# AC 2007-2128: CEDARLOGIC ? A NEW GRAPHICAL DIGITAL LOGIC CAD TOOL

**Clint Kohl, Cedarville University**

Dr. Kohl serves as Associate Professor of Computer Engineering at Cedarville University. He earned his B.S.E.E. from South Dakota State University, his M.S.E.E. from University of North Dakota, and his Ph.D. in Computer Engineering from Iowa State University. His areas of interest include digital electronics, computer architecture, programmable logic devices, and microprocessor systems.

**Keith Shomper, Cedarville University**

Dr. Shomper serves as an Associate Professor of Computer Science and has been at Cedarville Universtiy since August 2003. He received his B.A. in Mathematics from the University of Northern Colorado (1983) and his M.S. in Computer Science from the Air Force Institute of Technology (1984). Dr. Shomper received his Ph.D. in Computer Science from the Ohio State University (1993), specializing in computer graphics with minors in software engineering and distributed computing. His dissertation was in the area of visual degugging of computer programs. Dr. Shomper's research interests include computer graphics, distrubuted simulation, and virtual reality.



© American Society for Engineering Education, 2007

# CedarLogic - a new Graphical Digital Logic CAD tool to aid in the teaching of Digital Logic Design.


## Abstract

This paper describes, "CedarLogic" a graphical digital logic simulator that three senior undergraduate students created in fulfillment of their Senior Design Capstone course in the 2005-2006 academic year. This educationally valuable software is being effectively used in an introductory Digital Logic Design (DLD) course. This paper presents the background and need for this type of software tool, a brief analysis of currently available tools and then explains its functionality and usefulness.

This easy to use logic simulator is valuable in both Digital Logic Design lectures and labs. The environment is graphical in nature and allows the user to very quickly build a logic circuit by clicking and dragging components from the reasonably complete library of gates and functions including: AND, OR, NOT, NAND, NOR, XOR, Multiplexers, Decoders, Adders, Comparators, Flip-Flops, Counters, Registers, RAM, ROM, and numerous Input and Output options. One of the most helpful features of this software is the simultaneous build and simulate environment with wires colored according to their logic value (Red for logic High and Black for logic Low). This allows the user to quickly understand how the logic is working and, if it is not working properly, to correct mistakes. The freshman students using this program for the first time have found it to be stable, helpful and in some cases even "fun" to play with and design.

The paper concludes with some lessons learned through the Senior Design Capstone experience from which this multi-threaded software was designed, written, debugged, revised and released for experimentation in DLD. CedarLogic's 10,000+ lines of code is written in C++ and utilizes the wxWidgets GUI library and OpenGL to render the graphics. CedarLogic can be freely downloaded at http://sourceforge.net/projects/cedarlogic .


## Background and Need

Digital Logic Design is a foundational course for many engineering and computer science students. The first author has been teaching a freshman level Digital Logic Design course for over twelve years. The course includes laboratory projects in which students physically wire up TTL gates on a breadboard, use the CedarLogic software tool to build more complex circuits and are briefly exposed to Altera's Quartus II commercial logic software.

We believe student learning can be accelerated and enhanced by the effective use of logic simulation software. A student can connect a TTL logic circuit in lab and observe its functionality by flipping switches and watching LED's light and still not understand how each

logic gate is functioning. Logic simulation software enhances learning by allowing the student to "see" the logic state (high or low) of each wire and come to a better understanding of how every logic gate in the network is functioning simultaneously. Often students who use these software tools will have a "A Ha" experience and say; "Oh, now I see how that works." Additionally, debugging faulty circuits is often quicker with this type of tool, since all nodes are observable rather than just the inputs and outputs. After using logic simulation software for many years, we have determined the following list of desirable characteristics.

1. Easy to use Windows program
2. Free or low cost
3. Simultaneous editing and simulation
4. Capable of changing the wires color according to its logic value
5. Smooth transition to advanced commercial software used in the later courses

Prior to the development of CedarLogic, we used the software program "Diglog".[1] Diglog is one of the components of the Chipmunk distribution of computer-aided software tools developed at UC Berkeley in the late 1980's and early 1990's. This Unix-based software is very powerful. It has simultaneous editing and simulation, an extensive library of gates and can be placed in "Glow" mode where the wires glow red or black depending on their logic value. It was ported to the Windows platform in 1998 by a group in Germany where a free download (logwin32.exe)[2] is still available today.

Diglog was written by Dave Gillespie and is a unit-time-delay digital circuit simulation package. Circuit schematic editing and parameter adjustments can occur while the simulator is in operation, supporting the metaphor of a virtual lab workbench. A screen capture of a full adder is shown in Figure 1 below.

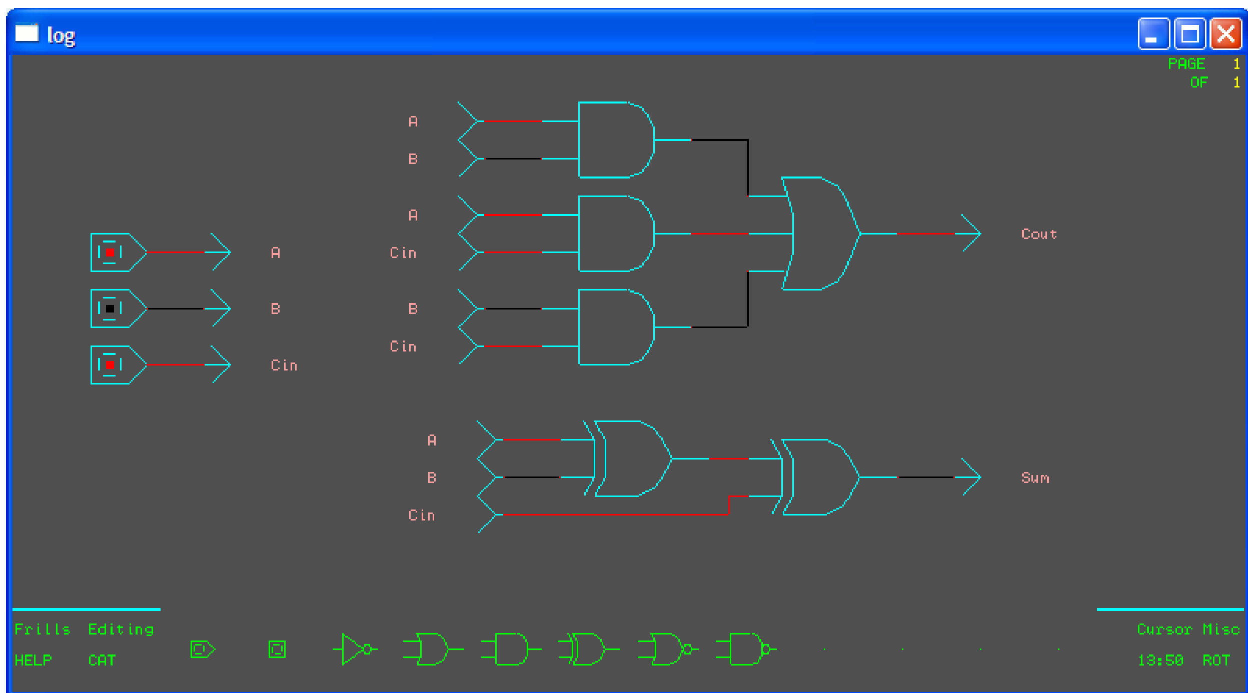


Figure 1 Color Screen capture of a full adder in Diglog.

The main disadvantages of Diglog are itemized in Table 1 below:

Table 1 List of Key Diglog Disadvantages

- No Undo feature
- Non standard Windows interface
- Busy wait implementation consumes 99% of CPU resources
- White gates on black background make screen captures difficult and require inverting colors
- Wires do not rubber band
- Cut and paste is non-standard and it is easy to copy two gates on top each other
- Printouts are cumbersome to obtain – non encapsulated Postscript only
- Missing some common gates in the library (8to1 Mux, 16bit Memory device, etc)
- Windows clipboard is not used
- Opening and saving files difficult: command line based rather than standard windows open/save dialog box. This requires remembering the exact file path and filename.
- Difficult to implement a large multi-page circuit. Multiple pages can be open at once but each page must be loaded/saved separately: students often forget and loose their work.

Despite these weaknesses Diglog remains an excellent alternative for use in teaching Digital Logic Design. We have been looking for other alternatives to Diglog for many years. A multitude of options are available. Some alternatives are listed in Table 2.

Table 2 Partial List of Digital Logic Design and Simulation Software Alternatives

- Quartus II from Altera Corp. http://www.altera.com/ Free Web Edition
- ISE *WebPACK* from Xilinx Corp. http://www.xilinx.com Free Web Edition
- Diglog http://wwwags.informatik.uni-kl.de/utils/DIGLOG/main.html Free
- MultiMedia Logic by Softronix http://www.softronix.com/logic.html Free
- Digsim by Paul Fishwick: http://www.cise.ufl.edu/~fishwick/dig/DigSim.html Free Web based Java Applet
- Digital Simulator by Ara Knaian 1994 http://web.mit.edu/ara/ds.html Shareware $10-$20
- EasySim: Research Systems in Australia http://www.research-systems.com/easysim/easysim.htm $14
- Digital Works Logic Simulator http://microcontrollershop.com/product_info.php?products_id=663 Cost $77
- B2 Logic Version 3 from Beige Bag Software http://www.beigebag.com/logic3.htm cost $179
- DesignWorks Professional http://www.capilano.com/index.html cost $395
- Electronics Workbench http://www.electronicsworkbench.com/products/proprod_pl.html Educational packages start at $479 USD

Each of these alternatives lack at least one of the items from the list of desirable characteristics shared earlier.

Cedarville University currently requires students to use the Quartus II software from Altera Corporation in advanced coursework, and thus we are very familiar with it. This commercial

tool set is offered free to the public, however it is very large (≈500Mbytes) and not easy to learn. It does not offer simultaneous editing and simulating and the wires do not change color based on their logic value. This outstanding program is just too complicated and not well suited for the introductory logic student. A similar argument could be made for the ISE WebPack from Xilinx Corp.

After years of searching for the right tool we decided to let some of our seniors attempt a simulator of their own design. In the 2005-2006 Academic year two teams of students, undertook the project, with one of those teams consisting of two computer engineers and one computer scientist. This team worked from scratch to create the program we call "CedarLogic".

**Senior Design Format**

The software engineering course sequence at Cedarville University is the capstone design and development experience for students in our computer science and computer engineering programs. The sequence consists of two courses. The first course, Software Engineering I (also cross-listed for Computer Engineering students as Computer Engineering Senior Design I), covers process models for software project management, customer requirements analysis, preliminary and detailed design modeling, test case development and application prototyping. The second course, Software Engineering II (Senior Design II), emphasizes iterative application development, program reviews, test execution, user documentation, and deployment.

While the course sequence begins in the fall semester of the students' senior year, planning for the course typically begins at the end of the previous spring semester. The process begins with students and faculty suggesting projects for the following year as the current year's projects wind down. Project ideas are usually a mixture of continued research from former student work and new applications for both the engineering department and industry partners. Any student or faculty member in the program may submit a project idea. We also receive a few external submissions from colleagues and business partners who are familiar with the courses. During the summer, designated faculty coordinate with the proposed project sponsors to determine project scope, technical complexity, resource requirements, sponsor support and availability, and development schedule. Such pre-coordination is necessary to help ensure we offer students projects which are suitably challenging in both size and content.

With regard to project size and scope, we endeavor to provide projects employing all phases of the software development cycle, having approximately 800 to 1200 man-hours of work effort, and also requiring at least a modest attempt at independent research beyond our programs' course curriculums. Once all candidate projects are approved by the faculty, we develop a brief presentation for each one to give to our senior students on the first class day. Students then rank order the projects in which they have the most interest. At the same time, they also identify who among their peers they would like as team members. This information is used by the faculty to make project assignments. In general, we attempt to provide all students their first or second project choice. We also usually honor their teammate choices.

**Requirements and Design Process**

After the teams are selected, the students spend the remainder of the first semester studying, and practicing, the topics listed for Software Engineering I above. We accomplish this activity using a series of timed deliverables. These deliverables are primarily, but not exclusively, documents in the first semester and software in the second semester. For each document, we discuss its purpose in the context of the software development process in which it occurs, give an overall outline for the document, and discuss techniques for developing the content.

Speaking of the development process, we should note here that we do not mandate that our students follow a specific software development process. However, after studying several common processes, including some popular agile processes (e.g., extreme programming, feature-driven development, etc.), we suggest to the student teams that they adopt an iterative development method. Our motive for using iterative development is to encourage the teams to first develop a functional product exhibiting core features with subsequent iterations adding additional capability. This development style tends to keep teams of three or four students fully engaged on their projects for the entire two-semester course sequence, with sufficient flexibility so that weak teams may successfully produce a core-level product and strong teams are motivated to accomplish additional build iterations to create feature-rich applications.

Most commercial or industry applications are only as good as the analysis and design effort spent to create them. However, most programming or term projects in undergraduate programs can be accomplished individually or with a small team in a few weeks. For this reason, we size our problems as discussed above and encourage students to take seriously the documentation of project milestones and objectives, stakeholder requirements, design ideas and alternatives, and procedures for test execution. Therefore, we require the following deliverables from the student teams in the first semester:

1. Weekly Report: Each week, each student team must submit an activity report indicating what actions each team member accomplished the previous week and what actions they plan on accomplishing the following week. Teams are also asked to report the hours they spent on the project to encourage accountability on the team and to the instructor. These reports, like most all other documents, have a specified content, but no specified format. Students are encouraged to keep these reports brief, usually just a paragraph or two from each team member, in order to minimize the management overhead.
2. Program Management Plan: The first major document for the team is the program management plan (PMP). This document, due approximately four weeks after the start of the semester, allows students an early opportunity to describe their assigned project in their own words, to self-organize into various team roles (e.g., team leader, configuration management, quality control, etc.), to consider the hurdles they must overcome in order to complete the project, and to begin to draft the project plan that they will then monitor and track until project completion.
3. Software Requirements Document: As students are planning their development schedule, they also begin conducting interviews with stakeholders to become more familiar with the application. The culmination of these interviews is reflected in the software requirements document (SRD). The SRD includes a full description of the application's

features from a user's perspective. It also presents the application in context with the systems (e.g., hardware, software, process) with which it must integrate. The final version of the SRD is due six to seven weeks into the semester.

4. Software Design Document: At 12 weeks, the software design document (SDD) is due. The SDD begins where the SRD left off by describing the user-visible components (e.g., the graphical user interface or GUI) of the application in a systems context. It then iteratively refines these components, typically into object classes with their attributes, methods, and associations, until a level of detail is reached that allows implementation to begin.
5. Test Plan: The final document in the first semester is the test plan. It is due at the same time as the SDD. The test plan identifies what mechanisms and procedures the team will use to ensure software is evaluated before being submitted to the application build.
6. Initial Presentation: The final deliverable for the first semester is a presentation on the software's semester-end capability. In addition to a demonstration of the software, students must evaluate themselves against their project plans and devise mitigation strategies for incomplete work. When referring to incomplete work we should note that students are not expected to have a complete or even nearly complete application at the end of the first semester. However, according to the technical risks they identified in the SRD, they are expected to demonstrate some concrete progress in implementing the technically challenging parts of their design to validate their design choices and lower their developmental risk in the spring semester.

With the completion of the initial presentation, each team finishes the first semester requirements and should have a solid understanding of the tasks that lay ahead in the project. We discuss a general outline for these tasks in the next section.

**Implementation and Testing**

As the second semester kicks off, there is far less classroom instruction and supervisory activity which must occur. In this semester, student teams operate independently with brief weekly contact from their team advisors. Therefore, in order to ensure consistent progress is made throughout the semester, each advisor identifies a set (e.g., three to five) of milestones which their team must demonstrate to show progress. For example, a professor might require a "core" product delivery four to five weeks into the semester. Teams not delivering their application by the milestone receive a grade penalty.

In addition to the instructor-set milestones, all students have the following additional deliverables in common:

1. Weekly Report: As in the first semester, a report on past and planned weekly activities is required to ensure teams maintain momentum.
2. Final Build: About three weeks prior to the end of the semester, students demonstrate and submit the final build of their application which also includes any user documentation, maintenance notes, and installation notes, software or scripts.
3. Final Presentation: At the same time as the final project submission, teams deliver a formal project presentation accompanied with a written project report. The presentation

is a 50-minutes briefing summarizing the team's work over both semesters on the application. The presentation is publicly announced and open to faculty, students, sponsors, and guests of these groups.

4. Final Report: The project report is written to accompany the presentation and become the permanent record of the student's work. This report includes a project abstract, definition statement, background, objective and constraints, design, results, and addenda (e.g., design diagrams, bill of materials, electronic media, etc.).
5. Poster Presentation: The final deliverable is a poster presentation of the student's work during graduation week. This activity lets the students share the results of their work with friends, family, and those interested in the work, yet unable to attend the formal presentation.

## Application Design and Implementation

As mentioned earlier, CedarLogic was designed to be simple to use and consistent with common Windows application conventions. We also were strongly interested in removing the busy loop simulation handling which caused poor program response and made the visual display flicker badly as the circuit model grew larger. Finally, as consistent with good program design, we wanted to cleanly separate the implementation of the graphical user interface (GUI) from the simulation engine—what we call the "logic core."

To accomplish the first objective, that of giving the application a consistent Windows look-and-feel, we turned to the wxWidgets GUI programming toolkit. While there are other toolkits supporting the same objective, we preferred the open source and free wxWidgets for its clean implementation, full user interface feature set, compatibility with OpenGL, and complete documentation. WxWidgets also provided some useful general-purpose programming libraries, one of which was very useful in implementing the message-passing interface to accomplish objective three. As can be seen in Figure 5, the wxWidgets interface presents its interaction elements: the scroll, menu and tool bars, the window decorations, the icons, canvas, and tabs, etc., in a manner consistent with a Windows application. That is, CedarLogic "looks" like a Windows application. What cannot be seen in the figure, but which is readily apparent when running the application, is that wxWidgets also gives CedarLogic the same "feel" as a Windows application. For example, toolbar icons have pop-up help, drag-and-drop actions are executed with the same mouse buttons and familiar movements, and keyboard shortcuts are available using common mnemonics.

The second objective, removing the busy loop, was the most technically challenging, but necessary improvement over Diglog. Regardless of the complexity of the circuit Diglog demands all available CPU resources. This makes using other programs perform very sluggishly when running at the same time. For example, students running Diglog and trying to write their lab reports in Microsoft Word found that Word behaved very sluggishly.

With CedarLogic we moved to an interaction method that is far more common and compatible to WIMP[4] applications: event programming with callbacks. Having made this decision, it was a natural choice to also separate the code which simulates the circuit, the "logic core" from the code which presents the simulation, the "GUI." The GUI is the part of the application that takes

advantage of wxWidget's presentation features as described above. It also uses OpenGL, a industry-standard graphics library, to accomplish all drawing on the circuit canvas and tool palettes. With OpenGL and wxWidgets, the new interface is clean, colorful, and responsive

The logic core is the heart of the application. It is an event-driven simulator which keeps an internal model of the diagram under construction and receives timed, state change events on the model's wires which it processes at each time step using a priority queue. This means that a circuit in CedarLogic has two models which must be kept consistent: the visual model shown in the GUI and manipulated by the user and the simulation model kept in the logic core and updated at each time step. To keep these models consistent, we first investigated using a shared-memory model. However, after experiencing several problems with race-conditions, we eventually moved to a multi-threaded model using sockets to communicate between the GUI and logic core threads. This technique was not only easier to program, but also made the message passing between the two threads clearly visible, simplifying testing. This method has also proved quite extendable, as we have begun to add additional devices to the CedarLogic palette.

The first CedarLogic delivery consisted of 61 C++ source files (both headers and source code) and one gate description file following the extendable markup language (XML) token format. Overall, this constitutes 12,700 lines of computer code, not including over 3,000 lines of comments.

## Explanation of the Key features of CedarLogic

CedarLogic is a native Windows application complete with a setup Wizard (see Figure 3)

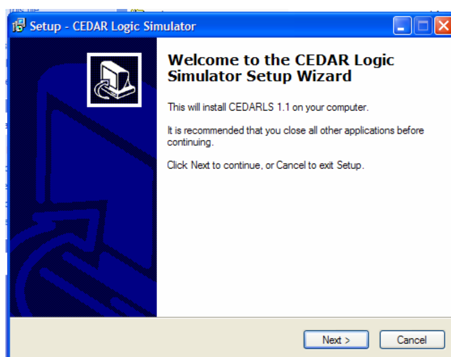


Figure 3 Setup Wizard

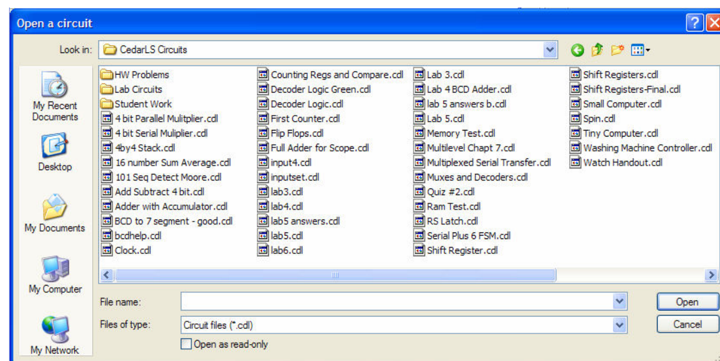


Figure 4 Open Dialog Box

Once installed a standard windows interface presents itself to the user. If previous files have been saved, a standard Windows Open Dialog box (Figure 4) greets the user and allows him to change directories and select the appropriate file. CedarLogic files use the extension .cdl which does not conflict with common file extensions. Figure 5 shows an example of CedarLogic implementing and simulating a full adder.

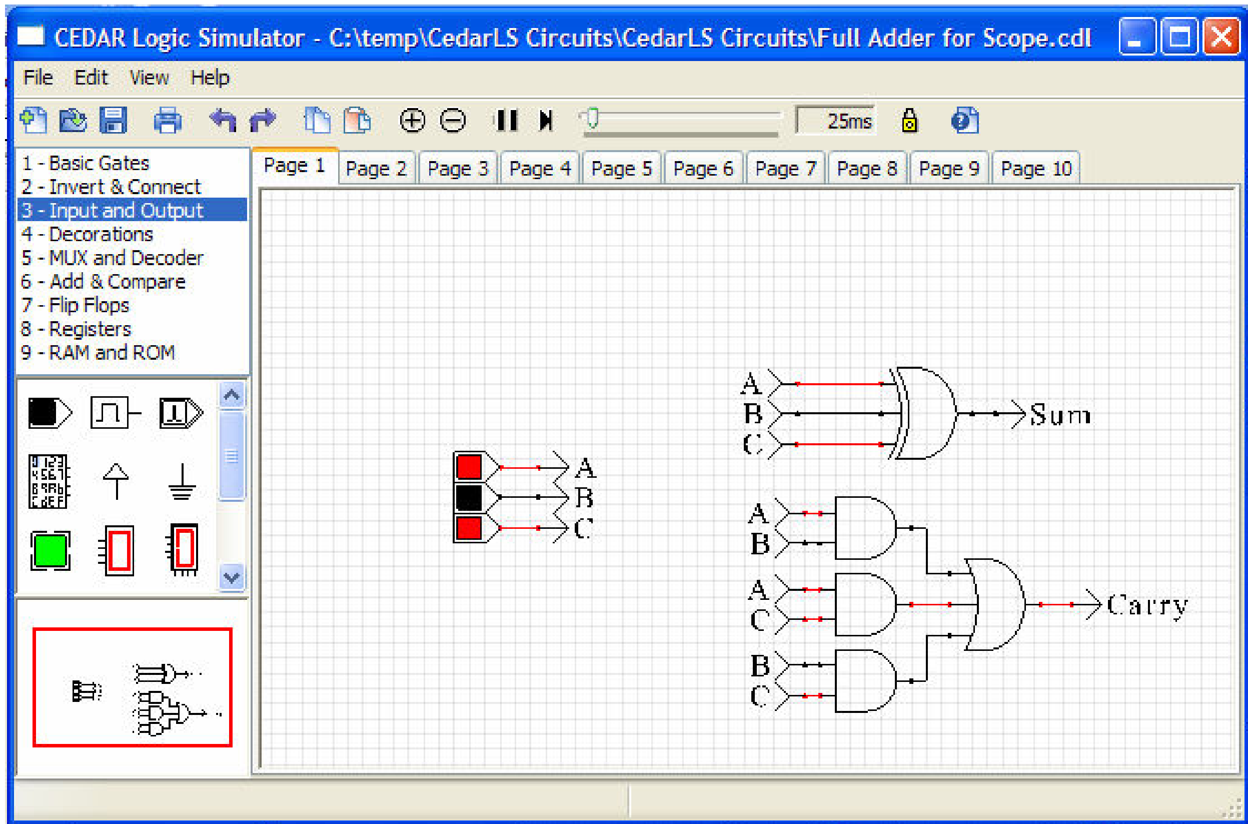

Figure 5 Basic CedarLogic Interface

Notice in Figure 5 that the wires are either red (logic high or 1) or black (logic low or 0) and that the program is immediately simulating the circuit. The user can drag and drop new components onto the canvas, rearrange the position of a gate with rubber banding wires, select, cut, copy and paste. Additionally, in the view menu, an O-Scope view is available to capture a waveform trace of the simulation. This view can be easily exported as a bitmap to the windows clipboard for pasting into a word processor. Figure 6 shows an example of the O-Scope tool.

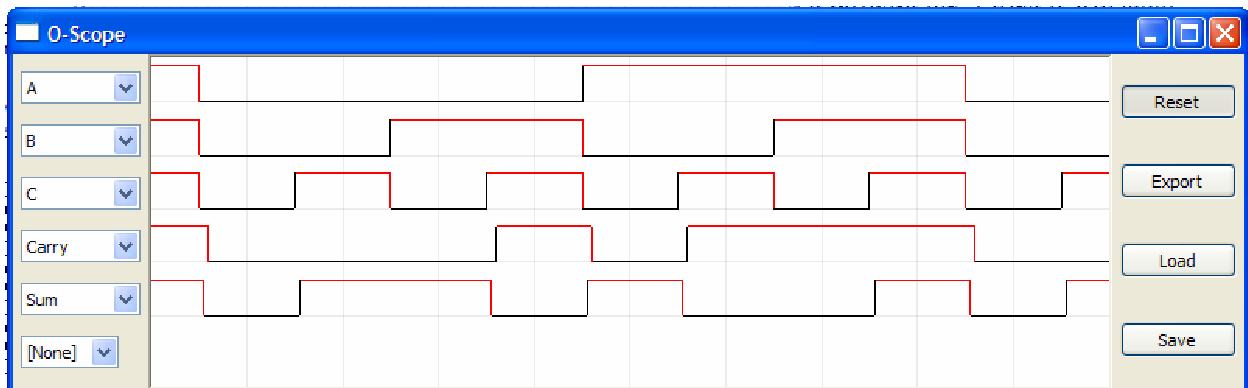

Figure 6 CedarLogic O-Scope View

To aid the new user, a complete help system is available as shown in Figure 7. The help system has Contents, Index and Search tabs. A number of easy-to-follow tutorials are also provided to

help the new user understand CedarLogic's features. Figure 8 shows a detail of the tool bar and page tabs.

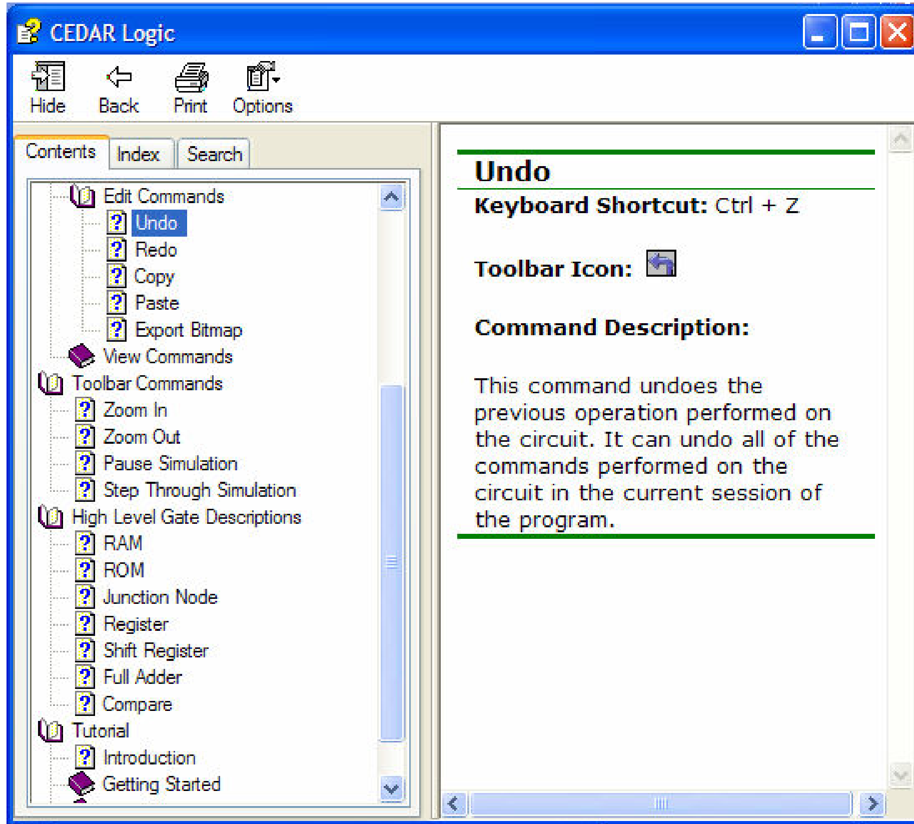


Figure 7 CedarLogic Help

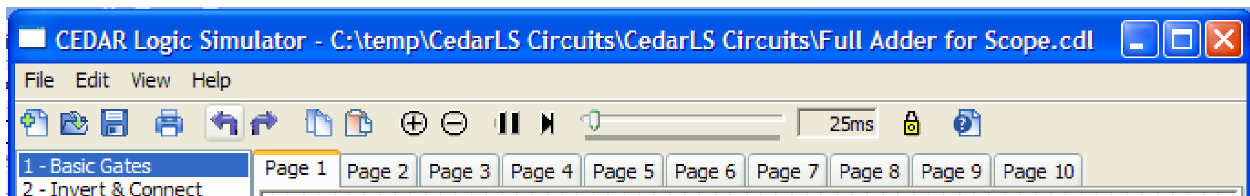


Figure 8 Tool bar and Page Tabs

Cedarlogic's target audience is freshman computer science and engineering students so it was designed to be simple to use. To reinforce this simplicity, we wanted an application that followed windows conventions, therefore all basic windows operations are present in their expected places using familiar menu and toolbars. Common operations such as New, File Open, Save, Print, Undo, Redo etc. have a corresponding icon on the tool bar and feature pop-up help when the cursor floats over them. A slider is provided to aid in adjusting the speed of simulation. Ten page tabs appear across the top allowing very large logic circuits to be built and interact with circuits on the other pages. Signals are allowed to communicate across pages through the use of off page connects called "To" and "From". Pressing the space bar centers and zooms the circuit to maximum viewable size, and the cursor keys allow for easy and intuitive movement around the circuit. Another interesting and helpful feature is the "mini-map" in the lower left hand corner of the main window. This birds eye view of the circuit allows the user to

see the full extent of all components with a red box showing the portion of the circuit that appears on the main page.

The Library pallet is organized into nine categories and contains all the basic logic functions needed to make complex logic circuits. Two, three, four and eight input basic gates are provided. A variety of inverters and tri-state buffers are found in the Invert & Connect category along with the To and From off page connectors. The Input and Output category contains logic switches, a clock, keypad, power, ground, LED's and two types of seven-segment displays. The Decorations category provides a line of text to document a circuit or make a written comment. Two, four, eight and 16 input/output muxes and decoders are provided in the Mux and Decoder category. These palettes are shown in Figure 9.

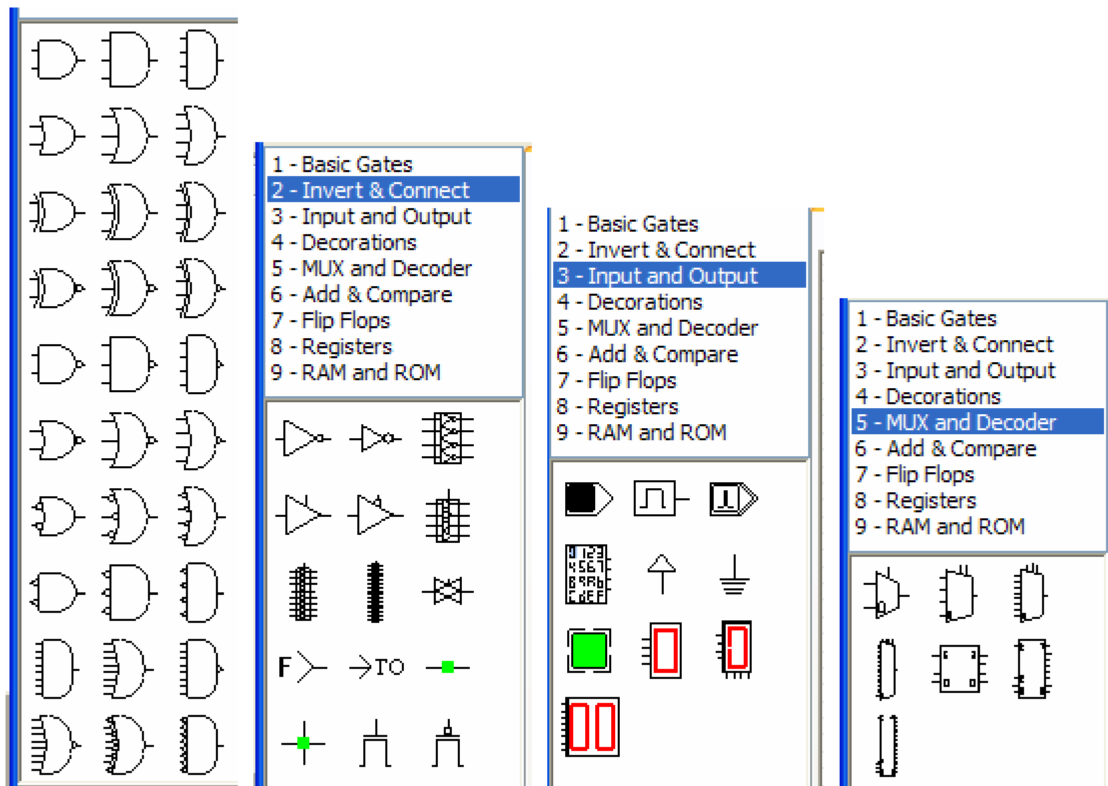


Figure 9 First five gate categories

The Add & Compare category contains a 1-bit and 4-bit full adder and a 4-bit cascadeable comparator. D and JK flipflops are provided in the Flip Flops category. In the Registers category, 4, 8, 12, and 16-bit registers are available along with a 4-bit shift register. These counting registers are capable of resetting, counting up or down, holding or performing a parallel load. Finally, the ninth category of RAM and ROM contain 4, 8, 12 and 16-bit chips. These categories are shown in Figure 10.

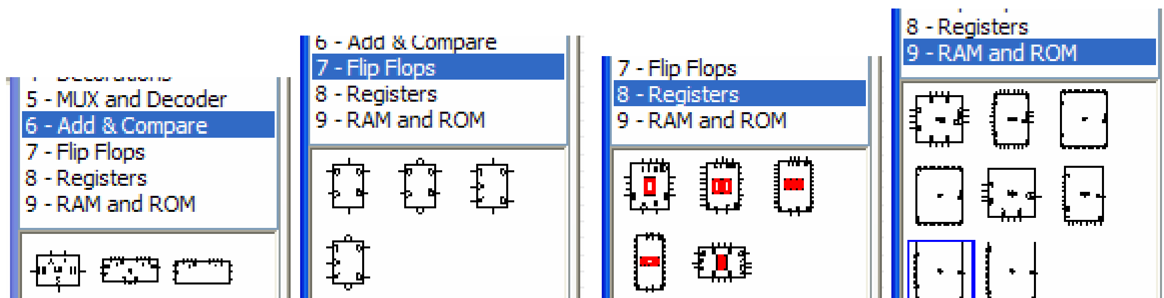

Figure 10 Final four gate categories

This complete set of gates fully supports all designs and homework assignments given in the Digital Logic Design course.

**Student Response**

In the Fall 2006, two sections of Digital Logic Design, consisting of over 60 students were required to use CedarLogic to complete both homework and laboratory assignments. The student response was very positive. Because of its standard interface, students learned the program quickly and would able to concentrate on learning the subject matter rather than be distracted by the oddities of the former Diglog interface. The ability to undo mistakes, easily export a bitmap of the circuit to the clipboard in a single operation, paste it into their word processor, and to save and load circuits using a standard dialog box were among the most popular improvements over Diglog. It has been exciting to have some students just play with the program and create interesting and fun circuits. Large multi-page circuits have been successfully completed with hundred of gates and components. Figure 11 shows a class project near the end of the course that implements a practical washing machine controller. This file gives a good representation of a more complex circuit.

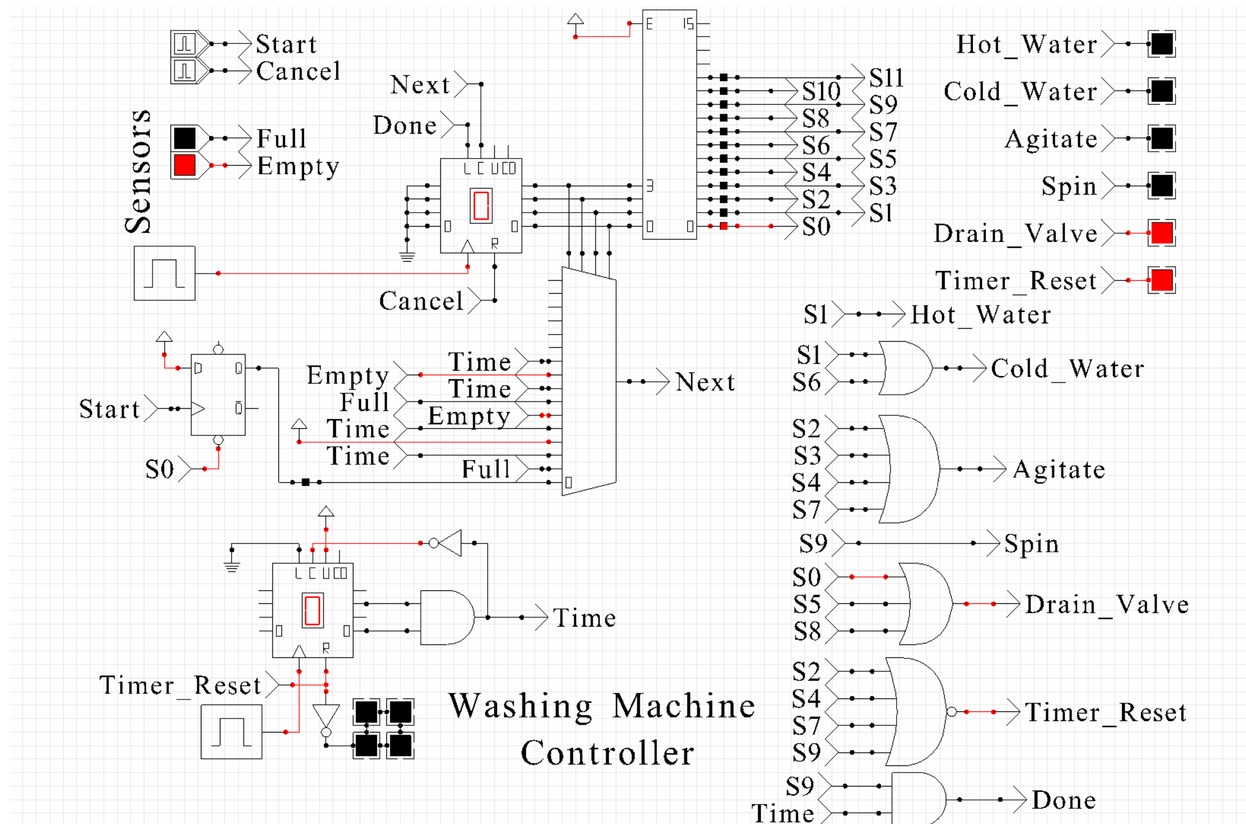


Figure 11 Washing Machine Controller Circuit

**Conclusion**

This project was a success in many dimensions. The final result is a robust, simple, and full-featured application for digital logic simulation. The senior design students had a good experience designing, coding, and testing their work.

We hope CedarLogic will be helpful to students and instructors at other institutions as it has been here at Cedarville University. In addition, we hope the readers find useful suggestions in this paper to develop their project-oriented courses in a manner that motivates students to develop software that has serviceability beyond their college years.

**Acknowledgment**

1.. UC Berkeley, CS Division 387 Soda Hall, Berkeley CA 94720
(http://www.cs.berkeley.edu/~lazzaro/chipmunk/)

2. http://wwwags.informatik.uni-kl.de/utils/DIGLOG/main.html

3. The QuartusII design software delivers the highest productivity and performance for FPGAs, CPLDs, and structured ASICs and offers numerous design features to accelerate the design process http://www.altera.com

4. WIMP stands for Windows, Icons, Menus, and Pointers and is the predominate user interface for interacting with computers and applications. The WIMP interface was developed at Xerox PARC in the mid-80's and popularized by Apple computer and later by Microsoft Windows operating systems.