Improving Student Performance Using Automated Testing of Simulated Digital Logic Circuits

Zachary Kurmas
School of Computing
Grand Valley State University
kurmasz@gvsu.edu

ABSTRACT

`JLSCircuitTester` helps automate the testing and grading of circuits built using digital logic simulators. With many simulators, the testing and grading of circuits is tedious and time consuming enough that students do not test their circuits thoroughly. `JLSCircuitTester` addresses this problem by simplifying the means by which users specify sets of input and expected output values. In addition, it automatically verifies that the circuit under test produces the correct output. The projects submitted during the pilot semester contained approximately half as many errors as the previous semester's projects. The automatic evaluation has also simplified the grading of those projects.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms

Verification

Keywords

Digital logic, JLS, simulation, testing

1. INTRODUCTION

One technique for helping students learn how CPUs work is to have them use a digital logic simulator to build a CPU and its major components [6]. Unfortunately, it is very time consuming and difficult to evaluate the correctness of all but the most basic circuits. Even moderately complex circuits or CPUs are too complex to be graded visually. In addition, none of the digital logic simulators that meet the constraints outlined in [4] (free, multi-platform, appropriate balance between simplicity and utility, etc.) also provide a means for automatically testing more than one set of input values at a time. As a result, grading student projects can be tedious and time consuming.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ITiCSE'08, June 30–July 2, 2008, Madrid, Spain.

Copyright 2008 ACM 978-1-60558-115-6/08/06 ...$5.00.

More importantly, the difficulties of thoroughly evaluating a circuit also discourage students from thoroughly testing their projects. Thoroughly testing a circuit requires simulating that circuit using a large number of values for the input pins and verifying that the output pins contain acceptable values. Similar to testing software, when an exhaustive test is not feasible, the sets of inputs must be chosen carefully so as to include "corner cases" and ensure that all components (the equivalent of code paths) are utilized. However, if a simulator requires that each set of input values be entered by hand, or if each output must be manually verified, students tend to take the time to run only a few (in our experience, well under 10) tests.

`JLSCircuitTester` addresses both problems by

- providing a means for students and instructors to easily specify multiple sets of input values and the corresponding expected outputs, and
- automatically simulating the circuit under test using each of the input sets and reporting any unexpected output values.

During the first semester of use, the percentage of projects with major errors decreased from about 75% to 33%. During the second semester of use, the error rate remained around 33%; however, in contrast to previous semesters, several of the imperfect submissions had their errors clearly documented. In addition, the ability to automatically test circuits reduced grading time. (Some of the reduction in grading time resulted from submissions having many fewer mistakes.)

Currently, `JLSCircuitTester` works only with the JLS digital logic simulator [4]. However, we designed the tool independently of any particular simulator and coupled it loosely with JLS. We look forward to adding support for additional digital logic simulators.

2. BACKGROUND

In a typical Computer Organization course, students learn how to use basic logic gates (e.g., AND, OR, NOT) to build the components of a CPU (adders, multiplexers, registers, ALUs, etc.). These components are called *digital logic circuits.* Figure 1 shows an example circuit called a half-adder that adds two bits. The interface between circuits and the outside world are called "pins". (Picture the pins that stick out from a typical computer chip.) Each pin takes on one of two values: 1 or 0. These input and output pins are analogous to parameters and return values for methods. In

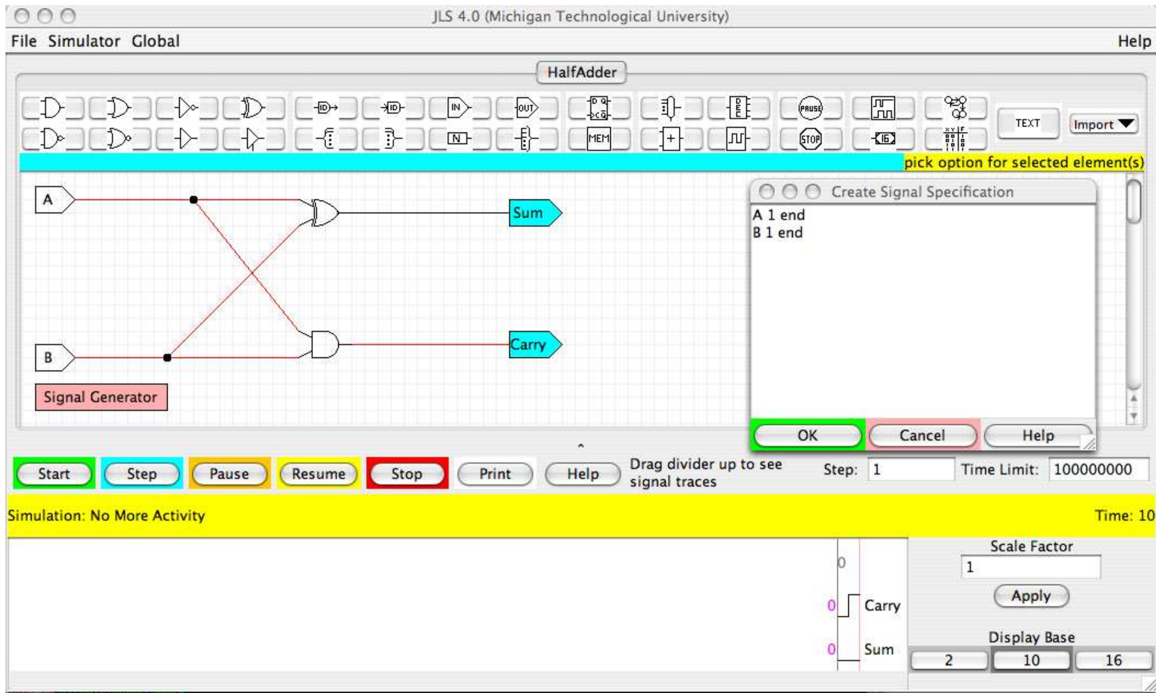

Figure 1: Testing a half adder using JLS

Figure 1, the input pins are named `A` and `B`. The output pins are named `Sum and Carry`.

One of the most effective ways to teach students how a CPU and its components work is to have them build examples. For reasons Wolffe, et al. outline in [6], students often build digital logic circuits using a digital logic simulator: a tool with which students can add and connect digital logic gates and see how the resulting circuit operates.

`JLS`[1] is one of many digital logical simulators. We use it because (1) it is written in Java and can, therefore, run on many platforms, (2) it is free, and (3) it provides an excellent tradeoff between complexity and usability. The most powerful simulators allow (or even require) the user to specify the electrical properties of the transistors and wires. The learning curve on these simulators is usually too steep for a typical Computer Organization class.[2] In contrast, some very basic simulators emphasize simplicity and lack the ability to bundle wires together. Such simulators are useful for exploring the basics of digital logic; but, are not practical for building complete CPUs. In [4], Poplawski further discusses the criteria that determine how well-suited a particular digital logic simulator is for a typical Computer Organization or Computer Architecture classroom.

3. MOTIVATION

To test a circuit using JLS, the user must (1) add an element called a *Signal Generator* to the circuit under test, (2) type text into the *Create Signal Specification* window to describe the desired input values, (3) run the simulator, and (4) compare the values on the output pins to the expected values. (See Figure 1.) This process works very well when examining the circuit closely and looking for specific problems (i.e., "debugging" the circuit). However, it is not as well suited for running a thorough test of a circuit. Testing a circuit like the half adder, which has only four possible input combinations is manageable, although somewhat tedious.

Consider a more complex circuit, like the 16-bit signed adder shown in Figure 2. In this circuit, the inputs `InputA` and `InputB` each represent a group of 16 pins.[3] Together with the 1-bit `CarryIn`, there are 2^{33} possible input combinations. It is clearly not practical to test all possible combinations. Fortunately, software engineering principles allow us to choose a set of a few dozen combinations and be fairly confident that a circuit that passes those tests is correct.

Pedagogical motivation: The problem is that most students lack either the time or the patience to set up more than a few tests by hand. As a result, these students submit under-tested, incorrect circuits. In addition to receiving a low grade, students who turn in incorrect circuits miss the opportunity to discover and correct misunderstandings about the construction of the circuits they are building.

Instructional motivation: The tedium of running tests by hand is also problematic for instructors. While it may be reasonable to require a student to run four tests on a half adder (or even 10 to 20 tests on a signed adder) by hand; it is not practical for an instructor to repeat this process for each of the students in a course. Thus, in addition to improving learning, automating the testing process will save the instructors time.[4]

[1] JLS stands for *J*ava *L*ogic *S*imulator.

[2] Such simulators may, however, be appropriate for Electrical or Computer Engineering courses.

[3] Figure 2 does not show the size of the input.

[4] JLS does have a batch mode that instructors can use to partially automate the evaluation process. Its main limitation is that the output is designed for humans to read and does not lend itself well to parsing. Thus, the instructor can automatically run the simulator, but must scan the output for mistakes.

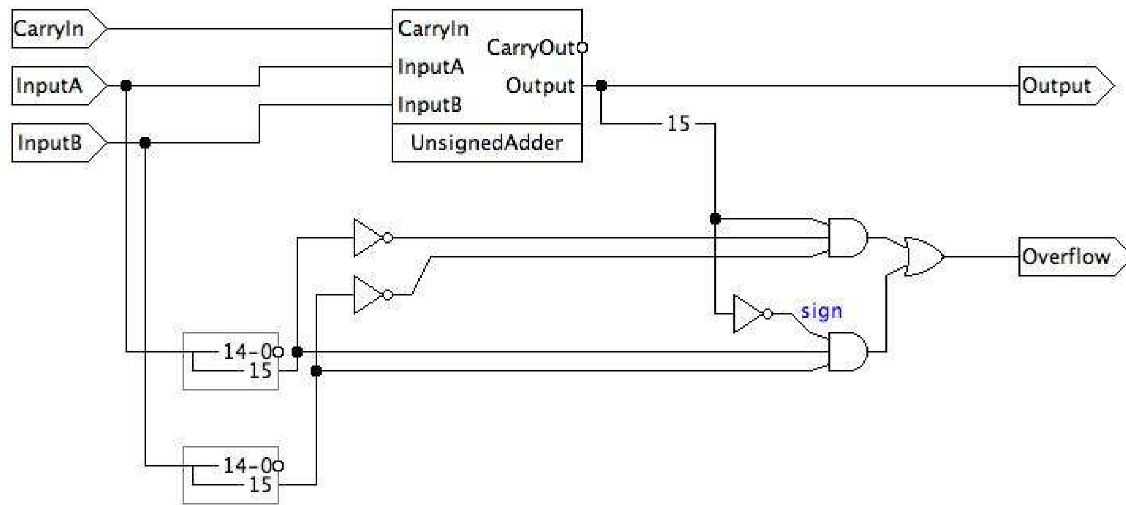

Figure 2: A signed adder with overflow detection

4. RELATED WORK

In [1], Edwards discusses how using test driven development to promote "reflection-in-action" (thinking critically about the program rather than making random changes and hoping the errors go away) can improve students' programming ability. He then discusses a tool called Web-CAT that supports his teaching approach. Our tool provides for circuit design several of the features that Web-CAT provides for program design. In particular, `JLSCircuitTester` allows students to define their own tests, and automatically determine whether their circuits pass their tests. It does not automatically assign a score to a circuit.

5. THE TOOL

`JLSCircuitTester` is a command-line tool written in Java that automates the testing of simulated digital logic circuits. Users specify the circuit to test and the list of tests to run. The tool then simulates the operation of the circuit once for each set of input values and reports any tests for which the observed and expected output values differ.

5.1 Input specification

Users generate text files listing (1) the values to be placed on the input pins and/or (2) the values to which any registers and memories are to be initially set. Figure 3 shows a file that thoroughly tests the half adder shown in Figure 1.

Notice that `A` and `B` each have a list of two values. The tool produces four tests to represent the four unique combinations of elements from these lists. Consequently, the output pins have lists of four values each.

Figure 4 shows an example test file that can be used to test a 16-bit signed adder. The `NAMED_VALUE_LISTS` section allows users to specify lists of numbers that can be referenced by multiple tests. Notice that the user can place simple expressions in the lists of numbers, thereby making it easier to specify "corner cases" (e.g., values near a power of 2). As with the previous example, the tool produces one test for each combination of input values; thus the section of code labeled `no_overflow` specifies 288 tests, and the section labeled `overflow` specifies an additional 578 tests.

```
# This file tests a basic half adder with
# one-bit inputs named "A" and "B", and
# one-bit outputs named "Sum" and "Carry".

BEGIN test1
    INPUTS
      A [ 0, 1 ]
      B [ 0, 1 ]
    OUTPUTS
      Sum [ 0, 1, 1, 0 ]
      Carry [ 0, 0, 0, 1 ]
```

Figure 3: Test file for half adder

Programmed Value Lists: Instead of listing all the desired input values by hand, users can specify an algorithm to generate a list of input values. For example `RANGE(0, 10, 2)` generates the list of even integers between 0 and 10.

Flexibility: `JLSCircuitTester` is written in Java. Each test is represented by an instance of the `InputSet` class. An `InputSetLoader` object is responsible for creating `InputSet` objects based on the input file. Users can implement their own test specification formats (e.g., XML) by implementing their own subclass of `InputSetLoader`.

5.2 Output specification

The user may specify a test's expected output in one of two ways: (1) explicitly list the desired output value when specifying the input values, or (2) provide a Java class (called an *Output Set Generator*) to calculate the output value. Figure 3 lists the desired output values in the `OUTPUTS` section of the test file. This technique is useful when the number of tests is small and the set of desired output values can be easily enumerated.

In contrast, the input file in Figure 4 specifies that the class `UnsignedAdderOutputSet` is responsible for calculating the correct output value. This technique is useful when the number of tests is large. In this case, it would be difficult to type the list of 288 expected output values for the

```
# Name of Java class that computes the correct answer
OUTPUT_SET_TYPE SHARED UnsignedAdderOutputSet

NAMED_VALUE_LISTS
  # The sum of any two smallPositive integers will not cause an overflow.
  smallPositive [ 0, 1, 2, 3, 4, 5, 10, 15, 16, 17, 30000, 2^15 - 1 ]
  allPositive   [ smallPositive, 2^15, 2^15 + 1, 2^15 + 16385,
                  2^16 - 2, 2^16 -1 ]

# These tests should not produce any overflow
BEGIN no_overflow
  INPUTS
    InputA smallPositive
    InputB smallPositive
    CarryIn [ 0, 1 ]

# These tests may produce overflow.
BEGIN overflow
  INPUTS
    InputA allPositive
    InputB allPositive
    CarryIn [ 0, 1 ]
```

Figure 4: Test file for a 16-bit signed adder

`no_overflow` tests without making any mistakes. Instead, the user provides a Java class that reads the input values, adds the values of inputs `A` and `B`, checks for overflow, and sets the output pins accordingly.

5.3 CPU testing

One typical Computer Architecture project is to have students use a digital logic simulator to construct their textbook's example CPU. Such CPUs are typically designed primarily to illustrate the key aspects of CPU construction. Therefore, in most cases, they contain only a minimal set of features and are considerably simpler than real CPUs. Our tool provides support for testing such CPUs.

First, users need a way to produce machine code. The textbook CPUs often have a machine language that is simple enough to code by hand; however, such "human assembly" is tedious and error prone. With our tool, instructors can add an assembler to the output set generator.

Second, users need a way to specify the desired output. In the case of CPUs, the "output" is the final state of the registers and RAM. Listing the final state of RAM could be quite challenging for some programs. With our tool, instructors can use a CPU simulator as the output set generator.

We provide an output set generator for testing MIPS-like CPUs (such as the ones presented in both the Patterson and Hennessy [3] and Harris and Harris [2] textbooks). Instead of a file similar to Figures 3 and 4, users pass as input a MIPS assembly program. The output set `JLSMipsCPUTester` then:

1. Assembles the code using the assembler built into the MARS MIPS Simulator [5].
2. Loads the machine code into the RAM of the CPU under test.
3. Uses JLS to simulate the running of the CPU under test.
4. Simulates the running of the assembly code using MARS.
5. Compares the final state of both simulations and reports the results.

Thus, students can test their CPU using almost[5] any MIPS assembly code.

6. RESULTS

We piloted `JLSCircuitTester` in a Computer Architecture course that assigns three projects: (1) implement an ALU that can add, subtract, and multiply signed integers and detect overflow; (2) implement the single-cycle CPU presented in [2] and [3], and (3) implement microcode for the multi-cycle CPU presented in [2] and [3]. Historically, students would submit these projects with an unacceptable number of mistakes. Therefore, students are required to resubmit their projects until they contain only minor flaws.

In Winter, 2007, approximately 75% of the initial submissions contained serious mistakes. We piloted our tool in Fall, 2007. That semester, fewer than 33% of the initial submissions contained serious mistakes. In Winter, 2008, the percentage of submissions with serious mistakes remained around 33%. However, several of the incorrect submissions came with lists of known problems. Thus, although these students were unable to fix the problems by the submission deadline, they had tested their circuits well enough to know what the problems were.

Although the project scores improved, scores of those exam questions related to ALU and CPU design did not improve. Fall 2007 students performed worse overall than Winter 2007 students. Winter 2008 students performed better than the previous two semesters' students. We suspect these results primarily reflect differences in composition of each class.

This analysis is informal. Each class contained approximately 8 teams of two students. A more rigorous analy-

[5] The student's CPU may not implement every MIPS instruction.

sis would require withholding JLSCircuitTester from students. Given that the improvement in project scores suggests that our tool is beneficial, conducting a more controlled experiment does not appear to be in the best interest of the students.

We do not have a record of grading times prior to the introduction of our tool. However, anecdotally, grading the projects in Fall 2007 and Winter 2008 was considerably easier and less frustrating than Winter 2007. (In other words, we can't prove that JLSCircuitTester saves time, but it sure feels like it does.)

7. FUTURE WORK

Presently, our tool directly supports the testing of only those CPUs that can also be simulated by the MARS MIPS simulator. Such CPUs must use the MIPS machine language and must utilize registers in a manner similar to MIPS (`$r0` fixed at 0, `$r1` reserved for the assembler, etc.). We expect that our next step will be to add support for the testing of CPUs presented in several different textbooks, thereby making our tool useful to more instructors.

The current version of our tool works only with the JLS digital logic simulator. We plan to add support for several additional compatible simulators, then move the core functionality into a library that will allow others to easily add support for new digital logic simulators. Currently, to be compatible with JLSCircuitTester, a digital logic simulator must either (1) have a Java API, or (2) have a batch mode with an output format suitable for parsing (as opposed to an output format designed primarily for reading by users).

During the past two semesters, we provided students a lot of help in generating thorough test cases. In the long term, we would like students to be capable of generating their test cases with minimal help. Reaching this goal requires that students learn testing techniques well before reaching a senior-level Computer Architecture course. Several of us in the School of Computing are looking to increase the coverage of testing in courses throughout the curriculum.

8. AVAILABILITY

JLSCircuitTester is available on the web at http:/www.cis.gvsu.edu/~kurmasz/JLSCircuitTester. In addition, the web page contains several example assignments.

Acknowledgments

We would like to thank David Poplawski for his support of this project — especially his advice and his work modifying JLS to make it integrate better with JLSCircuitTester. We also thank Pete Sanderson for his efforts making MARS integrate better with JLSCircuitTester. Finally, we thank Christian Trefftz, Paul Jorgensen, Steve Salerno, and Robert Adams for their contributions.

9. REFERENCES

[1] S. H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, March 2004.

[2] D. M. Harris and S. L. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, 2007.

[3] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 3 edition, 2005.

[4] D. A. Poplawski. A pedagogically targeted logic design and simulation tool. In *WCAE '07: Proceedings of the 2007 workshop on Computer architecture education*, pages 1–7. ACM, 2007.

[5] K. Vollmar and P. Sanderson. Mars: an education-oriented mips assembly language simulator. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 239–243. ACM, 2006.

[6] G. S. Wolffe, W. Yurcik, H. Osborne, and M. A. Holliday. Teaching computer organization/architecture with limited resources using simulators. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 176–180. ACM, 2002.