Circuit Modeling in DLSim 3

Richard M. Salter, John L. Donaldson, Serguei Egorov, Kiron Roy

Computer Science Department
Oberlin College
Oberlin, OH 44074

rms@cs.oberlin.edu, john.donaldson@oberlin.edu, segorov@cs.oberlin.edu, kroy@cs.oberlin.edu

Abstract

DLSim 3, is a GUI-based digital logic simulation program developed by Richard Salter at Oberlin College, extends the capabilities of such programs through the use of Java plug-ins. DLSim 3 makes it it possible to use the software for digital design at higher levels of abstraction. With DLSim 3, we are able to present the many levels of circuit design in a single environment, from low level combinational and sequential circuits through models of complete CPUs. This paper shows how DLSim 3 has been used in the classroom to model several CPUs well known to educators, and to support creative efforts on the part of students of Computer Organization.

1 INTRODUCTION

Many excellent GUI-based logic simulation systems have been developed and are available for download [1, 2, 3, 5, 8]. These systems are very useful for studying basic combinatorial and sequential circuits. While they generally all provide some abstraction mechanism ("black boxes") that permits circuit reuse, they are limited by their GUI based environments to relatively small models.

DLSim 3 [9, 10, 4] joins this group but goes beyond these efforts through its innovative use of Java plug-ins. A plug-in is a software module, written in Java, which is added to DLSim's design platform and can be used as a component in more complex circuit designs. The user can write Java modules that simulate higher-level logic components (i.e., memories, registers, ALUs) which supplement DLSim's collection of built-in components. The plug-in facility is built around an interface that describes what functions a plug-in must perform in order to be installed in the system, and an API of functions which support the writing of plug-ins.

Plug-ins allow the simulator to scale up to whatever level of abstraction is appropriate in a course. For example, at one stage in a course, students might be asked to design an ALU using only logic gates. Later, when studying CPU design, the instructor might provide an ALU plug-in to be used as a component.

In addition, plug-ins can perform I/O, either GUI-oriented user interaction or file operations. For example, we have written plug-ins which can load a microprogram from a file, store results of a simulation to a file, and provide an interactive keypad for a simulated calculator.

The design philosophy behind DLSim was described by Salter and Donaldson in [10]. In this paper we describe how we have used DLSim to visually simulate the CPU designs of Patt and Patel [6] and Warford [12]. With these designs, we have been able to illustrate important CPU design concepts such as datapath construction and and control unit design. In addition, we describe how we have used DLSim in the classroom in the Computer Organization course at Oberlin.

2 Plug-ins

The power of DLSim 3 to model large-scale circuit components, such as RAM chips and CPUs, is achieved through plug-ins. Plug-ins are described in detail in [10]. Here we give a brief summary.

A plug-in is a Java class which represents a circuit component. Every plug-in is a subclass of DLPlugIn, which gives it the basic structure it needs to fit into the DLSim 3 simulation engine.

By default, DLSim 3 displays a plug-in as a rectangle with inputs on the left and outputs on the right. The programmer may, however, provide a customized view for the plug-in by writing a separate view class.

3 CPU design

DLSim simulations for the well-known CPU models MicroMIPS [7] and Mic-1 [11] are given in [4]. To these we now add LC-3 [6] and Pep/8 [12].

3.1 LC-3

The LC-3 CPU is a 16-bit RISC architecture that forms the basis for the study of assembly language and computer organization in Patt and Patel [6]. There are 8 16-bit general purpose registers and a 64K word-addressable memory. The instruction set, described by Patt and Patel as "rich, but lean," consists of 15 instructions. All instructions are 16 bits long, with a 4-bit opcode.

Implementation of the LC-3 in DLSim 3 was facilitated by the level of detail that is provided in Patt and Patel, and by the simplicity of the design itself. Our datapath follows closely the logic diagrams presented in the text.

The LC-3 control unit uses a microprogram consisting of 49-bit microinstructions, 39 bits for control signals and 10 bits used for microinstruction sequencing. It is implemented with a 2^6 x 49 bit control store and a hardwired microsequencer. In our DLSim 3 version of the control unit, we used the microprogram from the text, but we used a different approach for its implementation. Instead of the control store and microsequencer, we used a finite state machine plug-in that has the functionality of the two combined, and which we could use for both the LC-3 and the PEP/8.

3.2 PEP/8

The PEP/8 CPU [12] is an educational CPU designed by J. Stanley Warford at Pepperdine University. It is a CISC architecture with a set of 39 machine instructions, each of which is either one or three bytes long. Eight addressing modes are supported. Memory consists of 2^{16} bytes. In contrast to the MicroMIPS and LC-3 CPUs, this machine's ISA-level architecture is significantly different from its microarchitecture. At the ISA level, it is a 16-bit CPU, with two 16-bit registers (an accumulator and an index register) and a 16-bit external databus. ISA instructions operate on 16-bit quantities. Internally, however, it is an 8-bit machine, with an 8-bit ALU, a set of 32 8-bit registers, and 8-bit internal buses. The two programmer-visible registers are mapped onto pairs of the internal registers. To perform an ISA instruction (e.g., add, and) on two 16-bit values, it is necessary to use two cycles of the datapath through the 8-bit ALU.

As with the other CPUs, the PEP/8 datapath, shown in Figure 3, was laid out using a combination of plug-ins and gate-level components, following the design presented by Warford. Plug-ins were used for most of the components, including muliplexers, adders, sign-extenders, the register file, and the finite state machine-based control unit. Two versions of the ALU were implemented: a plug-in version and a deep circuit version.

The size and complexity of the PEP/8 instruction set (39 instructions, 8 addressing modes, 2 instruction lengths) led to an increase in the complexity of the microprogram used to implement it. In addition, it was necessary to design our own microinstruction format. We were able to implement it using the same FSM plug-in component that was used for the LC-3. The microprogram is loaded from a file, making it easy to modify.

4 Design Issues

Component Reuse. In general, we have tried to reuse components wherever possible. DLSim facilitates component reuse through parameterization of plug-ins. For example, the MUX plug-in that we use in all of the CPU designs has parameters for the number of inputs to select from and the bit width of the inputs. The same Register File plug-in is used in the MicroMIPS (32 32-bit registers), the LC-3 (8 16-bit registers), and the PEP/8 (32 8-bit registers). On the Mic-1, the CPU registers have a more irregular structure, so a custom plug-in was used. Other examples of reusable components are plug-ins for a single n-bit register, an n-bit adder, a mxn random-access memory. and the FSM plug-in described in the next subsection. On the other hand, sometimes the irregularity of the components dictates the creation of custom plug-ins. The ALUs of the four machines are significantly different with respect to their implemented functions and function encoding, so each of these was designed as a separate DLSim component.

Control Unit Design. The four CPUs we modeled differ most significantly in their approach to control unit design. The Mic-1 is fully microprogrammed, allowing for implementation of long code sequences in a single ISA-level instruction (e.g., JVM's InvokeVirtual). All of the versions of the MicroMIPS use hardwired control (Patterson and Hennessy present a microprogrammed version in their text which we did not implement.) Every instruction requires the same number of clock cycles; the "control unit" is really just a decoder for the opcode bits of the instruction.

Both the LC-3 and PEP/8 are microprogrammed. Patt and Patel provide a complete microprogram in the form of a flowchart; translating the flowchart into binary microoperations is left as an exercise. Microinstruction sequencing is performed by a hardwired component, using a clever mapping of opcode bits. Warford, on the other hand, does not describe the control unit of the PEP/8 in detail. The requirements for these two control units are essentially the same: they both require a microinstruction sequencer that can step through a microprogram. The sequencer can be modeled as

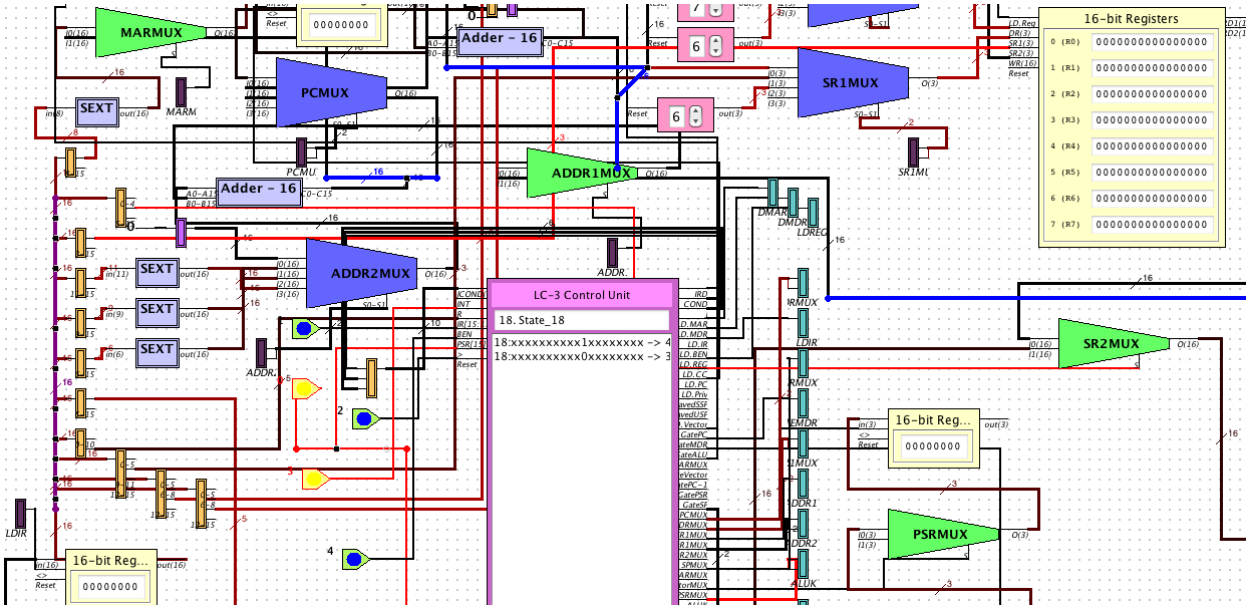

Figure 1. LC-3 Data-path (subsection)

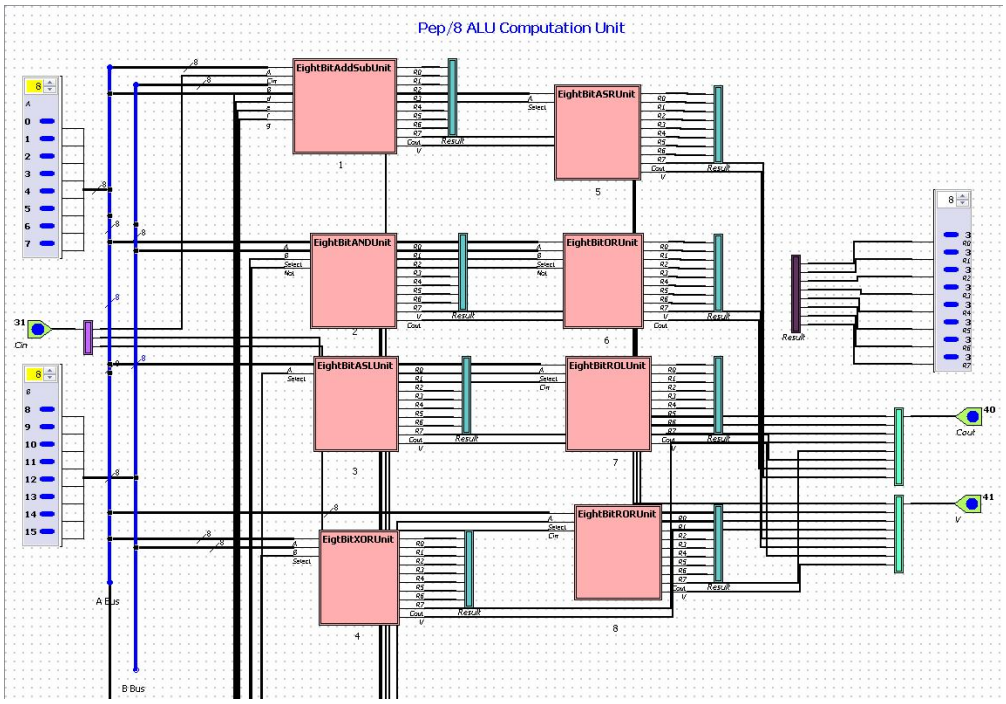

Figure 2. Pep/8 ALU

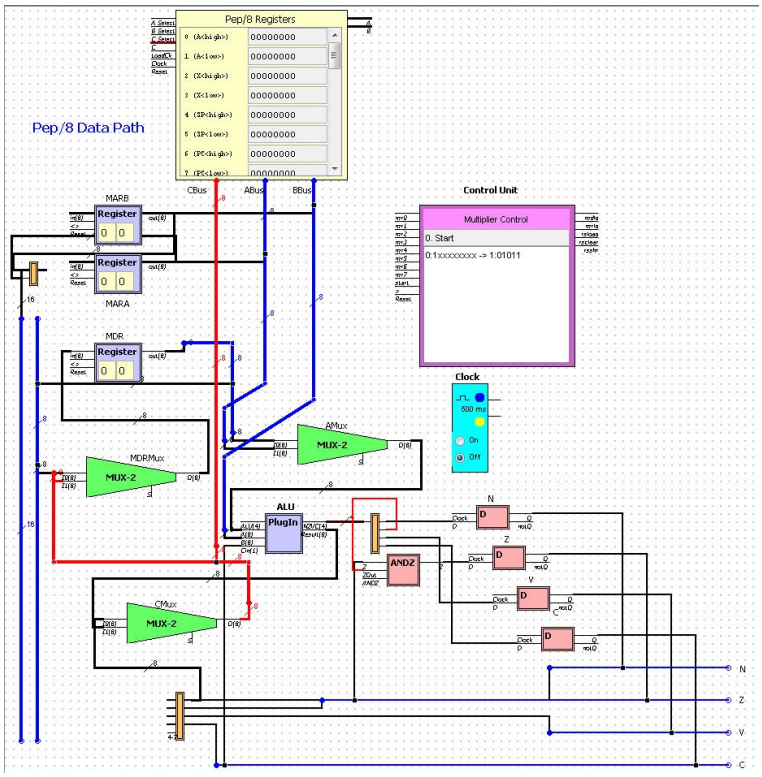

Figure 3. Pep/8 Data-path (subsection)

a finite state machine (FSM), which we implemented as a plug-in which is used for the control units of both the LC-3 and the PEP/8. The inputs of the FSM are the clock and several feedback lines; its outputs are the control signals. Internally, it keeps a state register (actually just a Java variable) to maintain the current state of the FSM, a transition table for the FSM, and an output table. The microprogram is loaded into the tables from a file prior to starting the machine simulation.

Plug-ins vs. Deep Circuits. DLSim 3 provides two ways to scale a design up to large-scale circuits. In addition to the plug-in facility, it is possible to build larger circuits using only basic gates by building several layers of progressively more complex components, using the *card* and *chip* abstractions described in [10]. We call this the *deep circuit* approach. A good example of this is found in the design of an ALU. It is complex enough to warrant a layered approach, but not so large that it would overwhelm the simulator if implemented at the gate level.

The ALUs of both the Mic-1 and the PEP/8 have been implemented both as plug-ins and as deep circuits. Figure 2 illustrates the deep circuit version of the PEP/8 ALU. Each box in the figure is a functional component performing a different operation: And, Or, Add/Subtract, etc. Clicking on any of these components will show a deeper circuit diagram, illustrating how it is designed using lower-level components, and so on, until all of the components are comprised only of gates.

Both approaches have pedagogical value. With a deep circuit, the hierarchical view of digital design is made clear. With a plug-in, the focus can be placed on the functionality of a component as a black box, while hiding the details of its implementation.

5 DLSim in the Classroom

At Oberlin College, the computer organization course is offered once per year and covers binary and hexadecimal arithmetic, digital logic, assembly language programming, and microarchitecture/microprogramming. The digital logic unit lasts about 3 weeks, with one homework assignment using DLSim to construct a circuit. Because of DLSim 3's capacity to model a complete CPU, it is also being used in the microarchitecture unit of the course to demonstrate the Mic-1.

5.1 Computer Organization Assignment

DLSim 3's plug-in feature has allowed for a more creative approach to the circuit design assignment. Students

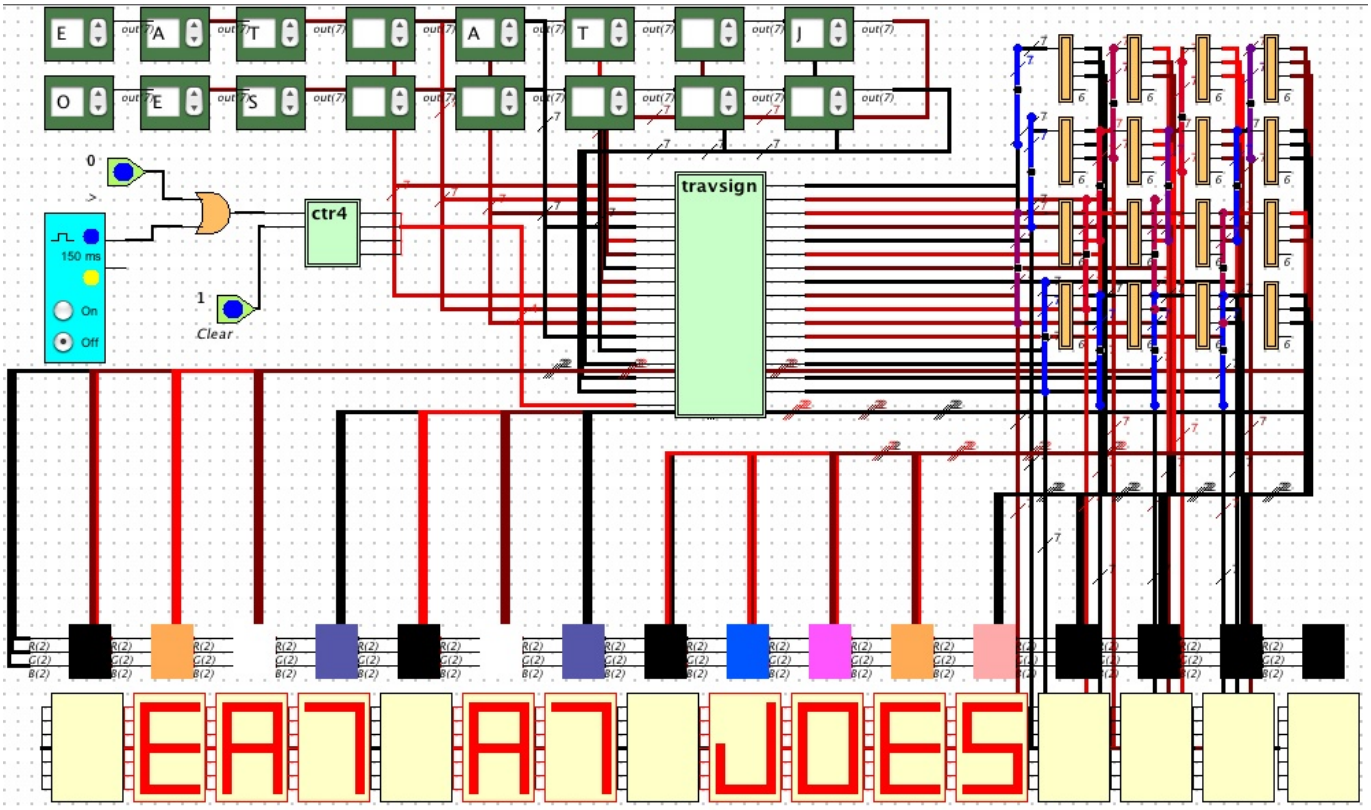

Figure 4. Traveling Sign (with marquee)

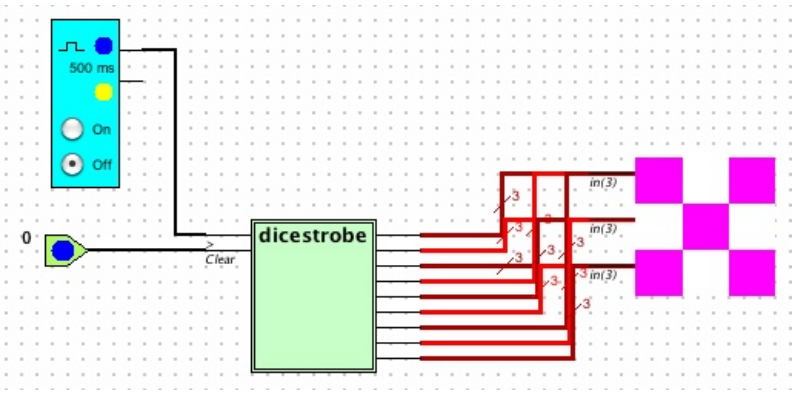

Figure 5. Dice

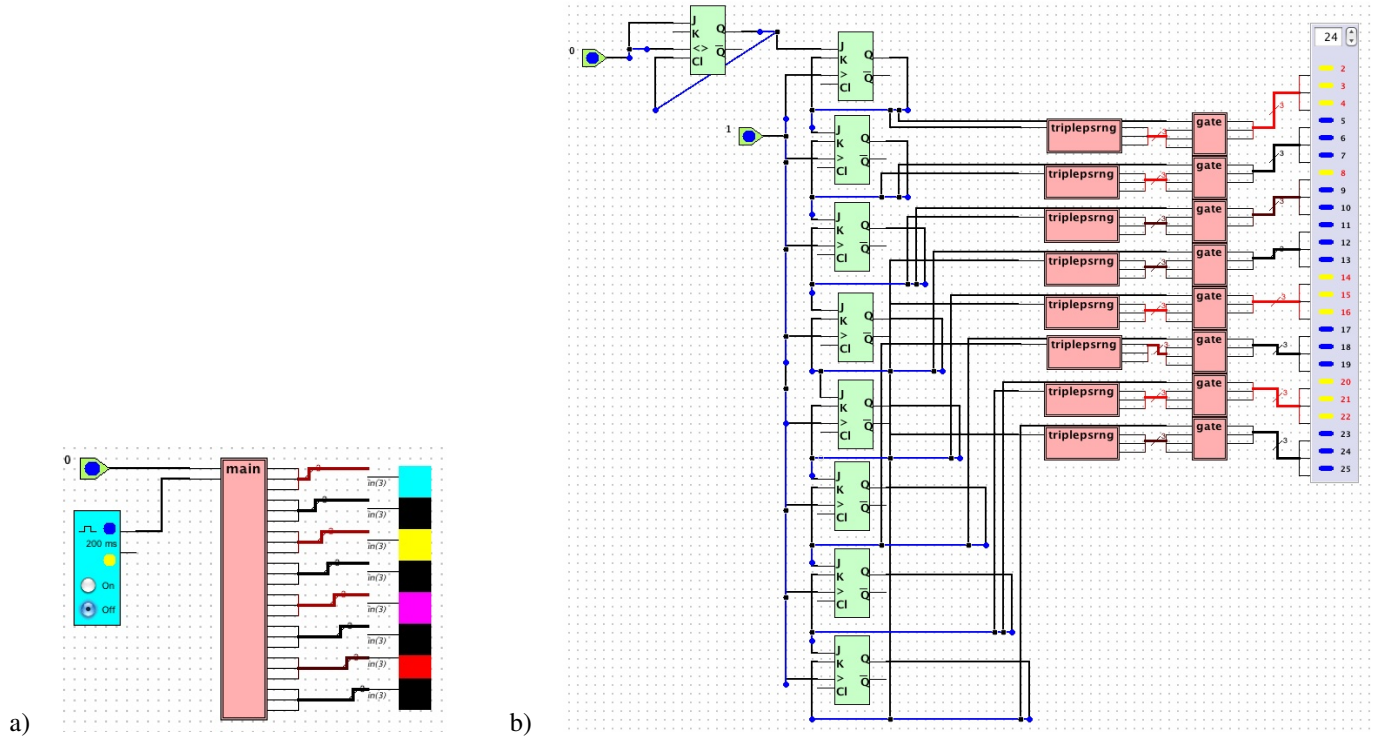

Figure 6. Fireworks Generator: a) top level; b) main circuit

were provided with plug-ins used to model visual display elements and drivers, such as color tiles and LED displays. They were required to implement a periodic display consisting of either a moving or flashing sign or a color display. The display could include randomness, but could not be simply random; i.e. it had to have some periodic repetition that required the implementation of a counter.

The students received a plug-in archive containing the following display elements:

- **LEDDriver:** a component that forms letter and numeral patterns on a 7-segment LED;[1]
- **Pixel and SuperPixel:** a tile that displays, respectively, 8 and 2^{24} different colors;[2]
- **PRNG:** a pseudo-random number generator;
- **FSM:** a finite-state-machine-based control unit.

As an example of the moving sign, the students were given the circuit shown in Figure 4. The 16 LEDDrivers can be seen in the upper left part of the figure. The spinners in the LEDDrivers allow the user to program the sign's message. In addition to the sign itself, which uses DLSim 7-segment display components, colored tiles (SuperPixels) appear above the message. Each tile's color is determined by the bits used to configure the letter appearing in the corresponding LED. The colors move with the letters, from right to left, producing a "marquee" effect. Note that the control logic for this circuit is contained in the **travsign** and **ctr4** chips; consequently this circuit could be distributed to students as a live example without also providing a solution to the assignment.

A second example is shown in Figure 5. The display is a square of 9 Pixel plug-ins that rotate through a "dice"-like presentation of the numbers 1 - 6. Each number is displayed using a different color from the 3 primary (red, green, blue) and 3 secondary (cyan, yellow, magenta) colors displayable by a Pixel plug-in. Once again, the **dicedemo** chip allows this circuit to be safely distributed.

5.2 Results

The resulting student projects were ambitious and showed great design ingenuity. (Running version can be viewed as applets at http://www.dlsim.com/demos.html .) They include a sign that flashes in a periodic pattern and a color "snake" that travels through a square display. Here is the description of a DLSim Fireworks Generator shown in Figure 6.

This circuit simulates exploding fireworks. It

[1] The 7-segment LED is a standard component in DLSim.

[2] Initially only the Pixel was distributed; the SuperPixel was added by popular demand.

consists of a counter which turns on a single bit from the LSB to the MSB of 8 bits in sequence. The counter is modified so that the final JK flip flop does not feed back to the first. This causes the circuit to switch between 01010101 and 10101010 when the first flip flop's input is 0. The circuit implements a control which allows the user to "shoot" a firework. This causes the first flip flop to output a 1, then a 0, then a 1, and so on until the shoot button is turned off. The alternating 1's and 0's ripple through the rest of the flip flop outputs. The flip flop outputs are sent to circuits which trigger random number generators, then filter out 0s, then gate the output so that 0 is output unless the specific flip flop is outputting 1. This number is then sent to a pixel. Each pixel represents a ring of the firework, with the top pixel representing the center ring.

Motivated by the high impact of the display elements, students lent their creative energies to designing unusually sophisticated circuits. They organized their designs in a hierarchical fashion, and showed uncharacteristic patience in fashioning their designs into running models.

5.3 Advanced Course

Beyond the Computer Organization course, DLSim 3 is appropriate for use in independent student design projects and more advanced courses. We have supervised several students in a variety of projects which were described in detail in [4]:

- writing a cache memory plug-in and interfacing it to the MicroMIPS processor model;
- designing a hand calculator which used the MicroMIPS for its computations. This project involved writing a plug-in to represent the calculator keypad and display, and interfacing it to the processor;
- writing a plug-in to represent a PLA;

Oberlin also offers an upper-division elective in Computer Architecture which covers topics such as pipelining, cache memory, and multiprocessor design. Because the course is not offered on a regular basis, we have not yet had the opportunity to use the new features of DLSim 3 in the course. We do anticipate, however, that DLSim 3 will prove to be useful in teaching the course in the future. In particular, our model of a pipelined version of the MIPS processor, described in [4], will be especially valuable in illustrating concepts such as data and control hazards, stall cycles, and forwarding.

6 Conclusion

DLSim 3 is a highly versatile software tool designed for the design and simulation of digital logic circuits. Through its use of software plug-ins, it is capable of simulating digital circuits of varying levels of complexity. The examples presented in this paper demonstrate some of the ways that DLSim 3 can be used as a pedagogical aid in Computer Organization courses. We have been successful in using it to simulate the CPU designs presented in the popular textbooks of Tanenbaum, Patterson and Hennessy, Patt and Patel, and Warford, in order to demonstrate a variety of implementation techniques; for example, hard-wired control, microprogramming, and pipelining. DLSim 3 has also strongly motivated students to produce their own creative circuit designs. We intend to continue to expand our library of plug-ins and circuits, which are available, along with the DLSim 3 software, at our website, www.dlsim.com.

References

[1] D. L. Barker. Digital works 3.0 . `http://matrixmultimedia.com/datasheets/eldwk.pdf`, 2006.

[2] C. Burch. Logisim: A graphical system for logic circuit design and simulation. *J. Educ. Resour. Comput.*, 2(1):5–16, 2002.

[3] C. Burch. Logisim 2.1.6 `.http://ozark.hendrix.edu/ burch/logisim`, 2007.

[4] J. L. Donaldson, R. M. Salter, A. Singhal, J. Kramer-Miller, and S. Egorov. Illustrating CPU design concepts using DLSim 3. In *FIE'09: Proceedings of the* 39^{th} *ASEE/IEEE Frontiers in Eduction Conference*, pages T4G–1 – T4G–6. ASEE/IEEE, October 2009.

[5] A. Masson. Logicsim . `http://wuarchive.wustl.edu/edu/math/software/mac/LogicSim/`, 1996.

[6] Y. N. Patt and J. J. Patel. *Introduction to Computing Systems: From Bits and Gates to C and Beyond,* 2^{nd} *Edition*. McGraw-Hill, New York, 2004.

[7] D. A. Patterson and J. Hennessy. *Computer Organization and Design,* 3^{rd} *Edition*. Morgan Kaufmann, Palo Alto, CA, 2004.

[8] 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, June 2007.

[9] R. M. Salter and J. L. Donaldson. Using DLSim 3: a scalable, extensible, multi-level logic simulator. In *ITiCSE'08: Proceedings of* 13^{th} *Annual Conference on Innovation and Technology in Computer Science Education*, page 315. ACM Special Interest Group on Computer Science Education, June – July 2008.

[10] R. M. Salter and J. L. Donaldson. Abstraction and extensibility in digital logic simulation software. In *SIGCSE '09:*

Proceedings of the 40th ACM technical symposium on Computer science education, pages 418–422, New York, NY, USA, 2009. ACM.

[11] A. S. Tanenbaum. *Structured Computer Organization,* 5^{th} *Edition*. Prentice-Hall, Upper Saddle River, NJ, 2006.

[12] J. S. Warford. *Computer Systems,* 4^{th} *Edition*. Jones and Bartlett, Boston, 2010.