Logisim: A Graphical System for Logic Circuit Design and Simulation

CARL BURCH
St John's University, Collegeville

Logisim enables students in introductory courses to design and simulate logic circuits. The program's design emphasizes simplicity of use, with a secondary goal of enabling design of sophisticated circuits. This motivates a two-tiered system, where users can move to the second tier by selecting a menu option.

Users draw circuits of logic gates using the toolbox model popular in drawing programs. The circuit automatically propagates circuit values through the circuit; by selecting the appropriate tool, users can toggle switches to see how the circuit behaves in other situations. In the advanced tier, the user can treat circuits as black boxes within larger circuits, enabling the simulation of hierarchical designs. The author has successfully drawn and tested a simple 8 bit CPU using the program.

The program has proven useful in a variety of introductory courses, from a nonmajors survey course to a sophomore-level systems course. Students find Logisim simple to follow, and find the laboratories designed around it useful in reinforcing the circuit concepts from class.

In this article, we identify and compare a variety of systems similar to Logisim, we explore Logisim's features in detail, and we examine its use in class assignments.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization—General**]: Modeling of Computer Architecture; I.6.5 [**Simulation and Modeling**]: Model Development; K.3.1 [**Computers and Education**]: Computer Uses in Education

General Terms: Design, Documentation

Additional Key Words and Phrases: Circuit simulation, digital logic, education

1. MOTIVATION

The computer science department of the College of St Benedict and St John's University (CSB|SJU) regularly offers three courses in which logic circuits play a major role. Circuits are one of the first units in both of our survey courses (a nonmajors course and a first-semester majors course); and circuits make up a large fraction of a sophomore-level systems course. All of these courses have laboratory sections where students complete structured assignments during scheduled meetings in computer laboratories. Due to the significant role of circuits, some laboratory assignments must involve hands-on practice with logic circuits.

Author's address: Computer Science, St John's University, Collegeville MN 56321; email: cburch@cburch.com.

Though an important part of the courses, logic circuits are not so prominent that students can spend much time building circuits on breadboards. The objective is that students know the basics of how gates combine to compute complex functions; breadboards require additional knowledge superfluous to this objective. Hence, we must use a simulator.

The capacity we need from the simulator is relatively simple and does not justify the purchase of commercial software. We would prefer the software to work on both Unix and Windows, as constrictions set by other course material dictate that two courses meet in a Windows laboratory and one in a Unix laboratory. A thorough search yielded no alternatives that met these criteria satisfactorily.

Thus this author began his own, dubbed *Logisim*. The emphasis of this project is to develop an easy-to-learn program for developing and simulating circuits of logic gates. The greatest emphasis of the project is on ease of learning, so that students in our introductory courses, who spend only two laboratory sessions with the program, meet as few problems learning the software as possible. For these students, sophisticated ability is not important.

But the systems students do build relatively sophisticated circuits, so a secondary emphasis is on the power of the circuits built with the program. They need the capability to build circuits illustrating the internals of a simple CPU. This requires the ability to "black-box" smaller circuits as components within larger circuits.

Logisim is a Java application, so that it can run on both Windows and Unix workstations. The program is actually one piece of a larger suite of Java programs developed locally for other modules of our introductory courses (numeric representation, assembly programming, and finite automata).

For those who wish to use Logisim in their own classes, Logisim is publicly available on the Web. There is no charge for its use.

`http://www.cburch.com/proj/logisim/`

The author would appreciate hearing of its use at other institutions. If feedback indicates that Logisim is proving useful, the author would welcome the opportunity to extend its capabilities further.

2. COMPARISON WITH EXISTING PROGRAMS

Many programs take an approach similar to Logisim's. To get an idea of how the alternatives compare, the author tried a large sample of inexpensive alternatives (many derived from Wolffe et al. [2002]). Figure 1 lists all the sampled programs. All are similar to Logisim in that they provide a graphical toolbox interface for composing and simulating logic circuits.

Omitted from comparison are the similar but more expensive Windows programs, Digital Works 3.0 and LogicWorks, both of which implement all or nearly all features listed [Mecanique 2001; Capilano 1999]. Also omitted are design programs that do not permit simulation (like DesignWorks [Capilano 2001]) and simulation programs where the circuit is specified in text files (like *esim* [Miller and Squire 2000]).

program	reference	platform	type
Circuit Builder	[Karweit 2000]	applet	freeware
Digital Circuit Simulator 1.0	[Zidar 2001]	Windows	shareware
Digital Simulator 1.1	[Knaian 1994]	Windows	shareware
Digital Simulator 3.2	[Herz 1998]	Windows	freeware
Digital Works 2.0	none[a]	Windows	freeware
DigSim 1.00b3	[van Rienen 1996]	applet	open source
DigiTCL 0.3b0	[Craig 1997]	Unix/Windows	open source
EasySim 2.04	[Research Systems 2001]	Windows	commercial
Logic Simulator	[Gordon College 2001]	applet	freeware
LogicSim 3.0b	[Masson 1996]	Macintosh	freeware
LogicSim 2.0β	[Tetzl 2001]	JVM	freeware
Logisim 1.04	–	JVM	freeware
Multimedia Logic 1.2c	[Softronics 1997]	Windows	freeware
Probe 0.97	[Boothe 1999]	applet	freeware
xLogicCircuits	[Eck 2000]	applet	open source
Simcir 1.2.1	[Arase 2000]	applet	open source

[a] Digital Works 2.0 predates the current commercial version of Digital Works [Mecanique 2001]. A Web search revealed several class pages enabling the download of version 2.0, but no permanant Web host to cite.

Fig. 1. Graphical logic circuit simulators included in comparison.

	Logisim 1.04	LogicSim 3.0b [Masson 1996]	Digital Works 2.0	xLogicCircuits [Eck 2000]	EasySim 2.04 [Research Systems 2001]	LogicSim 2.0β [Tetzl 2001]	DigSim 1.00b3 [van Rienen 1996]	Digital Simulator 1.1 [Knaian 1994]	Multimedia Logic 1.2c [Softronics 1997]
custom wire paths	yes	yes	yes	yes	yes	yes	yes	yes	yes
wires indicate values	yes	yes	no	yes	yes	yes	yes	no	no
variable-input gates	yes	no	yes	no	yes	no	yes	no	no
clock component	no	yes	yes	yes	yes	yes	yes	yes	yes
flip-flops	yes	yes	yes	yes	yes	yes	yes	yes	yes
custom components	yes	yes	yes	yes	no	yes	no	no	no
save circuit	yes	yes	yes	yes	yes	yes	yes	yes	yes
print circuit	yes	yes	yes	no	yes	yes	no	no	yes

Fig. 2. Comparison of graphical logic circuit simulators.

Figure 2 compares the features of the evaluated programs, which are ordered based on an approximate evaluation of their suitableness for a sophomore-level computer organization course. The table omits the evaluated programs judged least suitable for this purpose. With each program, the author of this article attempted to determine whether it addresses each of a selection of important features.

Custom wire paths. The program's interface allows the user to specify the wire paths. This is in contrast to simpler interfaces where the user must add wires by dragging between two component connections; the wire usually appears simply as a straight line between the component connections. While this interface is simple and intuitive, the reduced flexibility leads to ugly circuits and severely limits the size of circuits that one can practically build in the system.

Wires indicate values. During simulation, the program indicates the value that each wire is carrying. Such programs usually draw a wire carrying low voltage with one color and a wire carrying a high voltage with another color.

Variable-input gates. The program incorporates AND and OR gates that can accept more than two inputs.

Clock component. The program includes a clock device that toggles between outputting high and low voltage repeatedly.

Flip-flops. The program includes a flip-flop component permitting the simple development of sequential circuits.

Custom components. The user can design the internals of components to appear repeatedly within larger circuits.

Save circuit. The user can save a circuit to a file.

Print circuit. The user can print a circuit to the printer.

In some situations, an abundance of features is worth less than a short learning curve. Such is the case with our survey courses, when students complete only one or two laboratory assignments using the program; time spent learning the program is wasted laboratory time. Based on the author's subjective judgment (certainly not unbiased), none of the programs have a more intuitive interface than Logisim for these basic features. Most are more complex, some unsuitably so for these class requirements.

Nonetheless, compared to most of the evaluated programs, Logisim boasts a much wider set of features. In particular, the ability to build and edit hierarchical circuits is a rare feature, but one that is necessary if the program is to scale up to a full-semester course.

Of the evaluated programs, only two, besides Logisim, have a feature set large enough that students could build a reasonably sophisticated CPU circuit using it: Digital Works and LogicSim. But these programs have three major problems.

—The authors of Digital Works and LogicSim apparently do not intend to release any future noncommercial versions. (Digital Works has gone commercial, and no new versions of LogicSim have appeared for over five years.) The author of Logisim intends to continue creating free versions of Logisim.

—Digital Works and LogicSim are limited to their specific platforms (Windows and Macintosh, respectively). Logisim should work on any platform that supports Java 1.2 or later, including Unix, Macintosh, and Windows.

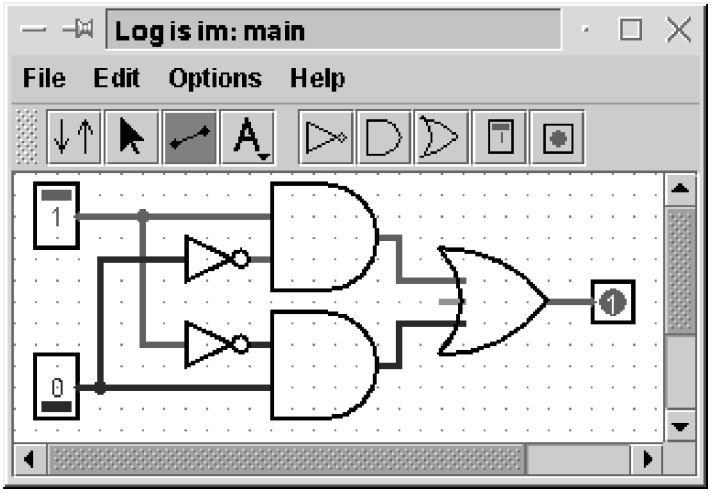

Fig. 3. A simple circuit built in Logisim.

—In this author's subjective judgment, Logisim requires less training than Digital Works or LogicSim for students who wish to use the program only for simple combinational circuits.

In their favor, Digital Works and LogicSim both have some significant features that Logisim does not share, which may make them more suitable for some topics in a full-semester organization course.

While Logisim is not unique, it appears as easy to use as other programs, while boasting a large feature set, a simple interface, a small price tag, and compatibility across many platforms. Digital Works 2.0 and LogicSim are good, inexpensive alternatives for institutions whose laboratories are compatible.

3. THE LOGISIM INTERFACE

We now examine in detail how we can use Logisim. Since Logisim works at two levels, we first describe the features of the beginners' level, and then the features added when the user selects the advanced tools option.

3.1 Beginning Features

Figure 3 contains a screen shot illustrating what beginning Logisim users see. Logisim uses the toolbox concept common in drawing programs. Starting from the left side of the toolbar, the tools available to the beginner are the simulation tool, the selection tool, the wiring tool, and the label tool. Continuing rightward, there are tools for adding NOT, AND, and OR gates, as well as tools for adding input switches and output LEDs.

A student building a circuit typically first lays down the components of the circuit. The student selects the tool for the gate to add. As the mouse moves across the canvas area, a ghost appears where the gate would be placed when the student clicks the mouse. All gates are snapped to a strict grid.

The second step is adding wires. The simplicity of Logisim's wiring tool may be its strongest feature. Students can drag anywhere on the canvas to place a wire. All wires must be horizontal or vertical, and they must fit exactly onto the grid. The program automatically detects and draws the connections as the user adds each wire: Anything hit by the new wire's endpoints are connected, as are any existing wires that terminate at the new wire. (This behavior is more intuitive than it sounds.) In attaching to AND and OR gates, all five grid points on the left side are open for attachments. (For circuits with many gates, Logisim includes a "Small Gates" option. When selected, the gate tools add smaller gates allowing only three connections. Figure 6 contains these gates.)

The final step is simulation. Even as the student adds wires, he or she can see the values flowing through the wires. (Bright green indicates a true value, while a dark green indicates a false value; wires not yet connected to a known source are a pale blue.) Once the student selects the simulation tool, a click on any switch toggles its output value, with immediate propagation through the circuit.

When the student adds something by mistake, a right-click (or command-click) on the circuit component will bring a pop-up menu, including a delete option. Also, the selection tool permits selecting components within a rectangular region of the circuit. The user can cut, copy, and paste sets of components via the edit menu.

The file menu gives students the ability to save their circuits for future editing and to print their circuits for attachment to their laboratory reports.

3.2 Advanced Features

For nonmajors, who represent about half of the users of Logisim at CSB|SJU, the above is all of Logisim that they see. But other students, who want to develop sequential and hierarchical circuits, will select the advanced tools option in the options menu. This enables Logisim's more complex tools.

Figure 4 contains a screen shot illustrating a more complex circuit built using Logisim. In fact, this screen shot is of an implementation of the simple 8-bit CPU with 32 bytes of memory that our nonmajors learn to program. It illustrates the practical limits of the current version of Logisim, but it goes well beyond what our students do in their classwork.

The advanced tools option adds tools for a few more circuit components: D flip-flops, constant-value sources, and tristate gates. But the most notable additions that come with selecting advanced tools are the subcircuit tool and the project menu, which enable hierarchical circuits.

Logisim views a circuit as a set of "subcircuits," each with a user-defined name. The project menu lists all subcircuits in the currently open circuit. Selecting a subcircuit's menu item brings up that subcircuit for editing. The default subcircuit (which indeed is all the beginning user ever sees) is titled "main." Users can add new subcircuits via "add new subcircuit" under the project menu. This enables them to design useful pieces of a larger circuit, such as the multiplexers or the ALU used in Figure 4.

To use a subcircuit as a black box within another subcircuit, the user would select the subcircuit tool on the far right side of the toolbar. The user selects the

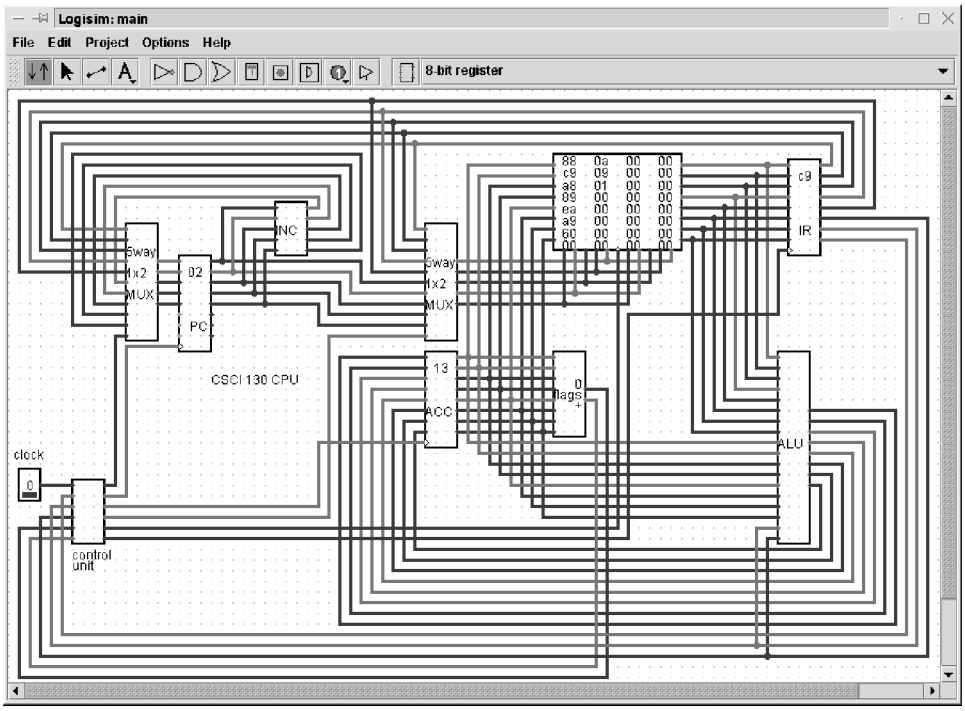

Fig. 4. A more complex circuit in Logisim.

subcircuit to add from the list box in the toolbar (listing all subcircuits within the current project).

A subcircuit component appears as a rectangle with pins on the left and right sides. The pins on the left side correspond to the subcircuit's switches (in the same top-down order), while the pins on the right side correspond to the subcircuit's LEDs (also in the same top-down order). This simple inferred structure from the subcircuit simplifies the circuit development process, though at the expense of less flexibility in how the subcircuit appears. It would be nice, for example, for a XOR gate subcircuit to appear as a XOR gate when it is used. But the added complexity both to the user and to the program conflicts with the design goals of Logisim.

When the circuit propagates values into a subcircuit component, it propagates those values through the subcircuit, and the values that this subcircuit's LEDs take on are the values propagated from the subcircuit component onward. (The subcircuit maintains values from previous propagations, so that a sequential subcircuit works well, even if it appears several times within the same circuit.)

4. EXPERIENCE WITH LOGISIM

At its inception in October 2000, Logisim had only fundamental features; its only advantages were its simple user interface and its compatibility with both

Unix and Windows. The restricted feature set was adequate for the level of usage in our introductory survey courses.

The instructor of our sophomore-level systems course noticed its potential, however, and began using it in that course. This usage motivated the addition of new features, lifting it to its current version. By the fall 2001 semester, Logisim reached version 1.0, sporting project-management features to enable building and debugging larger circuits. Also added were the selection tool, a more complete help system, and many minor enhancements. Bugs discovered and fixed during its use in the fall 2001 semester pushed Logisim to version 1.04.

Logisim has proven a boon for teaching logic circuits, simple enough for students to dive into practicing concepts in an immediate-feedback environment. At CSB|SJU, we now regularly use Logisim in three courses.

CSCI 130. A nonmajors survey of computing. One topic of this course is Boolean function computation with logic gates. Logisim is at the center of one laboratory assignment. In it, students practice deriving truth tables from circuits and building combinational circuits from truth tables.

CSCI 150. The first-semester course in our computer science curriculum, a more intensive computing survey. A quarter of the course emphasizes computer architecture; two of its labs center on Logisim. The first is similar to the CSCI 130 lab, giving students the opportunity to practice converting between truth tables and logic circuits and simplifying Boolean expressions. The second lab, described below, introduces students to sequential circuits through a systematic construction of a circuit inspired by vending machine controllers.

CSCI 210. A sophomore-level systems course that continues the concepts of CSCI 150 for another quarter-semester. In this course, one laboratory session uses electronics breadboards to give students the idea of connecting electronics. Experimenting with the more sophisticated circuits seen in the classroom is impractical on breadboards, however. So the next three laboratories use Logisim when they successively build a memory subsystem, an ALU, and a control unit to form a simple CPU based on Logisim's primitive components.

As an example of what students do with Logisim at CSB|SJU, we examine the second CSCI 150 lab, which introduces students to sequential circuit design. In the classroom, students see how to build both an R-S latch and a D latch, but the only time they see its use is in a brief overview of how to build a register system using D latches. The lab we examine introduces students to sequential circuit design.

The laboratory supposes the student has been contracted by FizzlePop, Inc., a hypothetical manufacturer of soft drinks for price-conscious consumers, to design a controller for their vending machines. FizzlePop, Inc. manufactures only one type of soda (hence buttons on the machine to select a soft drink are superfluous) and sells it for 15 cents.

The controller receives two Boolean inputs (in, amt): $(0, x)$ when nothing is being deposited; (1, 0) when the user deposits a nickel; and (1, 1) when the user deposits a dime. The controller is to have two Boolean outputs $(out, what)$: (0, 0) when the machine is to perform no action; (1, 0) when the machine is to dispense

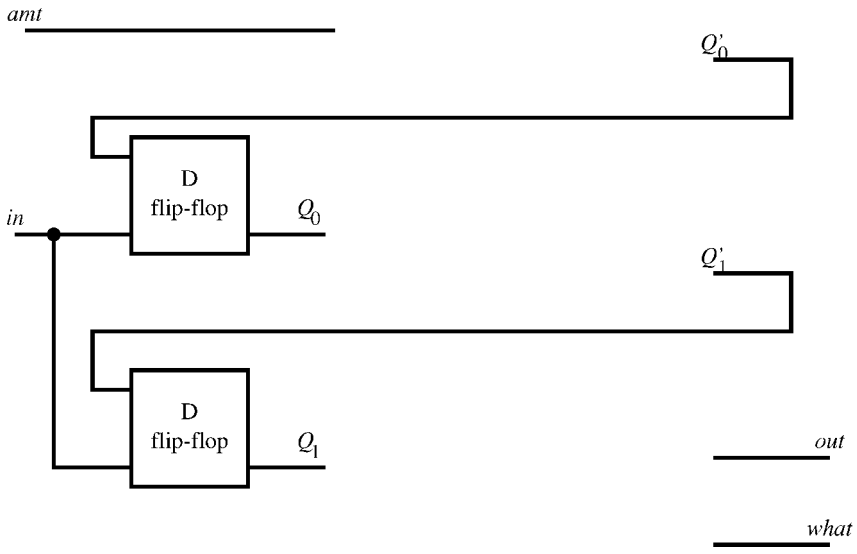

Fig. 5. Template for FizzlePop vending machine controller.

a soda; and (1, 1) when the machine is to dispense both a soda and a nickel. (The machine contains an infinite supply of sodas and nickels.)

The laboratory tells students that they will employ two flip-flops (Q_1, Q_0) in their machines: (0, 0) indicates that the vending machine user has five cents toward a FizzlePop; (0, 1) indicates ten cents; (1, 0) indicates the machine dispenses a FizzlePop; and (1, 1) indicates that the machine dispenses both a FizzlePop and a nickel.

In the assignment's first part, students prepare for circuit design. They create two truth tables specifying the circuit's internal behavior. The first truth table has the flip-flop values on its left side and the controller outputs on the right side; the second truth table has the controller inputs and the current flip-flop values on the left side and the updated flip-flop values on the right side. After building the truth tables on paper, the students then construct four minimized Boolean expressions, one for each of the controller outputs and one for each of the two updated flip-flop values.

In the second part, the student builds the complete controller circuit using Logisim. To aid the student in constructing a complete circuit incorporating the expressions from the first phase, the laboratory assignment shows Figure 5. Building the circuit in Logisim is generally a two-step process: First, the student lays down all the components of the circuit to get an outline of how things will fit (yielding Figure 6(a)). Then the student connects the various components with wires (Figure 6(b)).

The third part of the laboratory assignment has the student test the circuit to verify that it works as specified. Assuming the circuit currently appears as in Figure 6(b), the student notes that the current outputs are (1, 1), indicating that the machine has just dispensed a FizzlePop and a nickel. Suppose the student decides to try simulating the insertion of a nickel. The student selects

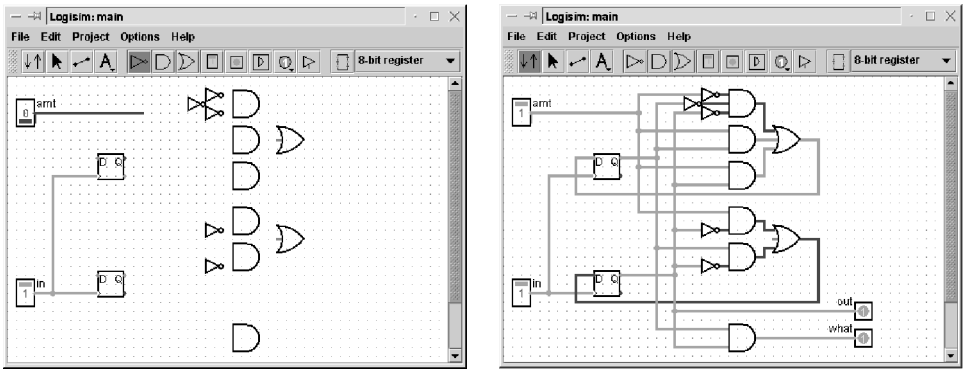

Fig. 6. Building the FizzlePop controller circuit.

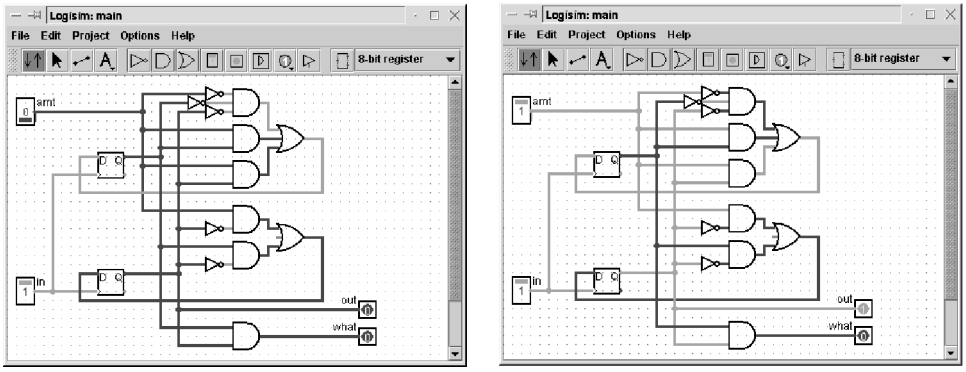

Fig. 7. Testing behavior of FizzlePop controller circuit.

the simulation tool and clicks on the *in* switch to change that input to 0; now we are simulating no coins being inserted. The student would then switch the value of *amt* to 0; still nothing is being deposited. And finally the student clicks the *in* switch, simulating the nickel's insertion. The displayed circuit would now be that of Figure 7(a). The student observes that the machine's output (0, 0) indicates that the machine dispenses nothing, which matches the proper behavior, as now the hypothetical user has deposited only five cents toward a FizzlePop.

Now, the student might reason that inserting a dime should alter the controller's output to indicate dispensing a FizzlePop. To simulate this, the student would click *in*, then *amt*, then *in*, to simulate the insertion of a dime. The output (1, 0), observed from the screen shot in Figure 7(b), indicates that the controller intends to dispense a FizzlePop. This too matches the requirements.

By continuing a variety of such scenarios, the student verifies that the sequential circuit is behaving as required. Finally, the student prints the circuit to be included in a lab report outlining the design process and the final circuit design.

5. LIMITATIONS AND FUTURE PLANS

One of Logisim's limitations derives from its use of the Java Virtual Machine. While this enables Logisim to work on a variety of platforms, it complicates both installing and starting the program. Locally, we have set up simple scripts on our Windows and Unix computers so that students can easily start the programs—so this is not an issue in our laboratories. But it does complicate Logisim's use for students on their home computers and for those at other institutions.

Also, the program's construction pays little attention to details of efficiency. Some sluggishness can appear. For example, in running the CPU of Figure 4 on a recently purchased Sun workstation, Logisim delayed a full second after each toggle of the clock switch. But for circuits of a moderate size, the delay is hardly perceptible. We have not noticed any problems with efficiency in our laboratory assignments.

But perhaps Logisim's biggest limitation is in its features set. While Logisim is far more powerful than needed for lower-division courses in which logic circuits are a small part, an upper-division architecture course would stretch it to its limits. Logisim could still prove useful in such a course—and, compared to most other free graphical tools, it is much more suitable. But some topics could not be addressed using Logisim. For example, Logisim provides minimal or no support for buses, circuit timing, and circuit layout issues.

Possible additions to Logisim in future versions include the following:

—Logisim should have a clock device.

—All of Logisim's components appear in a fixed orientation. It should enable the rotation of components.

—Logisim might permit the user to define the appearance of a subcircuit when it appears in another circuit. Currently, it appears simply as a rectangle resembling an IC chip.

—Each pin between a subcircuit and its surrounding circuit is currently one-way: values will flow either into the subcircuit or out of the subcircuit. Pins on more sophisticated IC chips often are two-way, acting as an input or an output based on other pins' inputs. Logisim does not currently support this.

—Currently, a circuit can use only subcircuits defined within the same file. It would be nice if Logisim allowed the use of components defined in other files, permitting the creation of libraries of Logisim components.

—Logisim might define a Java API for defining new circuit components, allowing others to define new custom components in Java, which may then appear in circuits within Logisim. For example, the CPU of Figure 4 uses a custom component to represent a 32-byte memory, a custom-written component not available through the interface. In addition, users should be able to bundle such files into a JAR file.

—Logisim does not permit bundling of wires. This makes the prospect of developing a circuit for dealing with wide units of data impractical. Thus, this author could certainly not recommend using Logisim to illustrate circuits

working with even 16-bit data. The 8-bit CPU illustrated in Figure 4 is nearly as complex a circuit as Logisim can conveniently handle.

These are merely possibilities for future development. Logisim currently runs without problems, and these features are well beyond the needs of the courses we regularly teach at CSB|SJU. If, however, feedback indicates that Logisim is proving useful, the author would welcome the opportunity to extend its capabilities further.

Logisim is currently in heavy use at CSB|SJU; about 125 of our students learn to use the program each semester. The simple sophistication of such a tool enables students to practice more concepts from class than previously possible, with immediate feedback. Our experience indicates that graphical design and simulation of digital logic circuits, and particularly Logisim, aid student learning in a variety of lower-division courses. Students have reacted positively, and Logisim has become a valuable tool throughout our curriculum.

ACKNOWLEDGMENTS

The author would like to thank J. Andrew Holey, who patiently endured many of the bugs of early versions of Logisim deployed in his classroom and suggested many of its features. The author also acknowledges the helpful referees, whose recommendations concerning possible additions to this article improved it considerably.

REFERENCES

ARASE, K. 2000. *Simcir*. http://www.tt.rim.or.jp/~kazz/simcir.
BOOTHE, B. 1999. *Probe*. http://www.scit.wlv.ac.uk/~cm1970/probe/webpage/.
Capilano Computing Systems Ltd. 1999. *LogicWorks*. http://www.logicworks4.com/.
Capilano Computing Systems Ltd. 2001. *DesignWorks*. http://www.designworks4.com/.
CRAIG, D. 1997. *DigiTCL*. http://www.cs.mun.ca/~donald/digitcl/.
ECK, D. 2000. *xLogicCircuits*. http://math.hws.edu/TMCM/java/xLogicCircuits/index.html.
Gordon College. 2001. *Logic Simulator*. http://www.cs.gordon.edu/courses/cs111/module7/logicsim/example1.html.
HERZ, A. 1998. *Digital Simulator*. http://www.ttl-simulator.de/.
KARWEIT, A. 2000. *Circuit Builder*. Johns Hopkins University, http://www.jhu.edu/~virtlab/logic/logic.htm.
KNAIAN, A. 1994. *Digital Simulator*. http://web.mit.edu/ara/www/ds.html.
MASSON, A. 1996. *LogicSim*. http://wuarchive.wustl.edu/edu/math/software/mac/logic/LogicSim/.
MECANIQUE. 2001. *Digital Works*. http://www.mecanique.co.uk/digital-works/.
MILLER, E. AND SQUIRE, J. 2000. Esim: A structural design language and simulator for architecture education. In *Proceedings of the Workshop on Computer Architecture Education*. 42–48.
Research Systems Pty. Ltd. 2001. *EasySim*. http://www.research-systems.com/.
Softronics, Inc. 1997. *Multimedia Logic*. http://www.softronix.com/logic.html.
TETZL, A. 2001. *LogicSim*. http://www.tetzl.de/.
VAN RIENEN, I. 1996. *DigSim*. http://www.iwans.net.
WOLFFE, G., YURCIK, W., OSBORNE, H., AND HOLLIDAY, M. 2002. Teaching computer organization with limited resources using simulators. In *SIGCSE Technical Symposium on Computer Science Education*.
ZIDAR, F. 2001. *Digital Circuit Simulator*. http://www.rocketdownload.com/Details/Home/925.htm.

Received October 2001; accepted February 2002