DLSim 3: A Scalable, Extensible, Multi-level Simulator for Computer Organization Courses

Richard M. Salter and John L. Donaldson
Computer Science Department
Oberlin College
Oberlin, OH 44074
rms@cs.oberlin.edu, john.donaldson@oberlin.edu

ABSTRACT
Students of Computer Organization should be able to “learn by doing” at all levels of computer design. This paper describes the multilevel simulation system DLSim 3, a unified platform for studying system structure, from low level combinational and sequential circuits, through design of a complete CPU. Among other features, DLSim 3 is able to accomplish this by providing three different levels of circuit abstraction: cards, chips, and plug-ins. Using DLSim 3, students recognize the uniformity of system structure, as well as the principles of abstraction that link the various levels of design.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education; K.3.1 [Computers and Education]: Computer Uses in Education—computer-assisted instruction

General Terms
Design, Experimentation

Keywords
Computer Architecture, Logic Circuit Simulation, Abstraction

1. INTRODUCTION
One objective of any good computer organization course is to demonstrate how a complete CPU can be designed using only gates as fundamental building blocks. However, the vast structural complexity present in any reasonable CPU architecture makes it challenging for students to appreciate the unifying concepts that are foundational to any such architecture, and which appear at all the various levels of abstraction. This challenge is made greater by a lack of pedagogical tools that uniformly address structural design at every level.

For example, together with Tanenbaum’s text [8] we have for many years been using the logic simulator DLSim 2 (created and built by one of this paper’s authors) to illustrate simple combinatorial and sequential circuits. DLSim is easy to use and supports useful features such as circuit abstraction (described below). Moreover, its ability to produce open, XML-based descriptions of its circuits has made DLSim attractive as a front end for experimentation with other software tools. In fact, it was used successfully in an NSF-funded project to drive the programming of field programmable gate arrays (FPGAs) [4]. Eventually the course moves on to consider higher level designs, such as busses and CPUs. While we know that the products of the logic level must be there to provide the necessary functionality, for the sake of controlling complexity we no longer worry about their fine structure. We also move on to other, higher-level, teaching tools to drive home the details of those levels.

At Oberlin, we consider the microcode-based CPU architecture MIC-1 [8], and we use a home-brewed simulator that focuses only on the microcode and ISA implementation. Unfortunately, the change in simulation environment necessitated by the change in level of abstraction has the unintended side-effect of creating a disconnect in student minds. It is difficult for students to appreciate the reality of the underlying structure of the CPU-level (i.e., it is built from the stuff presented some weeks ago at the digital logic level). More importantly, the principles of abstraction and the structural invariants that link the levels and control the complexity (e.g., as elucidated by the “virtual machine” concept [8]) become especially difficult to grasp.

In order to address this problem we have created DLSim 3, which extends the features of the earlier implementation to address the wider spectrum of levels present in CPU design. It does this through vertical scalability as well as horizontal extensibility, as described below.

In the next section we describe some of the features that empower DLSim 3 to act effectively as a multi-level simulator. We then provide several examples and comment on how the tool can be used to drive home ideas of homogeneity and structural invariance in computer system design. We continue with comparisons to other simulation software, and conclude with a discussion of future work.

2. DLSIM FEATURES
2.1 Background

Our simulator, Mic1MMV, is now distributed with [8].
DLSim began in 1996 as a Java reimplementation of an early Macintosh application by A. Masson called LogicSim [5]. LogicSim supported GUI WYSIWYG design and circuit abstraction. The latter was achieved by permitting any circuit designed in LogicSim to be inserted multiple times as a “black box” into another circuit (so long as doing so did not create a loop). The black box appeared as a blank rectangle with input and output pins respectively corresponding to the “switches” and “bulbs” (i.e., source inputs and terminal outputs) in the abstracted circuit. This simple capability permitted fairly complex circuits to be created by abstracting to multiple levels, though it was limited by, among other features, an inability to simply express composite elements (i.e., multibit expansions of single bit circuitry) and corresponding wire “bundling” (i.e., multibit data paths connecting composite circuits).

After many years of classroom experience, DLSim 2 replaced DLSim 1 in 2001 with some useful extensions and design features. Among the most important of these was the XML export. In DLSim 1 and 2 each subcircuit is stored in a separate file. Consequently, storage for any given complex circuit is distributed over all of the subcircuits in its substructure. Since management of this arrangement is difficult, the XML export provides a useful alternative for centralizing the description of a given circuit and all of its constituent subcircuits. A single XML file is used to capture the deep description of the circuit. This file can be used to recreate and reconstruct the entire set of circuits used in a given application. Students soon learned to appreciate the XML export as a backup, and instructors appreciate the convenience of the XML file for grading. As an unexpected dividend, the open structure of the XML representation made it possible for an FPGA configuration program to use DLSim-designed circuits as its source, as mentioned in Section 1 [4].

2.2 Objectives for DLSim 3

Students should be able to “learn by doing” at all levels of system design; consequently we wanted to create a unified platform for studying system structure, from low level combinational and sequential circuits, through design of a complete CPU. DLSim 3 is able to accomplish this by providing three different levels of abstraction. To the “black boxes” of earlier versions, now called cards, we add chips, and plug-ins, described in detail in the next section. These abstraction tools provide two crucial factors necessary for such a simulation system to achieve the desired level of expressiveness:

- **Extensibility.** Abstract circuits extend the palette of basic building blocks used to construct larger, more complex circuits.
- **Scalability.** Abstract circuits permit designers to focus on a particular level of design, needing only to understand the functional behavior of lower levels and not their implementation.

DLSim 3 is being used to design examples of complete 16-bit CPUs, including those described in leading computer organization texts by Tanenbaum [8], and Patterson and Hennessy [6].

2.3 DLSim 3 Abstractions

DLSim 3 builds on its predecessors, adding several significant new features that make it ideal for multi-level use. In this section we describe the abstraction features.

2.3.1 Organizing abstraction: the project

DLSim 3 activity is organized around a project that consists of a top-level entity (i.e., circuit, CPU) and its constituent parts (i.e., card, chip and plug-in subcircuits). XML import/export continues to be supported at the project level (i.e., each XML file corresponds to a complete project, and vice versa).

A primary goal of DLSim 3 is to provide fluid navigation between a circuit and its abstracted constituents. Two levels of functionality are provided. The splash display (first introduced in DLSim 2) permits the contents of a card to be opened for viewing in place by simply double-clicking on it. Furthermore, any cards within a splashed circuit may also be splashed. Using the splash feature, it becomes possible to temporarily transcend the design levels with ease, so as to, for example, follow the state of a simulation into lower level elements.

An example of the splash display is shown in Figure 1. In Figure 1a, we see a 4-bit ALU circuit containing four 1-bit ALU cards. In Figure 1b, the top-most 1-bit ALU card has been splashed, revealing its structure. (Note: any of the four 1-bit ALU cards could have been splashed revealing the same content.) In Figure 1c, we further splash the 1-bit ALU’s logic unit card, and could continue into its 2-to-1 mux card if so desired.

A splash box can only be used to view a subcircuit. DLSim 3’s navigation panels provide quick access to each subcircuit type for possible revision. Figure 2 shows the same 4-bit ALU project as Figure 1, however in this case subcircuits are selected for editing by clicking in one of the left-hand navigation panels.

These navigation panels provide immediate access to subcircuits at any depth. Note that the top navigation panel shows the subcircuits according to their pattern of inclusion, while the lower one is essentially a catalog of the subcircuits used in the project. Although a subcircuit may be duplicated throughout the project, only one prototype is maintained so that an edit to a subcircuit is immediately propagated to all instances at all levels.

2.3.2 Exportable abstract subcircuits

A subcircuit can be added to any other circuit in the current project (so long as a loop is not created) by dragging it onto the canvas from the subcircuit list. In this way, subcircuits extend the palette of primitive circuit elements. Circuits (including compound items containing further subcircuits) may also be exported and reused in other projects. There are three types of subcircuit encapsulation, each providing a different level of visibility and functionality:

- **Cards.** A subcircuit may be encapsulated as a card, and may be exported (in binary and/or XML) for use in other projects. When exported and reused, it and all of its subcircuits will be visible. For example, the 4-bit ALU shown in Figures 1 and 2 has several subcircuits that might be useful elsewhere. If one were to, say, export its 1-bit ALU subcircuit as a card, the export would include all cards telescoped into that subcircuit (e.g., the 1-bit addition and logic units and the 2-to-1 mux). Any new project using the 1-bit ALU...
Chips. Encapsulating a subcircuit as a chip has two effects: 1) All of its constituent subcircuits are flattened to the level of primitive elements, and optimized by removing unnecessary connections; and 2) the chip is opaque, appearing as a complete, closed entity in the circuit list without any of its own subcircuits. Thus, for example, if we were to export the 4-bit ALU as a chip, all traces of its subcircuits would disappear. Chips may be inserted into any circuit in the present project without fear of creating a loop. Or they may be saved and inserted into any other circuit. They are complete and closed, behaving functionally as if they were primitive elements.

Plug-ins. Chips are ideal opaque “black boxes” for top-down design, but are limited to abstracting circuits created within DLSim’s GUI canvas. Plug-ins are opaque high-level circuit components derived from Java class files which implement the DLSim plug-in interface. In effect, a plug-in is specified similarly to a circuit defined in hardware description language such as VHDL or Verilog. Plug-ins are crucial for achieving the scalability that allows the same tool to operate at both very low and very high levels.

Chips and plug-ins extend the expressiveness and utility of DLSim in a number of ways:

- A chip or plug-in can be used by instructors as a template for a class assignment. For example, the instructor could supply a simple circuit (e.g., decoder, multiplexer, etc.) written as a chip to demonstrate its functionality without revealing its fine structure, and then assign students the task of fleshing out the details to the level of elementary gates.

- Chips and plug-ins can be used to support top-down design. For example, in designing a CPU, a high-level view of the CPU can be constructed using chips or plug-ins for basic components such as the ALU, random-access memory, control unit, and register file. One by one, the plug-ins or chips can be replaced by cards until the design is complete at the gate level.

- Plug-ins can be used in situations, such as the design of a medium or large scale random access memory, which would otherwise lead to scalability problems if attempted through DLSim’s GUI interface.

- Because they are written in Java, plug-ins can be enhanced with capabilities beyond the requirements of the plug-in interface itself. For example, a plug-in can be used to perform I/O operations, interact with files, or use the keyboard and screen in special ways. A plug-in representing the control store for a microprogrammed CPU, for example, can read the contents of the store from a file. As another example, a plug-in could create a console window which could be used to display the contents of circuit components or supply input values to external bus lines. Plug-ins can also determine their appearance on the DLSim canvas, so that an ALU can “look” like an ALU.
3. EXAMPLES

In this section we consider examples of increasing complexity and show how DLSim 3’s abstraction mechanism create a coherent approach to their structure.

3.1 Example 1: ALU Design

A popular example of modular design is the arithmetic-logical unit, or ALU. The ALU performs basic arithmetic (addition and subtraction) and logical (AND, OR, NOT) operations on a pair of multibit inputs to produce a result of the same width. This example demonstrates a number of fundamental design concepts that recur often in digital circuits, such as bit-slicing, carry-ripple type cascading, data-paths, and control inputs and outputs. It naturally decomposes into clearcut subproblems (i.e., half- and full-adders, multiplexers, and single-bit ALUs).

A 16-bit ALU is a powerful circuit with 35 inputs and 17 outputs, yet the modular approach makes it possible to complete the design while introducing only 8 gates (4 in the multiplexer, 2 in the logic unit, 1 in the full adder, and one in the adder/subtracter), each of which is replicated many times in the final circuit.

Using the bottom-up approach, one would first design the 1-bit half-adder and multiplexer, then incorporate them into the 1-bit full adder. The latter is then incorporated into the 1-bit adder/subtracter. A similar 1-bit logical unit is constructed and combined with the arithmetic unit to form the 1-bit ALU (see Figure 1). Multi-bit ALUs are then built by cascading the carry in/carry out bits. This is the approach most readily supported by current logic simulation software.

DLSim 3, however, will support a top-down approach, whereby various stages of decomposition can be encapsulated in chips. At each stage the chip is replaced by a corresponding card containing some or all of the details of that stage’s design, which may in turn require some additional chips. For example, decomposing the top level may involve the instructor supplying a chip opaqueely implementing the 1-bit ALU.

Students would then build an 8- or 16-bit ALU version from the 1-bit chips using the bit-slice design pattern. When assigned this way, the design pattern itself becomes the focus of the exercise. Implementation of the 1-bit version can continue to follow this top-down approach by supplying chips for half- and full- adders and multiplexers. Alternatively, one could switch to the bottom-up approach. The entire design is made fully visible once the fleshed-out 1-bit ALUs are installed in the multibit circuit.

3.2 Example 2: An 8 Bit Multiplier

As a second example we consider a sequential circuit (i.e., one with state) that multiplies two eight-bit quantities using the standard add-shift algorithm that is presented in most computer organization classes. As an added feature, we include a display module that presents the numerical values using the simulated 7-segment LED display provided as part of DLSim’s primitive element set. Consequently, input and output values are read as hexadecimal values from the “LEDs” rather than directly from binary sources.

This circuit decomposes naturally into several subcircuits that can be provided as chips for top-down design, or constructed from the bottom up and linked together. The subcircuits themselves also exhibit pedagogically important structural patterns, such as bit-slicing and recursion. The principal subcircuits are:

1. 8- and 16-bit registers that act as accumulators for addition but can also left-shift their contents.
2. Multiplication logic, which delivers enabling signals to the various states at times appropriate for the computation.
3. LED logic that drives the simulated displays.

DLSim comes equipped with 7-segment LED display elements that remain visible in circuits abstracted to cards or chips, but without the logic to translate a register pattern into hexadecimal LED output. The LED circuit requires:

1. A decoder for translating binary register values to in-
dividual enabling signals; and
2. A LED driver that translates those signals into the appropriate LED signals to form each symbol.

The main register is built from a single-bit subcircuit and operates in either add or shift mode. The register contains a full adder, alternately using the carry input and output of that adder to perform the left-shift. The 8- and 16-bit registers are constructed similarly to the ALU described above, using bit-slicing with cascading carry and shift values.

This circuit could be assigned in a class as a top-down exercise. The instructor supplies chips implementing the registers and multiplication logic, in which case initial student attention will be primarily on interfacing the multiplication logic with the registers and datapath, to manage data movement and control register function. As with the previous example, filling in the details could combine top-down and bottom-up approaches.

### 3.3 Example 3: Datapath For a CPU

A logical step in development of more and more complex circuits is to design a CPU. This exercise utilizes basic components and techniques which can be developed earlier in the course:

- Decoders
- Multiplexers
- Arithmetic/Logical Unit
- Registers
- Register File

The Computer Organization texts of Patterson and Hennessy [6], Tanenbaum [8], and others present models of instructional CPUs, any of which would be suitable as a course project to be implemented using DLSim 3. This example also illustrates how a top-down design approach could be used in a large project. The CPU datapath could be used as a starting point, with the instructor providing chips and plug-ins for various stages (memory, 32-bit adders, shifters, etc.) so that a running CPU would be simulated. Students would then be able to fill in the details of each circuit component.

### 4. COMPARISONS WITH OTHER SYSTEMS

Many GUI-based logic simulation systems have been developed and are available for download (see [2], [9], for comprehensive, though dated, summaries.) Among the more recent additions are JLS [7], Digital Works 3.0 [1] and Logisim 2.1.6 [3].

Many of these are excellent, and are similar to DLSim 3 in their GUI design, and by supporting some amount of circuit abstraction. DLSim 3, like JLS, supports some advanced design features, such as connection and wire bundling, that simplify construction of larger and more complex circuits. Unlike the systems we looked at, DLSim 3 provides uniform access to all circuits of a project, either for simple viewing (i.e., splash views) or for editing. DLSim 3 also appears to be the only simulator to offer XML-based representation of circuits, and is unique in supporting both the chip abstraction and a plug-in API. The former means that circuits designed in DLSim may be used to drive activities in other contexts (which, as mentioned earlier, has occurred with FPGA configuration). Moreover, as discussed above, there are significant pedagogical advantages to using chips in sample circuits assigned as exercises.

The plug-in API is likely the most significant innovation in DLSim 3, providing unrestricted scaling and extensibility. Both JLS and Logisim scale by providing a fixed library of high-level elements such as flip-flops, registers, memories, etc. DLSim 3’s plug-in facility provides the hooks for incorporating any element that the user might need, at any level. The power of this feature should become apparent as plug-in libraries are developed.

### 5. CONCLUSION AND FUTURE WORK

DLSim 3 has been designed to address the “fragmentation problem” that strikes many students studying Computer Organization, specifically that the myriad of topics covered in that class truly unite to form a unified whole. More importantly, by taking such a holistic approach, the conceptual “glue” of abstraction and recursion is illuminated.

Our current goal is to build a library of cards, chips and plug-ins which, together with set of progressive laboratory exercises, will provide a complete approach to circuit and CPU design. We also plan improvements to DLSim 3’s simulation environment by introducing timing factors and simulation scripts.

DLSim 3 may be downloaded from http://www.cs.oberlin.edu/~rms/dlsim

### 6. REFERENCES


