AUTOMATED EVALUATION OF STUDENT DESIGNS IN DIGITAL SYSTEM DESIGN LABORATORIES

Mark C. Johnson

Purdue University, West Lafayette, Indiana; Email: mcjohnso@purdue.edu

ABSTRACT

Rigorous evaluation of the functionality and originality of student designs in upper level digital system design courses is challenging at best. Courses such as the ASIC (Application Specific Integrated Circuit) Design Laboratory and Computer Design and Prototyping Laboratory require students to create simulations and hardware realizations of systems that have complex functional and performance requirements. Student demonstrations are usually inadequate to verify much more than a minimal subset of required functionality. Furthermore, it can be time consuming to verify in a simulation or FPGA prototype based demonstration that the design being demonstrated is in fact the student's design. In the ASIC design Laboratory and the Computer design and prototyping Laboratory, several automated processes have been developed to verify the correct functionality and originality of student designs. In this paper, examples are presented of design requirements to be verified, the means developed to evaluate the designs, and results of these evaluations.

1. INTRODUCTION

Thorough, timely, and consistent evaluation of student design work in any engineering domain is very challenging particularly as the design complexity increases. In the ASIC and computer architecture design arena, this is especially true because of the large number of subsystems and individual components in those systems. It has been widely reported (Bergeron, 2003) that in industry that design verification requires on the order of 60% to 80% of total design effort for a new ASIC or processor design. Although most student designs are very small compared to the latest Intel ™ or AMD ™ processor, the complexity of student designs have grown immensely thanks to modern EDA (electronic design automation) systems which have made it possible to prepare complex designs in a short time. Consequently, it is not surprising that the verification of student designs for grading purposes has become similarly complex. When the

ASIC design laboratory at Purdue University moved from schematic based design to language based circuit synthesis in 2001, we quickly realized that the evaluation process would have to be at least as automated as the design process itself. This led to the creation of the automated processes described in this paper. While the processes have evolved over time, most of what is presented here has been in use since 2001 for the ASIC design laboratory and since 2002 for the computer design and prototyping laboratory. In the remainder of this paper, we describe the design process followed by student, identify milestones in the design process that are amenable to evaluation, describe the evaluation processes that were automated and present sample results and observations on the use of these processes.

1.1 Student Design Flows

Students in the ASIC design and Computer design and prototyping laboratories follow a design flow illustrated in Figure 1. It would be more accurate to refer to this as an implementation flow since much of the creative work should occur prior to writing code for individual functional blocks, but "design flow" is the common industry terminology for the sequence of computer aided design processes required to take a design from concept to physical implementation. The design flow for each laboratory is nearly identical up until physical implementation. Please note that in practice the design flow involves considerable iteration and back-tracking, but these iterations usually occur within the framework illustrated by Figure 1.

Students start with a conceptual block diagram of a system to be implemented. The block diagram must be refined to the point where each block can be represented by common digital logic structures such as state machines, registers, and non-sequential combinational logic networks. The design flow illustrated in Figure 1 begins at this point. Students capture detailed block diagram information using a graphical hardware description language (HDL) entry tool (HDL Design ™ from Mentor Graphics Corp.) and they write VHDL (VHSIC Hardware Description Language) code to describe the logic required in each block and to describe test benches for each block and the entire design. Circuit simulation (using Modelsim ™ from Mentor Graphics Corp.) can be performed on source code for portions and the entire design to verify functionality. The design is then converted into a circuit netlist by use of logic synthesis software (Synopsys Design Compiler or FPGA compiler). The circuit netlist is simulated using the previously created test benches to verify functionality. This is necessary since poorly

written code can easily result in nonfunctional circuits even if the source code simulation behaved in the intended manner. On the next step, the ASIC and FPGA flows diverge. For an ASIC design, place and route software (SOC Encounter ™ from Cadence Design Systems) is used to produce an integrated circuit layout, check layout design rules, verify connectivity, and analyze the timing characteristics of the circuit. For an FPGA design, mapping software (Quartus II ™ from Altera Corp.) creates an FPGA configuration file based on the circuit netlist. Students can then download the FPGA configuration to a prototyping system to perform hardware tests.

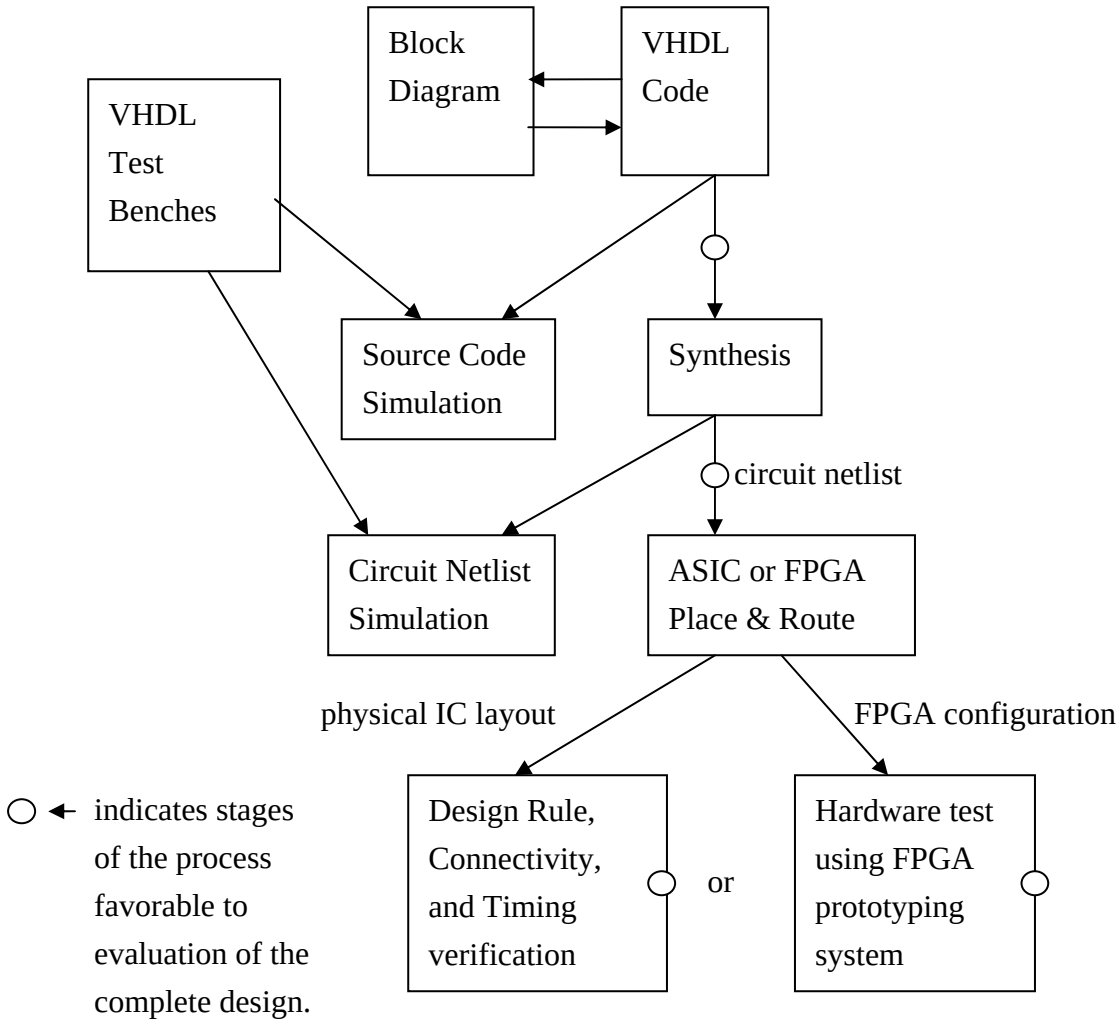

Figure 1: Design Flow in ASIC and Computer Prototyping Laboratories

1.2 Design Evaluation Requirements

In the previous design flow, there are numerous opportunities for design verification, but we focus on three times indicated in Figure 1 at which overall functionality can be verified: 1. the source code has been completed and simulated by the students, 2. the circuit netlist has been synthesized and simulated, and 3. the physical design is complete. Functionality requirements are given to students from a black box perspective much as one might delegate a portion of a project to a design team or a contractor. The required input and output signals are identified and functionality is given in terms of expected behavior of output signals in response to specified patterns on input signals. In the computer prototyping laboratory the target design is a microprocessor design, so an instruction set is also specified. In each laboratory, and at each stage of the design flow, it is necessary to verify that the design implements the complete specification. In addition, a way of quantifying the degree of compliance with the specification is necessary in order to assign a grade and give feedback to the student.

Design originality requires an assessment of the extent to which a student or team design is the result of independent effort. In the event of plagiarism, evidence must be collected that is at least strong enough to convince a technically capable neutral party. It is better still if the students involved find the evidence undeniable as well. It is not possible to precisely define the boundary between plagiarism and originality, but one can identify conditions for which plagiarism is difficult to deny, especially for relatively complex designs. The most extreme case is verbatim copying of the entire source code for a design. Almost as obvious, though more difficult to detect, are cases where the code structure is nearly identical, but the coding style and naming conventions are different.

2. IMPLEMENTATION AND SAMPLE RESULTS

In the ASIC, microprocessor, and embedded systems industries, delivery of systems that function as intended for an enormous range of input conditions require extensive verification of the design during all stages of the design process. Prior to physical implementation, this is accomplished by means of test benches, i.e., simulation code that generates input patterns to the design under test and checks the correctness of design outputs. While the verification needs for student designs are not quite as stringent, the use of test benches still provides a very efficient mechanism for verifying and re-verifying a design.

2.1 Evaluation of Simulated Student Designs

Given a clear design specification for a student design assignment, test benches can be written to verify simulated versions of student designs both in source and circuit netlist form. In industry, specialized verification languages have evolved to facilitate extremely extensive and efficient functional verification. However, VHDL includes syntax to facilitate generation of test waveforms and automated inspection of simulated circuit outputs. Rather than learn yet another language, both students and teaching assistants (TAs) prepare test benches in VHDL.

For grading purposes, each evaluation test bench is written to exercise a student design for a range of operating conditions (input patterns and timing) that can be inferred from the specification. Assertion statements in VHDL (Bhasker, 1995) are inserted liberally in the test bench to check for correct design behavior. Consider the following VHDL code fragment taken from an evaluation test bench for a simplified IIC serial interface design:

```
MDATA <= "11110000";
wait for 100 ns;

doStartCond(MSDA,MSCL);
wait for 1250 ns;

-- sending address 1111000 to slave with RW=0 (master read)
masterTransmit(MDATA,MSDA,MSCL);
-- Check for Ack/notAck from slave
probeACK(MACK_SAMPLE_1,MACK_SAMPLE_2,sda,MSDA,MSCL);
wait for 100 ns;

-- I2C: Check to see if Slave toggled the SDA line during
-- Height Time of the SCL
assert (MACK_SAMPLE_1 = MACK_SAMPLE_2)
  report "TEST 1-1: FAILED" severity ERROR;
assert (MACK_SAMPLE_1 /= MACK_SAMPLE_2)
  report "TEST 1-1: PASSED" severity NOTE;
```

Each test consists of a sequence of input pattern data interleaved with time delays and with inspection of output values. Commonly repeated input sequences and output checks are encapsulated in VHDL procedures such as `doStartCond, masterTransmit,` and `probeACK` above. The VHDL `assert` statement allows one to specify a conditional expression that describes the check to be performed, the message to be generated if the condition is not satisfied, and a severity level that is also reflected in the message generated by the `assert` statement. In the evaluation test benches, all of the test messages follow a standard format to facilitate later analysis of the test results.

In addition to test benches, a mix of python, perl, procmail, cron, and UNIX shell scripts were developed to automate execution of the evaluation test benches and subsequent analysis of results. Following is a sample of the results generated for a student design of the IIC bus. The percentage passed and the final grade represent a weighted sum of numerous individual test results. “SOURCE” refers to source code evaluation. “MAPPED” refers to evaulation of the synthesized netlist. Note: during execution of the grading script, the student’s source code is re-synthesized to ensure that the synthesized version to be tested corresponds to the source code that was submitted. In this particular case, the student’s source and synthesized versions behaved identically, but it is very common for the synthesized version to perform relatively poorly as a result of sloppy coding practices.

```
Sun Oct 16 21:06:05 2005

SOURCE RESULTS:
RESET  - Passed: 100.00%
TEST 1 - Passed: 100.00%
TEST 2 - Passed: 100.00%
TEST 3 - Passed: 100.00%
TEST 4 - Passed:  55.00%
TEST 5 - Passed: 100.00%
TEST 6 - Passed:  77.00%
SOURCE GRADE: 35.58/40
```

```
MAPPED RESULTS:
RESET  - Passed: 100.00%
TEST 1 - Passed: 100.00%
TEST 2 - Passed: 100.00%
TEST 3 - Passed: 100.00%
TEST 4 - Passed:  55.00%
TEST 5 - Passed: 100.00%
TEST 6 - Passed:  77.00%
MAPPED GRADE: 35.58/40
```

The grading outputs are deliberately non-descriptive with respect to what tests were failed. A correct version of the design is demonstrated to students after the design deadline to help them to understand where their design may have failed, but the burden is on the students to identify deficiencies in their own testing. Prior to the deadline, students receive a finite number of runs against the grading test bench by means of an automated queuing system. The student will run a turn-in script that emails a design submission to the course account. A .procmailrc file redirects the student submission to be processed by a script that unpacks the submission and places it in an evaluation queue. Another script is called at regular intervals by the UNIX cron utility to check the queue and run a grading script on the student submission. When this process completes, the student should receive an email back with results identical in format to what is shown above.

There are three primary motivations for this policy: 1. to push students to analyze specifications and test their designs as thoroughly as possible rather than tailoring their design to a particular test bench, 2. to discourage the passing of grading script information to future semesters, and 3. to emphasize recognition that in the “real-world” one will not be informed in advance of all of the situations to which a design will be subjected, user documentation notwithstanding.

When testing an entire microprocessor design in the Computer design and prototyping laboratory, a different method for checking correct operation is used but the automation of grading is similar. For an entire microprocessor, it is most efficient to compare a dump of simulated memory to the memory dump from a design that is known to operate correctly. A variety of test programs are executed on source code and synthesized versions of each

student's design. Some test programs are provided to students as examples, but many are not in order to encourage students to find ways to test their design as thoroughly as possible.

2.2 Evaluation of Design Originality

The system for evaluation of design originality has been documented previously (Johnson, 2004), but an overview is included here for convenience of the reader. Code plagiarism checkers have existed for many years (Culwin, 2001; Prechelt, 2000) but none supported hardware description languages such as VHDL or Verilog. However, a file compression based similarity measure (Bennett, 2003; Chen 2004) was shown to be effective in estimating the similarity of symbol sequences for a variety of applications including gene sequencing, C-code, and chain letters. The general concept is to perform file compression separately on three files: the first file to be compared, the second file to be compared, and a concatenation of the first and second file. The compressed file size for the concatenated source code is compared to the compressed file sizes of the individual source code files. If the compressed file size is close in size to the smaller of the compressed versions of the individual source files, one can draw the conclusion based on information theory that the two source files are nearly identical in terms of information content. A normalized similarity metric was defined so that results for numerous file pairs can be compared. A system of scripts was developed to compare all student submissions for a particular design, including past semesters. Histograms of similarity measures and a ranked list of the mostly closely matched student submissions are then used to identify student designs that require further inspection to determine if there is strong evidence of plagiarism. A small but non-negligible number of cases of plagiarism have been detected and successfully prosecuted as a result.

2.3 Evaluation of Hardware Implementation

In the ASIC laboratory, physical implementation of student designs is not feasible in one semester due to the time requirements of mask generation and fabrication. Select teams of students are given the opportunity to prepare designs for fabrication and test in subsequent semesters. However, for most students, the final product of this course is a picture of an IC layout for which the circuit connectivity, timing, and geometric design rules have been verified. No automation has been created for layout evaluation since connectivity, timing, and design rule check results can be inspected readily by laboratory TAs.

In the Computer design and prototyping Laboratory, the final product of each design is (in most cases, at least) a physical implementation using an FPGA based prototyping system. Over the course of a semester, students implement a 16 bit multicycle processor based on the MIPS processor described in the text Computer Organization and Design (Patterson and Hennessy, 1998), a 16 bit pipelined processor for the same instruction set, and small instruction and data caches that are integrated with the pipelined processor. Reasonably thorough testing (though not by industry standards) of the complete processor designs can be accomplished by executing a collection of test programs compiled for the specified instruction set. The test programs are designed so that correct program execution can be verified by examination of memory contents. However, some intermediate milestones in the course require evaluation of functional blocks, notably the arithmetic logic unit (ALU) and the data and instruction caches. Hardware implementations of functional blocks, especially the caches, can only be demonstrated manually for a very small set of test cases. A LabView ™ (from National Instruments) virtual instrument has been created to run on a Tektronix 720 log analyzer and pattern generator. The virtual instrument will download each of a collection of student designs to the FPGA prototyping board, use the pattern generator to drive the circuit inputs, and use the logic analyzer to capture outputs. The resulting outputs are compared to expected outputs. Results are compiled in a manner similar to the simulation based evaluations. The virtual instrument has been demonstrated, but it has not yet been used in the evaluation of student designs.

3. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented examples of the kinds of automation that can be used in digital hardware design courses to provide thorough and consistent testing of all student designs, encourage thorough testing by students of their own designs, validation of the originality of student designs, all with relatively little manual intervention on the part of TAs or the instructor. An automated FPGA hardware testing system has been created and demonstrated using LabView ™, but the system has not yet been used in the evaluation of student designs. Future work will focus on using the automated hardware testing for evaluation of actually student designs, and comparing the evaluation results to what would have been determined from live student demonstrations and from simulated test cases.

It should be noted that in scholarly literature, nothing seems to have been published on the subject of automated functional evaluation of complex student digital system designs. Extensive literature is available on the automated evaluation of many other types of student work including programming assignments and on research in functional verification of digital system designs. Both areas are related to this work, but documentation of the application of basic functional verification techniques to automated grading seems to be unique.

5. ACKNOWLEDGEMENTS

This work would have not been possible without the contributions both from industry and from numerous graduate and undergraduate teaching assistants who developed scripts and test benches currently used in the ASIC and Computer design and prototyping laboratories. By far, the largest contributors to the script based systems described in this paper were Shawn Davidson, now a design engineer for Intel and Curtis Watson who is now a graduate student at the University of Illinois. The LabView virtual instrument for automated evaluation of hardware designs was created by Evie Salim, currently a graduate student at Purdue University. Additional development and testing of the LabView virtual instrument was done by Siddharth Sen prior to his recent graduation from Purdue University. FPGA prototypes in the computer prototyping laboratory are done using FPGAs donated by Altera Corporation. Hardware testing in the computer prototyping laboratory is done using logic analyzers donated by Tektronix, Inc. The design flows used in the ASIC and computer prototyping laboratories would not be possible without the very generous and helpful university programs of Cadence Design Systems, Mentor Graphics Corporation, and Synopsys, Inc.

REFERENCES

Bennett, C.H., Li, M, Ma, B. (2003), “Chain Letters and Evolutionary Histories”, Scientific American, 76-81.

Bergeron, J. (2003), *Writing Testbenches: Functional Verification of Hdl Models*, Kluwer Academic Publishers, Norwell, MA.

Bhasker, J. (1995), *A Guide to VHDL Syntax*, Prentice-Hall, Englewood Cliffs, NJ.

Chen, X., Francia, B., Li, M., McKinnon, B., Seker, A. (2004), Shared Information and Program Plagiarism Detection, *IEEE Transactions on Information Theory,* **50**, (7), 1545-1551.

Culwin, F., MacLeod, A., Lancaster, T. (2001), Source Code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools, Technical Report SBU-CISM-01-01, South Bank University, London.

Johnson, M.C., Watson, C., Davidson, S., and Eschbach, D. (2004), Gene Sequence Inspired Design Plagiarism Screening, *ASEE Annual Conference and Exposition*, June 2004, Salt Lake City, UT.

Patterson, D.A., and Hennessy, J.L. (1998), *Computer Organization & Design,* Morgan Kaufmann, San Francisco, CA.

Prechelt, L., Malpohl, G., and Phillippsen, M. (2000), Jplag: Finding Plagiarisms among a Set of Programs, Technical Report 2000-1, Universitat Karlsruhe, Germany