);
    write(message, " The value of t_Q_bar is ");
    write(message, t_Q_bar);
    writeline(OUTPUT, message);)
cew_Stop_Menu
```

Figure 2: Interactive Test Script

```
FAILURE (Ntest) in test number 21
 Initial test trace = jk_preset
 Actual value = 1
 Expected value = 0
 Actual expression = t_Q
 Expected expression = '0'
 Source script line number = 142


 a: PRESET
 b: PRECLEAR
 c: SET
 d: RESET
 e: TOGGLE
 f: DUMP (the values of t_Q and t_Q_bar)
 q: Quit

Enter menu selection:
```

Figure 3: Interactive Test Menu

3 Implementation to Specification

Testbenches are ideal for impressing on students the need to read and adhere to specifications. Consequently, all modeling assignments are issued with a specification (in VHDL or natural language) and a *cew* script called the basic acceptance tests (BATs). Students can run the BATs as many times as they wish before submitting their work to the instructor. However, no credit is given for work unless it passes the BATs. In addition, the instructor runs a comprehensive testbench called the instructor's acceptance tests (IATs) to further establish quality.

We have automated the BATs/IATs testing strategy by developing scripts that handle code submissions, run BATs/IATs on all correctly submitted code and e-mail BATs/IATs summary statistics back to the students.

BATs and IATs are used systematically to expose an implementation that does not meet its specification. They reinforce the key engineering principle of implementation to specification.

4 Texts and Tools

There are two main approaches to text book selection for a first course in logic design using VHDL. The first approach is to select two books, one that covers the logic design theory and the other that covers modeling in VHDL. The VHDL primers by Pellerin, Yalamanchili, Mazor and Bhasker would be suitable for the two text approach [6, 8, 9, 10]. The second approach is to select a text that integrates the logic design theory with modeling in VHDL. The text by Dewey takes this integrated approach and it is the text we selected for our course [2]. Other integrated texts exist but they are aimed beyond a first course in logic design [3, 4, 5, 7].

With regard to the selection of a VHDL compiler, we considered VHDL design environments from Mentor Graphics (Unix), Aldec (Windows 95) and Green Mountain (DOS, Linux). Our selection criteria included compiler cost and the ease of integration of *cew* into the compiler's design environment. In addition, we considered the stability and ease of maintenance of the host operating system. For the next offering of our course, we plan to use Green Mountain's compiler under Linux.

5 Experience

The goals of this project were to modernize the laboratory component of our introductory logic design course by introducing system and component modeling using VHDL and, to do so in a software engineering context. We decided to use a phased approach. Initially, we had some breadboarding experiments and some modeling experiments in VHDL. It soon became apparent that our students found breadboarding more enjoyable and a better learning experience than writing VHDL models. To account for their learning preference, we designed experiments with both modeling and breadboarding elements. The following is an outline of the nine experiments we plan to use in the next offering of the course (all experiments with the exception of Lab 1 and Lab 4 were used in the first offering of the course):

Lab 1: Develop a behavioural description of a `Set` abstract data type (ADT).

Lab 2: Develop a behavioural, data flow and structural description of a two-bit comparator.

Lab 3: Develop a behavioural, data flow and structural description of a 4-input multiplexor.

Lab 4: Develop a structural description of an 8 bit ripple carry adder.

Lab 5: Develop a structural description for a 5 variable logic function using instances of the 4-input multiplexor from Lab 3.

Lab 6: Develop structural descriptions for a gated SR latch and a master slave pulse triggered SR flip flop.

Lab 7: Develop behavioural descriptions of edge triggered JK, D and T flip flops; develop a structural description of a 5-bit counter using instances of the T flip flop.

Lab 8: Develop a structural description of a sequence detector using instances of the JK flip flop from Lab 7.

Lab 9: Develop a behavioural and structural description of a traffic light controller using instances of the D flip flop from Lab 7.

The experiments were designed to introduce VHDL language constructs that support the behavioural, data flow and structural design abstraction hierarchy. Each Lab also has a breadboarding task (with the exception of Lab 1). Typically, it involves implementing the associated structural model using TTL components. Each Lab has set of BATs and IATs. Students are not allowed to start breadboarding until such time as their models pass both the BATs and IATs.

6 Concluding Remarks

This paper describes our experience in integrating VHDL into a first course in logic design. We discussed design verification issues and course support materials such as text books and design development environments.

We gleaned two key insights from this project. First, student preference suggests that breadboarding should be retained since it enhances their learning experience. Second, a BAT strategy greatly increases the quality of VHDL code submitted by students. The strategy itself is widely used in the software industry where programs are not accepted by quality-control departments until they pass basic acceptance tests.

In the main, student reaction to the course was positive. The course seems to have the right blend of being current (using VHDL) and being hands-on (breadboarding). We conclude by stating that in our experience, VHDL can be effectively integrated into a first course in logic design.

Acknowledgments

This project was supported by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada. We also acknowledge Warren Paterson-Welsh for his work in developing the CSCI 355 Lab Book.

References

[1] P. Walsh and J Uhl, "*cew*: Component Exerciser Workbench ," *Technical Report MC10*, Department of Computing Science, Malaspina University-College (in preparation) 1999.

[2] A. Dewey, *Analysis and Design of Digital Systems with VHDL,* ITP, 1997.

[3] C. Roth, *Digital Systems Design using VHDL* PWS, 1998.

[4] J. Armstrong and F. Gray, *Structured Logic Design with VHDL,* Prentice Hall, 1993.

[5] K. Skahill, *VHDL for Programmable Logic,* Addison Wesley, 1996.

[6] D. Pellerin and D. Taylor, *VHDL Made Easy,* Prentice Hall, 1996.

[7] J. Bhasker, *VHDL Synthesis Primer,* Star Galaxy Publishing, 1996.

[8] S. Yalamanchili, *VHDL Starters Guide,* Prentice Hall, 1998.

[9] S. Mazor and P. Langstraad, *A Guide to VHDL 2ed,* Kluwer Academic Publishers, 1993.

[10] J. Bhasker, *VHDL Primer 3ed,* Prentice Hall, 1999.