

A Pedagogically Targeted Logic Design and Simulation Tool

David A. Poplawski
Department of Computer Science
Michigan Technological University
Houghton, Michigan 49931
pop@mtu.edu

Abstract

JLS is a GUI-based digital logic simulation tool specifically designed for use in a wide range of digital logic and computer organization courses. It is comparable in features and functionality to commercial products, but includes many student and instructor-friendly aspects not found in those products such as state-machine and truth table editors, extensive error checking, and multiple simulation-result views. Students quickly become proficient in its use, enabling them to concentrate on circuit design and debugging issues. The circuit drawing interface is convenient enough to allow instructors to use it for classroom presentations, and circuits can be modified and tested so quickly that it promotes exploring alternatives not prepared for in advance. Its non-interactive (batch) execution capability, with parameter settings, configuration files and textual output simplifies the grading of large numbers of student projects.

Categories and Subject Descriptors

K.3.2 [Computers and Education]: Computer and Information Science Education

Keywords

Logic Simulation

1. Introduction

Choosing a logic simulation tool for introductory courses in logic design and computer organization is problematic. On the one hand there are industrial strength tools that are

expensive, have steep learning curves, and are generally used by experienced designers. Many of these tools use textual, as opposed to graphical, input.

On the other hand there are many free tools. Most are (very) limited in functionality. Few will work on multiple platforms.

JLS is a portable powerful digital logic simulation tool created primarily for educational use that addresses the problems listed above. It is written in Java and has been tested on many platforms. It is free. It consists of a simple to use yet powerful graphical editor that allows users to create and modify logic circuits, and a simulator that will show (in multiple ways) the operation of the circuit over a period of time. Circuits as simple as a few logic gates or as complex as complete CPU microarchitectures with embedded subcircuits have been created and simulated in JLS.

Logic circuits can contain the standard gate types: AND, OR, NOT, NAND, NOR, XOR, tri-state buffer and logically neutral time delay element; composite elements: decoder, multiplexor, and adder; memory elements: registers, SRAM, and ROM; a clock and various mechanisms for connecting gates and elements via wires and wiring elements. State machines can be created by using JLS's state machine editor. Combinational circuitry specified by a truth table can be generated by using JLS's truth table editor. Complex, synchronized multi-signal inputs can be specified. Circuits can include copies of other circuits (subcircuits), nested to an arbitrary depth. Circuits can be printed or exported as image (JPEG) files.

JLS has many features that are useful for classroom instruction, student use, and for the grading of students' circuit assignments. These include simple and intuitive drawing and editing, comprehensive error checking/reporting and extensive on-line help and undo/redo.

This paper discusses the pedagogical requirements of a digital logical simulation tool. It then describes JLS and, in particular, its features that address those requirements. Finally it compares JLS with other available tools.

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.

WCAE '07, June 9, 2007 San Diego, CA

Copyright 2007 ACM 978-1-59593-797-1/07/0006...\$5.00

2. Pedagogical Issues

A logic simulation tool designed for pedagogical use shares many attributes with large, expensive professional tools, but many attributes are peculiar to this use. Among the most important are cost (the cheaper the better, but free is best), extreme ease of use, convenient ways to view a circuit's dynamic operation, and mechanisms to simplify the grading of multiple student projects. These and other issues are addressed below.

2.1. Graphical User Interface

Instructors draw logic diagrams when they lecture, textbooks show logic diagrams when illustrating circuits, so a simulation tool must also have a graphical user interface so that students can experiment with circuits in the way they have seen them presented. Text-based circuit specification is too cumbersome and non-intuitive for students new to the area of study. Circuit errors such as missing and wrong connections are hard to find, and debugging can be extremely tedious. The old adage that a picture is worth a thousand words certainly applies here.

It is extremely helpful when illustrating circuit operation or debugging new circuits to be able to watch a circuit as it operates, either by including special circuit elements that display an input value or annotating wires. Signal traces can be displayed or, better yet, updated in real time as the simulation of a given circuit proceeds at a speed that is under user control.

2.2. Wide Applicability

Many logic design and/or computer organization classes deal with circuits as simple as a few gates to those as complex as complete CPU implementations. Hence a wide range in the complexity of components is needed:

- Simple logic gates (AND, NAND, OR, NOR, XOR, etc) to construct simple circuits illustrating basic logic concepts.
- Composite elements (adders, multiplexors, decoders, registers, memories) so that complex circuits can be constructed with ease.
- Wire bundling so that a single line in the circuit can represent multiple wire signals, with circuit elements to bundle and unbundle the wires in arbitrary ways.
- Truth-table specification to avoid having to resort to cumbersome circuit specifications for otherwise simple logic functions.

- State machines to illustrate complex signal generation, bus protocol interfaces, and CPU control signal generation, without having to map an intuitive state diagram into state register, output signal and next state logic.
- Arbitrarily nested subcircuits inclusion to create circuit elements that can abstract logic functions (e.g., half adders, full adders, ALUs, register files) and to simplify circuit appearance.
- Gate and element propagation delays so that race conditions, circuit timing, clocking, and CPU cycle time can be analyzed.

2.3. Platform Independence

Today's college and university computing environments often include multiple hardware (Intel PC, Mac, Sun, etc.) and software (Windows, Linux, OS X, Solaris, etc.) configurations. Even when a single platform is available, students often own personal computers that are different. Hence a tool that is portable to all platforms is desirable. This flexibility allows the students to work in any environment that suits them.

2.4. Instructor Requirements

A good tool will have two important uses for the instructor. One is for class presentation of circuit concepts. The other is to grade numerous student projects easily and efficiently with minimal hand work.

Using a logic simulation tool during classroom instruction requires that it be extremely easy to use:

- Creating and placing logic elements should be convenient (drag and drop, cut and paste).
- Connecting elements (i.e., drawing wires) should require minimal work (e.g., via simple mouse clicks).
- Elements (and their connections) should be easily rearranged so that new elements can be introduced to encourage experimentation and the testing of alternatives.
- Subcircuits created prior to class should be importable and integrated with minimal effort.
- Input signal values and/or sequences and memory element values, created in advance of class, should be supported, and during-class modifications easily accommodated.
- Simulation of a given circuit should progress either quickly, to see the end result, or incrementally (under instructor control) to observe time-varying circuit behavior.

- Signal values and memory element values should be easy to see and/or monitor in real time.

Grading student circuits for correctness must be as convenient as possible. In its simplest terms, this means that there must be a way to load and simulate a circuit in "batch" mode (i.e., from the command line), without the GUI appearing and without further human intervention. It requires that input signal values and/or memory element values be configurable externally to the circuit being tested, preferably in files that are read by the simulator prior to or during simulation. It also requires the ability to print the circuit being tested on paper, display final signal and memory element values, and set simulation time limits to avoid runaway simulations.

2.5. Student Requirements

It is assumed that students in an introductory logic design and/or computer organization course will have little, if any, experience with logic simulation tools. This implies several requirements:

- Everything must be as intuitive as possible and easy to learn, requiring a very gradual learning curve.
- Tutorials walking the students through the steps of creating increasingly complex circuits using more and more features are needed.
- On-line help must be extensive, complete and easily navigable.
- The GUI should detect and prohibit obvious errors, such as nonsensical circuits (e.g., two wires to the same input, directly connecting two non-tristated outputs) and overlapping elements.
- Undo/redo should be available so non-working "solutions" can easily be eliminated.
- Cut/paste should be supported in order to make circuit replications convenient.
- Debugging should be simple, especially w.r.t. slowly stepping the simulation of a circuit and the interrogation and display of signal and memory element values.
- The tool should be portable so that the student can use it at their convenience on their platform of choice.

3. JLS

JLS addresses all the pedagogical issues discussed above. It is easy to learn and use but powerful for both instructor and student in a wide range of digital logic and

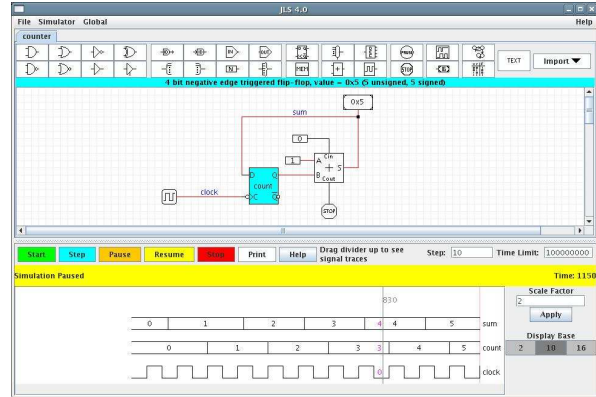


Figure 1. JLS window with simple circuit and paused simulation signal trace.

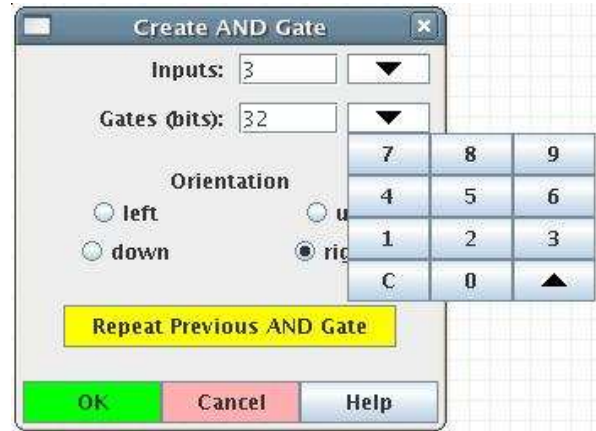


Figure 2. Gate creation dialog (32 3-input AND gates).

computer organization courses. Rather than enumerate all its features and describe each, its more interesting and unique capabilities will be described here. Figure 1 shows the JLS GUI with a simple circuit and simulated signal trace.

3.1. Circuit Creation

Circuits are constructed by selecting, configuring and placing logic elements in the drawing area. The user simply clicks on an element on the tool bar, which brings up a dialog box with which the user can select characteristics such as the number of inputs to logic gates, the number of replications of the element orientation, names for registers, sizes and widths of memory, etc (see Figure 2).

When satisfied the resulting element is dragged to its desired location and placed. As it is being dragged, overlaps

with existing elements are indicated, and placing an element on top of another is not allowed. Also, when inputs or outputs of the new element overlap unattached inputs or outputs (unattached inputs/outputs are small circles), or unattached wire ends, the circles turn green if a legal connection can be made or a message is displayed indicating why they cannot be made (for example, if the user attempts to directly connect the outputs of two non-tristate gates, or if the bit-widths differ).

Multi-segment wires are drawn by simply clicking them in place. The wires automatically assume the bit width of the input or output or other wire they become connected to. The end of a wire can be connected to another wire at any point along that wire. Illegal connections are detected (for example, creating a wire loop) and displayed as the wire is being drawn.

The characteristics of an element already placed are displayed by simply moving the cursor over the element. Wire widths and type (i.e., tri-state or not) are also displayed when the cursor is over any part of the wire.

Single elements and/or wires can be dragged to new positions; wires attached to moving elements move with the elements. Collections of elements and wires can be selected by simply tracing a rectangle over them, then moved (dragged), cut, copied or deleted en-mass. As collections are being moved, overlap and possible connections are detected and reported. Cut or copied collections can be pasted at other places in the circuit being edited, or into other circuits that are currently open (JLS can have several circuits open at the same time, using tabs to select the currently visible, edited one).

3.2. Subcircuit Inclusion

Subcircuits are a powerful abstraction tool, and JLS supports the nesting of subcircuits to an arbitrary depth. Subcircuits are imported (i.e., a copy is made), not linked. Imported subcircuits can be opened for editing without leaving JLS or having to re-import modified versions. Imports can come from existing circuit files, or from circuits just created but not yet saved.

3.3. Truth Table Editor

Rather than specify complicated logic using many individual gates and complex wiring, a user can use the truth table editor to specify the behavior of the equivalent circuit. Truth tables with an arbitrary number of single-bit inputs and outputs can be specified. Both input and output don't-cares are supported.

Figure 3 shows the truth table for a full adder. The user enters the input and output signal names, then clicks on any value in the truth table to toggle among 0, 1 and don't-care.

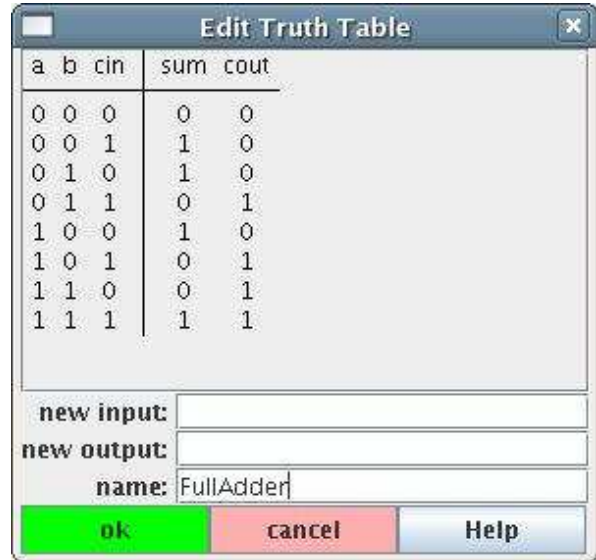


Figure 3. Truth Table Example.

Inconsistent outputs when an input don't care is "proposed" are prevented. The effect of the equivalent logic is simulated rather than the actual gates and wiring being generated.

3.4. State Machine Editor

Complex state machines (Moore machines) can be created and modified using the state machine editor. The user draws the state diagram (states and transitions), conditions the transitions, and specifies the output signals at each state with a simple-to-use GUI. The resulting machine can trigger on rising or falling edges. JLS does not construct the equivalent machine from a register and logic, but simply simulates the effect of such logic. There is a (user definable) delay between the trigger edge and the resulting state transition and the appearance of new output signal values.

Figure 4 shows a sample state machine (the control section of the multi-cycle machine in [4]) in the editor. Note that states are labeled, transitions show their conditions, and the outputs of the branch state are shown.

The user can add or remove states and/or transitions, change conditions and outputs, move things around, and even align groups of states horizontally or vertically. The editor ensures that the state machine is deterministic and prevents inconsistent usages such as outputting two different values for the same output signal in the same state or differing bit widths for the same input or output signal.

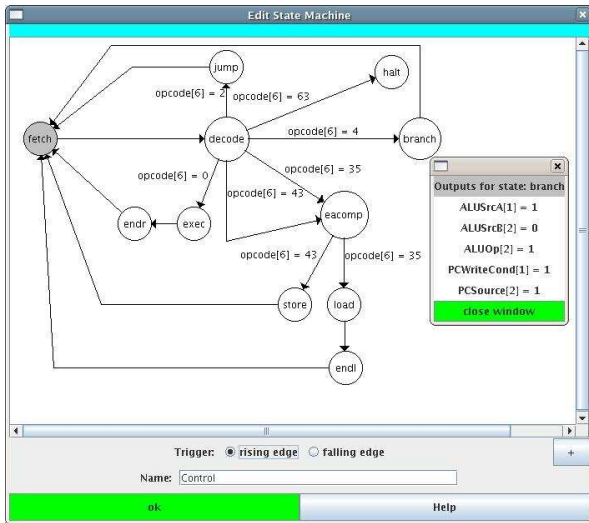


Figure 4. State Machine Example.

3.5. Simulation

JLS implements an event-driven simulator that supports (user defnible) propagation delays for all elements (but not wires, which have 0 delay). When started normally, the simulator will run as fast as possible until one of four things happens: there will be no more signal changes, the time limit is exceeded, the user clicks the pause button, or special circuit elements (stop, pause) receive non-zero input values. Paused circuits can be resumed, stopped circuits will restart from initial settings. The simulator can also be stepped by an arbitrary amount of simulated time that can be varied between steps.

Whenever the simulator stops or is paused the user can interrogate the values of any wire or register by pointing at it. All values are displayed in three ways - hexadecimal, unsigned decimal and signed (twos complement) decimal. In addition, the color of a wire indicates if it has a zero or non-zero value on it (black for zero, red for non-zero). Hence for simple circuits with non-bundled wires the values of signals as simulation progresses can be observed with no extra effort. A click of the mouse on a memory element will display the contents of that memory element in a scrollable window.

Signal traces are also shown when desired by the user. In Figure 1 the simulation of a simple 4-bit counter (clock, register, adder) has been paused at simulated time 1150. The traces of the register was enabled by clicking on the register and selecting a "watch" menu item. The traces of the clock and adder were enabled by putting "probes" on the wires coming from their outputs. The probe names are shown on the wires in the circuit and are placed by clicking on the wires, selecting a menu item, and typing a name.

3.6. Signal Generator

Complex signal sequences can be specified with JLS's signal generator element. For example, if a circuit has input pins named A and B, then the user could specify

```
A 0 for 10 1 for 20 0 until 95 1 end
B 0 until 15 1 for 20 0 for 5 1 end
```

in the signal generator dialog. This generates to the pin named A the value 0 for 10 simulated time units, then changing to 1 for 20 time units, then back to 0 until simulated time 200, and then finally back to 1 until the simulation stops. It also generates the B value sequence simultaneously.

3.7. Batch Operation

JLS can be run in "batch" mode, in which a circuit is loaded and simulated to completion (or time limit) without the GUI appearing or any other user intervention. When simulation ends the final contents of all watched elements (registers, memories, output pins) are printed to the standard output. In the case of memory elements, only those "words" that changed during simulation are printed. Here is an example output from a simple CPU circuit:

```
Simulation Stopped at 42977
Register PC: 0x12
(18 unsigned, 18 signed)
Changed locations in memory MEMORY
0x4: 0x0 (0 unsigned, 0 signed) ->
0x37 (55 unsigned, 55 signed)
```

When a circuit is run in batch mode several parameter files may be specified:

- A signal generator file that will send the specified signals to the input pins of the loaded circuit during simulation.
- A parameter file in which element propagation delays can be changed, element watches can be set or reset, registers can be given initial values, and memories can be told to read their initial contents from specific files.

There are also command line flags for setting the simulation time limit and generating a printable image file of the circuit.

3.8. Locking Circuit Elements

It is often useful to be able to give students a partially completed circuit and have them finish it. However there may be parts of the circuit that the instructor doesn't want

students to be able to modify. JLS supports a locking mechanism on individual elements enabling the instructor to specify those parts of the circuit they don't want modified. JLS then prohibits the modification of any properties or the deletion of any locked element.

4. JLS versus Other Tools

Burch [1] and Wolffe [6] contain excellent, although slightly dated, summaries of the existence and features of many free, GUI based circuit simulators. Most of those listed are platform specific (mainly Windows and Macintosh). A few are Java applets and hence restricted from loading and saving files, and generally have very restricted functionality. Two are Java application programs and hence available on all platforms supporting the JVM: LogicSim [5] and Logisim [1].

LogicSim has an intuitive circuit drawing mechanism and the ability to display current signal values. It also has a subcircuit (module) inclusion mechanism. However it only supports basic logic gates, flip-flops, and clock. Wires cannot be bundled. Simple inputs values come from switch and number-input elements. Outputs are displayed by LED and LCD-like elements. There is no concept of time and hence no propagation delays. There is no batch execution mechanism and hence little to assist with grading of students' circuits.

Logisim also has an easy-to-use drawing mechanism and the ability to display current signal values. It supports basic logic gates, tri-state gates, flip-flops, constant value sources, and has a subcircuit inclusion. There are no other complex elements, no wire bundling, no propagation delays and no batch execution capability.

The tools most comparable to JLS in functionality are LogicWorks 5 [2] and TkGate [3]. LogicWorks is a commercial product (i.e., not free) that runs on Windows and Mac platforms only. It has little support for grading purposes.

TkGate is free and designed to run in a Linux environment. It has a fairly intuitive drawing mechanism, but does not detect error conditions like wire width mismatches or directly connecting the outputs of two ordinary gates. Logic elements can overlap, which is confusing. It has a wide range of logic elements from simple gates to multiplexors, adders and memory, but no state machines or truth-table specifications. Documentation is available only from the tkgate web site, not built into the tool. There is no text-based output making batch grading difficult.

5. Availability

JLS is a free Java (version 5) application program packaged in two jar files and can be obtained from the author

by sending an email request. It has been thoroughly tested on Windows and Linux based platforms and has been used by students in computer organization courses at Michigan Technological University and Grand Valley State University. It has been tested to a lesser extent on Sun-Solaris and Apple Macintosh-OS X. It will not work with one-button mice.

An applet version of JLS exists at www.cs.mtu.edu/~pop/jlsp/bin/JLS.html. A simple tutorial accompanies the web page that leads new users through the basics of constructing and simulating three increasingly complex circuits: a simple combinational circuit for the boolean expression $A|!B$, a 4-bit counter constructed from a register, adder and clock, and a full adder that imports half adder subcircuits. All the drawing and simulation features work as in the application program version. However, since it is an applet, existing circuits can not be loaded and user-drawn circuits cannot be saved.

5.1. Circuit Library

A library of circuits is being constructed and is available from the author's web site. Included are circuits such as simple half and full adders, various latches and flip-flops, static RAM, and complete CPU circuits such as the one-cycle and multi-cycle CPU circuits from Patterson and Hennessy [4].

6. Acknowledgments

JLS began as a student project many years ago after several failed attempts. Eric Simonton designed the first practical GUI version that generated a textual representation that was then simulated with a separate tool. His simple circuit drawing mechanism, slightly modified, is the basis for the current version. Eric Dalquist, Benny Evans and Jonathan Even, in a senior software engineering project, added a subcircuit inclusion mechanism and an integrated time-based simulator, thereby avoiding the extra steps required to use a separate tool and the ability to query signal and memory element values during, and not just at the end of, circuit simulation. Jachary Kurmas of Grand Valley State University, his students and my students are also credited for finding several bugs and suggesting many improvements.

References

- [1] C. Burch. Logisim: a graphical system for logic circuit design and simulation. *J. Educ. Resour. Comput.*, 2(1):5–16, 2002.
- [2] Capilano Computing. *LogicWorks 5 Interactive Software*. Prentice Hall, 2003.

- [3] J. Hansen, 2006. <http://www.tkgate.org>.
- [4] D. Patterson and J. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, third edition, 2005.
- [5] A. Tetzl, 2006. http://www.tetzl.de/java_logic_simulator.html.
- [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, New York, NY, USA, 2002. ACM Press.