Experience With the CADAPPLETS Project

John A. Nestor, *Senior Member, IEEE*

***Abstract*—The CADAPPLETS Project provides Web-based animations (implemented as *applets*) of very large scale integration (VLSI) computer-aided design (CAD) algorithms as a learning resource for electrical and computer engineering students, designers, and CAD tool developers. Concepts from software algorithm animation are adapted to illustrate the problem formulation and the operation of physical design algorithms for placement and routing. This paper surveys these animations and describes how they have been used in the classroom at Lafayette College, Easton, PA, by users over the Internet, and in courses at other universities worldwide.**

***Index Terms*—Algorithm animation, physical design, placement, routing, very large scale integration (VLSI) computer-aided design (CAD) education.**

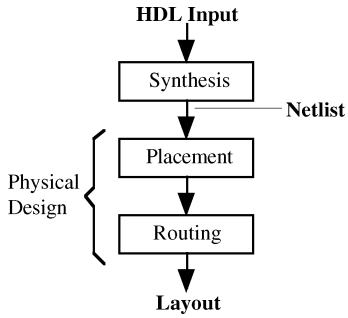

Fig. 1. Simplified design flow for integrated circuit CAD.

I. Introduction

VERY large scale integration (VLSI) is the key implementation technology for modern electronic systems. VLSI chips containing hundreds of millions of transistors are now used in a wide range of electronic products. The design of these complex chips requires an extensive set of computer-aided design (CAD) tools [1], [2].

Fig. 1 shows a simplified diagram of the tools used in the design of an application-specific integrated circuit (ASIC). First, synthesis tools translate a hardware description language (HDL) input into a *netlist* that specifies a set of predesigned *standard cells* and connections (*nets*) between these cells.

Next, *physical design* tools assemble the cells and connections into a *layout* that is suitable for fabrication. The assembly of this layout is usually performed in two steps: *placement*, which assigns specific positions to each cell, and *routing*, which defines the physical connections between the terminals of the cells using a limited number of metal layers. Physical design is followed by *circuit extraction* and *timing analysis* (not shown), which model parasitic capacitances and resistances in the final layout and predict whether timing constraints can be met successfully. If this is not the case, the process is repeated until *timing closure* occurs.

The development of CAD tools has been the subject of extensive research [1], [2]. The general approach to is to first develop a *problem formulation* that models the design problem and provides metrics for estimating the quality of solutions in terms of area, timing, power, and other concerns. Next, an algorithm is selected and applied to the problem formulation.

Manuscript received June 21, 2007; revised September 24, 2007. First published June 17, 2008; last published August 6, 2008 (projected).

The author is with the Electrical and Computer Engineering Department, Lafayette College, Easton, PA 18042 USA (e-mail: nestorj@lafayette.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TE.2008.919650

Electrical and Computer Engineering (ECE) students typically learn about chip design and CAD tools in an introductory VLSI design course [3]–[5] that is taken as a senior-level elective. This course usually starts with a review of CMOS devices, then covers integrated circuit processing and layout, analysis and design of combinational and sequential logic circuits, and the design of subsystems and eventually full chips using CAD tools. Such a course often includes a design project in which students create a complete chip design.

However, in these courses CAD tools are typically treated as "black boxes." Few details are provided about the underlying algorithms. Study of these algorithms is instead relegated to advanced-level courses that are intended for CAD tool developers. The disadvantage of this approach is that it can be difficult for students to understand the capabilities and limitations of CAD tools without some knowledge of the underlying problem formulations and algorithms.

The goal of the CADAPPLETS project [6], [7] is to provide students with this knowledge by adopting the techniques of software algorithm animation [8], [9]. Animations been developed to illustrate the problem formulation and operation of several important CAD algorithms. These animations are useful both in VLSI design courses where students learn to work with the tools, and in CAD algorithms courses where students learn to implement the algorithms. They are also useful for working designers and CAD tool developers who wish to review these algorithms.

The animations are implemented as Java applets. These applets are embedded in Web pages that explain each algorithm and are published at http://cadapplets.lafayette.edu so that they can be accessed by students, faculty, designers, and tool developers worldwide.

This paper describes the development, use, and evaluation of the CADAPPLETS animations, which have been used extensively at Lafayette College, Easton, PA, in an undergraduate VLSI design course and have been used in VLSI design courses and CAD algorithms courses at several other institutions. Web

server logs indicate that hundreds of Internet users access the applets each month to study CAD algorithms independently.

The rest of this paper is organized as follows. Section II describes the background of the project and its relation to prior work. Section III surveys the animations developed to date. Section IV describes how the animations have been used and evaluated. Section V provides conclusions and suggestions for future work.

II. Background

The CADAPPLETS project draws extensively on ideas developed for software algorithm animation in computer science education [8], [9]. Innovations stemming from this research include the use of *multiple views* – graphical renderings of data structures and execution history – and the *interesting events paradigm* – an approach that identifies important events in the execution of an algorithm where the views should be updated.

As an animation proceeds from one interesting event to the next, the position, shape, and color of objects in a view are changed to illustrate data structure changes, plot historical information about execution, and present statistics about the results of the algorithm. Multiple views allow the emphasis of different features of the algorithm.

The advent of the World Wide Web has made it possible to provide software algorithm animation tools over the Internet, most commonly using Java [10] applets. These tools can be implemented as part of a large animation framework (e.g., [11]), or can be written as freestanding applets using the graphical user interface capabilities of the Java environment.

While these techniques have been used extensively for software algorithm animation, only a few animations of physical VLSI CAD algorithms have been developed. For example, [12] and [13] animate fixed-size cell placement using simulated annealing. The APHYDS [14] system provides a framework that allows students to implement physical CAD algorithms and includes some support for visualizing the results. However, APHYDS does not support the interesting events paradigm that is needed for in-depth animation.

Limited visualization aids are also built into commercial CAD tools. For example, placement tools often use a *rat's nest* diagram [2], in which nets are displayed as straight-line segments laid over a diagram of cell placements. This diagram gives the user a way to grasp the size of the nets and the relative congestion of different parts of the placement. Routing tools often display the end result of the routing, or provide displays that can be used to select wires for rerouting. However, these displays are intended to provide the user with feedback about the end result and not the operation of the algorithm.

In contrast, the animations developed in the CADAPPLETS project are intended to provide insight about both the operation of the algorithms, and the end result. Displays like the rat's nest are used in these visualizations, but are augmented to illustrate dynamic behavior. In each of the animations, the interesting events paradigm is used to identify key events for display during animation.

III. A Survey of the Animations

This section surveys the current CADAPPLETS animations, which focus on placement and routing.

A. Placement

The goal of placement is to assign physical locations and orientations to a set of connected cells, while minimizing a cost function that estimates chip area and routing quality. Depending on the design style used, cells may be of varying size, or may be fixed in size in one or two dimensions. Cells may be rotated or reflected about an axis.

The typical problem formulation for placement represents cells as rectangles on a planar surface with a *cost function* that provides a measure of placement quality – usually a weighted sum of chip area (estimated as the bounding box of the cells), cell overlap (used as a penalty function), and routing quality. Routing quality is typically estimated by a prediction of wire length and wire congestion.

Wire length can be estimated in several ways. One of the simplest (and most popular) methods is to calculate the "half perimeter" of a bounding box containing the terminals of a net. Other approaches include calculating the length of either a minimum spanning tree, or a Steiner tree [1], that connects the terminals of a net. Congestion is usually estimated using a count of the number of nets that cross a set of cut lines on the layout surface.

Given this problem formulation, several techniques have been proposed to find a low-cost placement. *Iterative improvement* techniques find a low-cost placement by repeatedly applying a sequence of small changes called *moves*. *Partitioning-based* techniques recursively break a design into minimally connected groups of cells before assigning locations. *Analytical* placement techniques minimize a cost function directly by solving a system of equations.

The CADAPPLETS placement animations focus on iterative improvement techniques for cells of varying sizes. These techniques perform of a sequence of *move attempts* in which a simple transformation is applied to a randomly selected cell and the change in placement cost is calculated. Any move which decreases the cost is accepted, while any move that increases the cost is rejected and reversed. Each move attempt is one of the following randomly selected actions: 1) moving the cell a random amount horizontally or vertically; 2) reflecting the cell horizontally or vertically; or 3) rotating the cell 90 degrees.

The interesting events which occur during iterative improvement are the selection, evaluation, and acceptance or rejection of moves. The animations are designed to illustrate these events.

Fig. 2 shows a screen capture of the iterative improvement animation. It presents two views: on the left, a placement display that shows the current placement as a "rat's nest" diagram that highlights the current move being attempted, and on the right a move history display that illustrates the impact on placement cost of a sequence of move applications. A status bar at the top of each display shows statistics about the placement and the history of move attempts.

Moves are depicted on the placement display during animation by highlighting the selected cell and depicting the selected move as an arrow. Before the move is applied the cell is highlighted in orange. After the move the cell is highlighted green if accepted and red if rejected. A text string at the bottom of the display summarizes the status of each move attempt. For example, in Fig. 2 a move has been applied to the leftmost cell

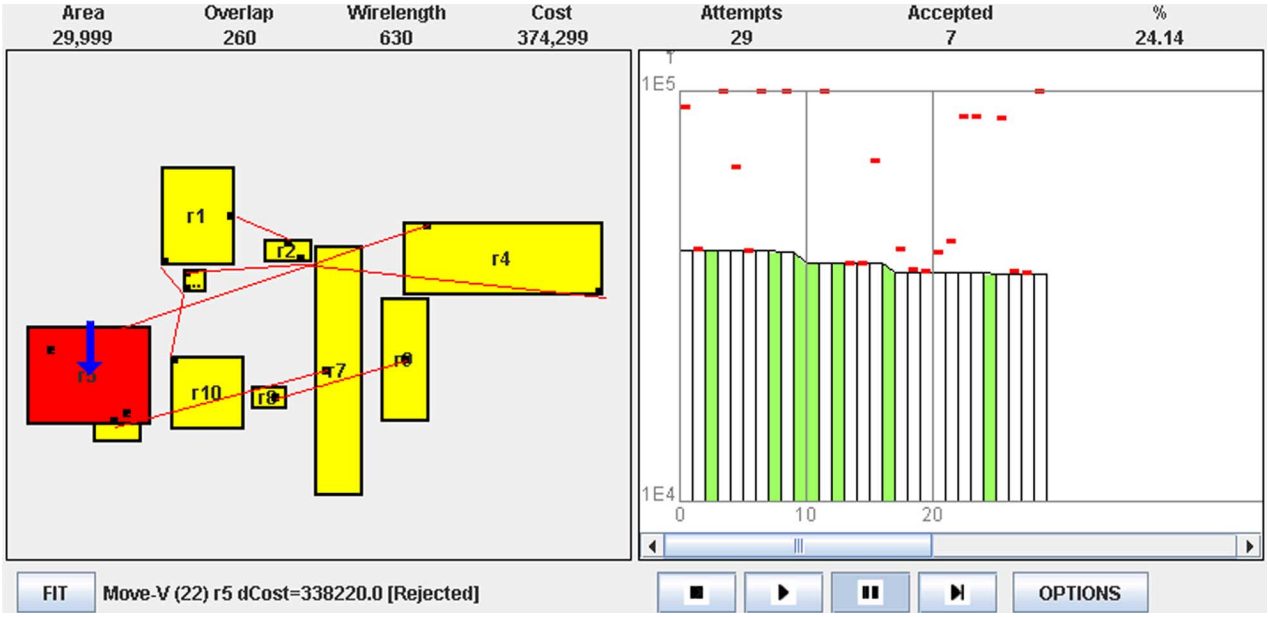

Fig. 2. Visualization of placement by iterative improvement.

that moved it downward. Since the result of this move is an increase in the cost function (due to increased overlap and wire lengths), the move is rejected.

The move history display shows the history of move attempts from the beginning of iterative improvement to the end. The status bar displays the number of move attempts and the number of accepted moves

Since iterative improvement accepts only "downhill" moves that decrease placement cost, it tends to become trapped in a local cost minimum. *Simulated annealing* [15], [16] is a popular optimization technique that addresses this drawback by allowing "uphill" moves to be accepted in a controlled, probabilistic fashion.

As in iterative improvement, simulated annealing repeatedly applies moves to different cells in a placement, and downhill moves are always accepted. However, uphill moves are accepted *probabilistically* under the control of a temperature parameter T. Specifically, a move that increases the cost function by ΔC is accepted with probability $P = e^{-(\Delta C/T)}$.

The algorithm operates in two nested loops. In the outer loop, the value of T is gradually lowered from a "high" initial value (that allows most uphill moves to be accepted) down to a final value (where almost no uphill moves are accepted). In the inner loop, a large number of moves are attempted at each temperature value. Each time a move is attempted, P is calculated and compared to a random number r $(0 < r < 1)$. If $P > r$, the move is accepted, otherwise it is rejected and reversed.

To animate simulated annealing, the placement display is used as one view of the algorithm's operation, while an extended move history display is used as a second view.

Fig. 3 shows the enhanced move history display used by the simulated annealing animation. This display plots the result of each move attempt as a cost plot as before, but here it is extended to show the acceptance of uphill moves, which are shaded in orange (in contrast to green for downhill moves). Rejected moves remain unshaded.

The move history display for simulated annealing also includes a display of how the acceptance decision was made for each move attempt; this appears immediately below the cost display. For each move, the cost probability is shaded to show the acceptance probability P. Downhill moves are shown with $P = 1$ and shaded green. Uphill moves are shown with the probability $P = e^{-(\Delta C/T)}$ and shaded in orange when the move is accepted or gray when rejected. The random number r that is used for the decision is plotted as a small diamond shape – when it lies within the shaded region of the move probability (i.e., $P > r$) this indicates an acceptance decision.

A second view is used to explore how placement cost varies as a function of temperature – the classic "annealing curve" [16]. This cooling schedule display (Fig. 4) plots the minimum, maximum, and average cost placement accepted at each temperature.

While simulated annealing is conceptually simple, there are a number of issues that make creating an effective implementation difficult [16], including the determination of initial temperature, number of move attempts per temperature, stopping criteria, and selection of different types of moves. The "options" control button brings up a popup menu (not shown) that allows the user to experiment with several of these parameters, and also to adjust the animation speed.

The simulated annealing animation operates in either a fine-grain or coarse-grain mode that can be selected from the options menu. The fine-grain mode updates the display after each move attempt and is used to illustrate the consideration of individual moves. The coarse-grain mode updates the display after each temperature change and is used to illustrate how the placement evolves over the course of the annealing process. In both modes, the animation can be paused, single-stepped, or restarted using "VCR control" buttons.

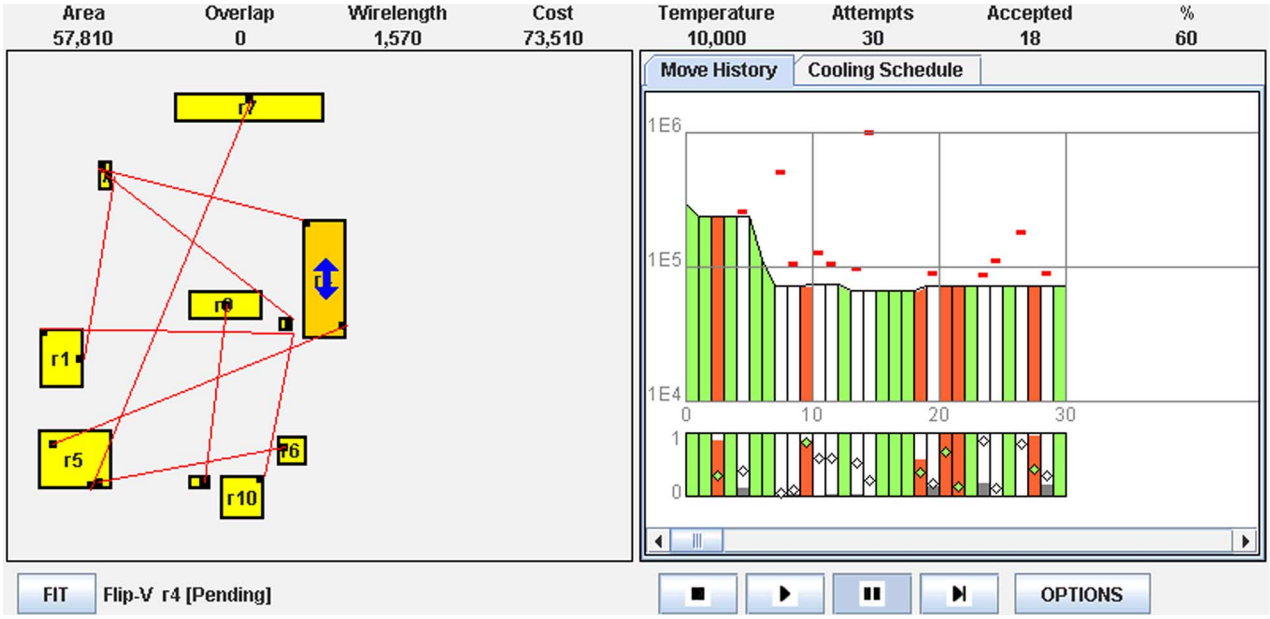

Fig. 3. Simulated annealing – move history display.

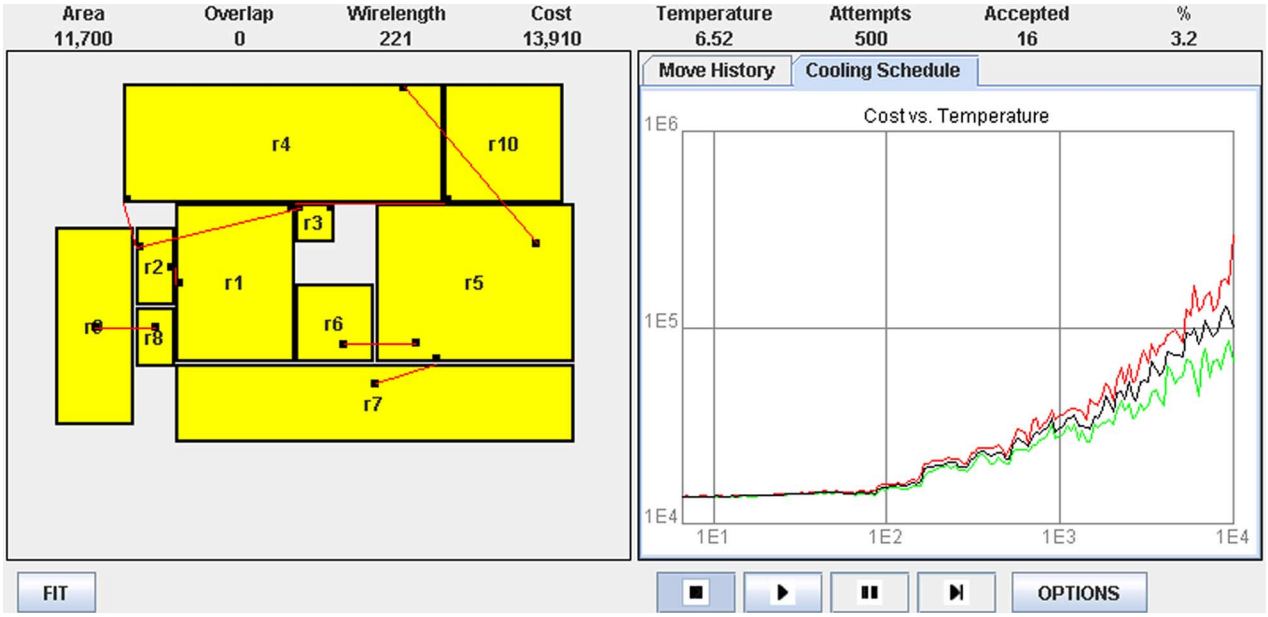

Fig. 4. Simulated annealing – cooling schedule display.

B. Routing

The goal of routing is to find connections for nets between the terminals of cells. In most integrated circuits these connections are constrained to be rectilinear. There are several approaches to routing, including maze routing, channel routing, and techniques based on spanning trees and Steiner trees [1].

Maze routing models the routing surface as a grid. Each grid point can be a terminal of a desired connection (known as either the *source* or *target*), a wire that connects adjacent grid points, or an *obstacle* that represents space that is not available for interconnections. The grid is described by a two dimensional array which records the state of each grid point

The Lee algorithm [17] for maze routing is popular because it is guaranteed to find a shortest-path connection if one exists. This algorithm operates in three phases. During the *expansion* phase, the algorithm searches outward from the source terminal while labeling each node with its distance from the source. When the target is reached, the *backtrace* phase selects a path by following decreasing label values and marks these as wires (which act as obstacles for later routings). The *cleanup* phase erases unused expansion labels.

Fig. 5 shows the CADAPPLETS animation of the Lee algorithm, which displays the routing grid. The grid display is updated when the following interesting events occur: 1) the selection of source and target terminals (labeled "S" and "T") of a desired connection by clicking on the grid; 2) labeling of individual gridpoints during the expansion phase (shown in Fig. 5); 3) marking of gridpoints as a connection (and obstacle for later

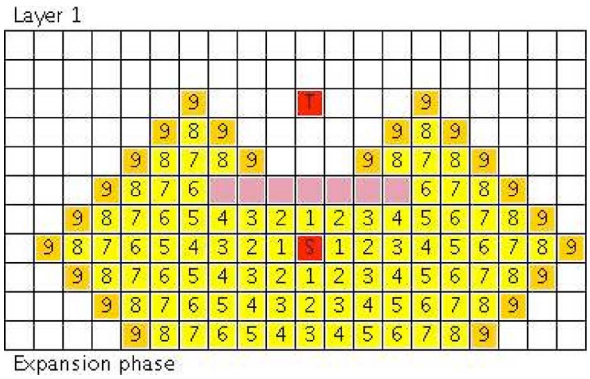

Fig. 5. Lee algorithm maze routing animation (expansion phase). Gridpoints are labeled in increasing order of distance from the source (S) until the target (T) is found.

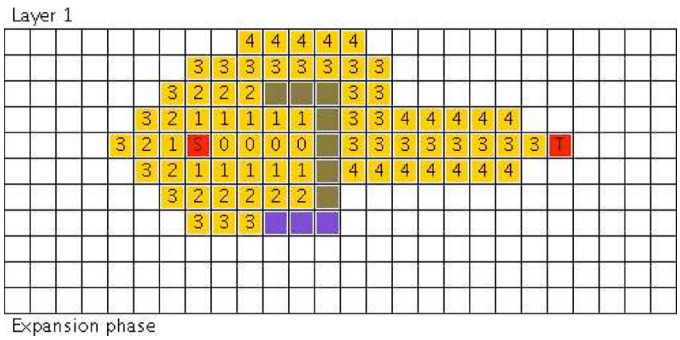

Fig. 6. Hadlock's algorithm maze routing animation.

searches) during the backtrace phase; and 4) removal of all unused labels during the cleanup phase.

Other maze routing algorithms reduce execution time by biasing the search in the direction of the target. For example, the A* algorithm [18] labels each gridpoint with the sum of two values: the distance from the source (the same value used by the Lee algorithm), and a lower-bound estimate of the distance from the labeled gridpoint to the target. Hadlock's algorithm [2] labels each gridpoint with a *detour number* that reflects the number of distance units a gridpoint "detours" away from the minimum-length path. Animations have been developed for each of these algorithms.

Fig. 6 shows the animation of Hadlock's algorithm. When no obstacle is present, expansion is biased in the direction of the target. When the search encounters an obstacle, expansion proceeds away from the path only as long as the search is blocked by an obstacle.

A major drawback of maze routing is its computational complexity. The *channel routing* problem formulation reduces this complexity by constraining the locations of connection terminals and wires. Specifically, connection terminals must be placed in columns at the top and bottom of the routing region (channel), and two layers of wiring are used for horizontal and vertical connections. Horizontal connections between terminals are assigned to *tracks* in the channel. Vertical connections occupy columns on a separate layer, so that they pass over unconnected layers in tracks, while *vias* connect the horizontal and vertical connections of connected nets.

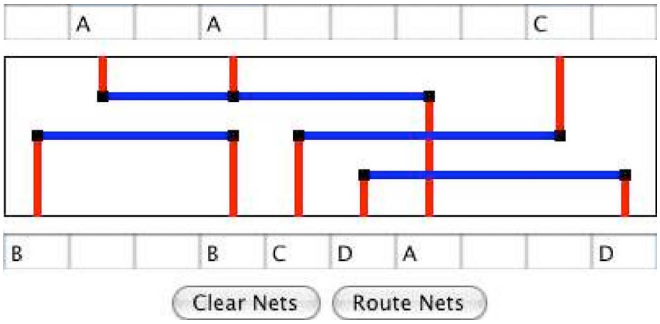

Fig. 7. Left-edge algorithm channel routing animation. Note the vertical constraint that requires that net A be assigned to a track above net B.

The *left edge algorithm* (LEA) [19] uses this constrained formulation to find connections by sorting nets by their left edge and then assigning each net in that order to the first track which contains no overlapping nets. The algorithm is complicated by the possible presence of *vertical constraints* – if different terminals are placed at the top and bottom of the same column, the net connected to the top terminal must be routed above the net connected to the bottom terminal.

Fig. 7 shows the CADAPPLETS animation of the left edge algorithm, which focuses on displaying the problem formulation – terminals are entered by the user at the top and bottom of the channel. The interesting events in the LEA are the assignment of each net to a horizontal track along with vertical connections to the terminals and the recognition of *cyclic* vertical constraints that conflict and cannot be routed. These events are displayed for all nets when user clicks the "route nets" button.

Several modern routing algorithms are based on the concepts of the *rectilinear minimum spanning tree* (RMST) and *Steiner rectilinear tree* (SRT) [1]. An RMST is a graph in which the nodes are connected so that the total edge distance is minimum. An RMST can be constructed by Prim's algorithm [20], which builds up a partial tree starting with a single node and adding the closest node in each iteration until all nodes are connected. Fig. 8 shows an animation of Prim's algorithm. The graphical representation shows the problem formulation by plotting location of the points and the edges in the partial tree. The animation sequences through each interesting event of adding the closest node to the partial tree. Control buttons allow the animation to be paused, single-stepped, and restarted.

A minimum rectilinear Steiner tree [21] connects nodes with edges and additional points known as Steiner points such that the total edge distance is minimum. It has been proven that a minimum Steiner tree can be constructed with all Steiner points lying on a *Hanan grid* where grid lines correspond to the x and y coordinates of the nodes in the tree. It has also been proven that a minimum Steiner tree can be up to $1.5\times$ shorter than the corresponding spanning tree. Fig. 9 shows an animation that demonstrates the concept of the Steiner tree and Hanan grid. This animation displays the problem formulation of the net terminals along with added Steiner points; a user interacts with this display directly to add and remove Steiner points and evaluate the cost of the resulting tree.

The minimal RST problem is NP-hard [1]. For this reason, heuristics that produce good but not necessarily minimum Steiner trees are popular. Fig. 10 shows an animation of the

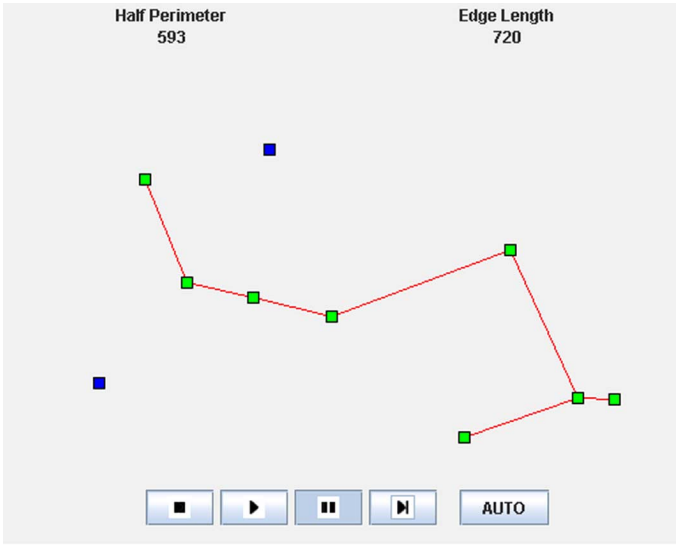

Fig. 8. RMST animation.

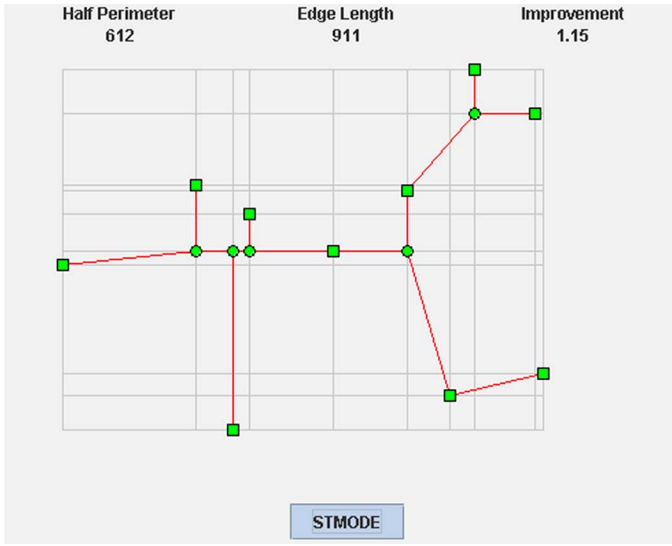

Fig. 9. SRT demonstration.

Borah–Owens–Irwin (BOI) heuristic [22], which examines the effect of connecting a node in the graph to the bounding box of an edge (implicitly inserting a Steiner point at the connection) while removing the longest edge in the resulting cycle. In some cases, this modification reduces the edge length of the resulting tree, which is desirable. The BOI heuristic evaluates the potential gain of all node/edge combinations and ranks them in decreasing order before applying them in a greedy fashion. This animation shows the interesting events of enumerating the possible edge/node combination followed by the application the combinations with the highest gain. Control buttons allow the animation to be paused, single-stepped, and restarted.

IV. Implementation and Evaluation

Each of the CAD algorithm animations is implemented as a Java applet. The maze routing and channel routing applets were developed first, using the original Java abstract windowing toolkit (AWT). The others were developed later and using the more capable Swing user interface library [23]. The applets range in size from 500 lines of code for the channel router applet to 4500 lines for the iterative improvement and simulated annealing placement applets, which are implemented together. Compiled code is relatively small, ranging from about 10–80 Kbytes in a Java archive (JAR) file, allowing the applets to be downloaded quickly.

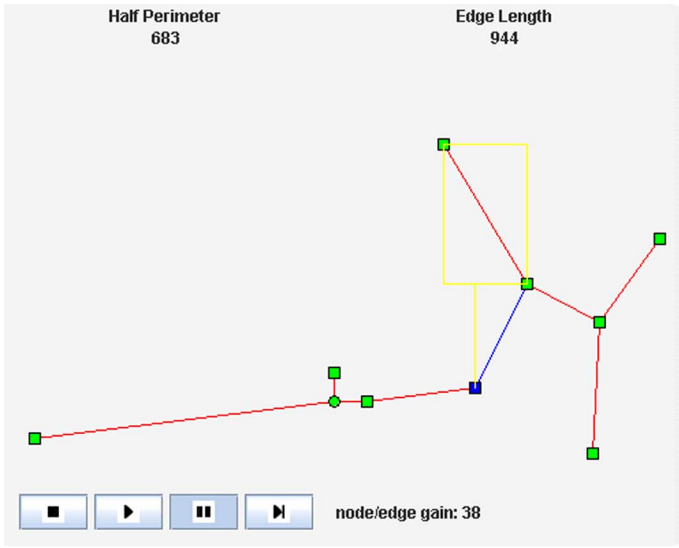

Fig. 10. Borah–Owens–Irwin SRT heuristic animation.

The applets have been used as a teaching aid during lectures in a VLSI design course at Lafayette College that is similar to the "typical" course described in Section I. The goal of these presentations is to provide students with a qualitative understanding of placement and routing algorithms while relating the function of these CAD tools to the tasks involved in manual chip layout.

For example, the maze and channel routing animations are first presented during a lecture on cell layout. Similarly, placement animations are presented during a lecture on ASIC design. A final lecture at the end of the course provides a complete overview of CAD systems. This lecture reviews the routing and placement algorithms and presents the spanning tree and Steiner tree animations for the first time. Student response to these demonstrations has been positive and has been noted in teaching evaluation comments.

To assess student response more fully, a survey was collected during the spring 2007 offering of the course. This survey listed a set of eight specific teaching objectives and asked students to rate their confidence that each objective was met on a scale of 1 (not confident) to 5 (very confident) and the contribution of the animation to achieving this objective. Table I summarizes the results of this survey, which show that students react positively to the animations and believe that the animations help them learn CAD topics.

To assess the level of use of the applets by other institutions and individuals, Web server logs were analyzed to count accesses during calendar years 2005 and 2006, excluding search engine references. Table II summarizes accesses to the CADAPPLETS main page and each individual animation. Visitors include users at universities, CAD tool companies, and

TABLE I
STUDENT RESPONSES – CAD OBJECTIVES SURVEY

Objective – Students understand:	Objective Achieved	Contrib. of Animations
1. General role of placement	5.0	5.0
2. "Half-perimeter" wire length estimate	5.0	5.0
3. Min. Spanning tree wire length estimate	5.0	5/0
4. Steiner tree wire length estimate	4.75	4.67
5. Placement using Simulated Annealing	5.0	5.0
6. General role of routing	5.0	5.0
7. Maze routing / Lee Algorithm	4.5	4.67
8. Channel routing / Left-Edge Algorithm	4.5	4.67

TABLE II
APPLET ACCESS SUMMARY

Accessed Item	2005	2006	Change
CADAPPLETS Homepage	1808	2022	11.8%
Routing Applets			
Left Edge Algorithm	2959	3041	2.7%
Lee Algorithm	1456	1554	6.7%
RMST	N/A	492	N/A
Steiner Demo	N/A	571	N/A
Steiner BOI	N/A	172	N/A
Placement Applets			
Problem Formulation	715	710	-0.7%
Iterative Improvement	345	433	25.5%
Simulated Annealing	1497	1598	6.7%
Total Applet Accesses	6972	8571	22.9%

chip design companies as well as individuals using Internet service providers. While the bulk of these accesses were from North America and Europe, there were also significant accesses from South America, Asia, Africa, Australia, and New Zealand. Overall, accesses increased 22% between 2005 and 2006, showing growing interest in the animations.

Finally, to assess how the animations are used directly by instructors at other colleges and universities, a Google search was used to find lecture notes and reference pages that include links to one or more of the applets and have been posted on the Web. Results of this search indicate such references at 15 institutions in Europe (8), North America (4), Asia (2), and New Zealand (1).

V. CONCLUSION

This paper has discussed the use of software visualization techniques to develop animations of common VLSI CAD algorithms. The applets have been used successfully as an aid to instruction in VLSI design and CAD algorithms courses and have been downloaded extensively via the Internet for individual study.

There are a number of areas for future work in this project. First of all, the animations developed so far provide only a subset of the algorithms used in VLSI CAD – many more are needed for thorough coverage. Existing animations should be extended to include considerations such as congestion and timing. Animations should be applied to larger and more realistic examples. Finally, the use of the animations in course work should be expanded so that students learn more through direct interaction.

REFERENCES

[1] M. Saraffzadeh and C. Wong, *An Introduction to VLSI Physical Design*. New York: McGraw-Hill, 1996.
[2] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 3rd ed. Norwell, MA: Kluwer, 1999.
[3] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. Reading, MA: Addison-Wesley, 2004.
[4] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2003.
[5] W. Wolf, *Modern VLSI Design: System on Chip Design*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2002.
[6] J. Nestor, "Web-based visualization tools for teaching VLSI CAD algorithms," in *Proc. Int. Conf. Microelectronic Systems Education*, Las Vegas, NV, 2001, pp. 100–101.
[7] J. Nestor, "Animation of VLSI CAD algorithms – A case study," in *Proc. American Society for Engineering Education Annu. Conf.*, Montreal, QC, Canada, Jun. 2002, CD-ROM.
[8] M. Brown and R. Sedgewick, "Techniques for algorithm animation," *IEEE Softw.*, vol. 2, no. 1, pp. 28–39, Jan. 1985.
[9] J. Stasko, J. Domingue, M. Brown, and B. Price, Eds., *Software Visualization: Programming as a Multimedia Experience.* Cambridge, MA, MIT Press, 1998.
[10] K. Arnold, J. Gosling, and D. Holmes, *The Java Programming Language*, 3rd ed. Reading, MA: Addison-Wesley, 2000.
[11] M. Brown, R. Raisamo, and M. Najork, "A Java-based implementation of collaborative active textbooks," in *Proc. IEEE Symp. Visual Languages*, Capri, Italy, 1997, pp. 372–379.
[12] R. Hentschke and R. Reis, Blue Macaw Didactic Placement Tool, 2007 [Online]. Available: http://www.inf.ufrgs.br/~renato/bluemacaw
[13] S. Szollar and J. Young, The Incredible Anneal-O-Matic, 2007 [Online]. Available: http://www-cad.eecs.berkeley.edu/~jimy/classes/ee244/hw2/index.htm
[14] S. Hauck, "APHYDS: The academic physical design skeleton," in *Proc. Int. Conf. Microelectronics Systems Education*, Anaheim, CA, 2003, pp. 8–9.
[15] S. Kirkpatrick *et al.*, "Optimization by simulated annealing," *Science*, vol. 220, no. 2298, pp. 671–680, 1983.
[16] R. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits Devices Mag.*, vol. 5, no. 1, pp. 19–26, Jan. 1989.
[17] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Comput.*, vol. EC-10, pp. 346–365, 1961.
[18] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. Comput.*, vol. C-23, no. 9, pp. 907–914, Sep. 1974.
[19] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proc. 8th Design Automation Conf.*, Atlantic City, NJ, 1971, pp. 165–173.
[20] R. C. Prim, "Shortest connecting networks and some generalizations," *Bell Syst. Tech. J.*, vol. 31, pp. 1398–1401, Nov. 1957.
[21] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, Mar. 1966.
[22] M. Borah, R. Owens, and M. J. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Trans. Comput.- Aided Design Integr. Circuits Syst.*, vol. 13, no. 12, pp. 1563–1568, Dec. 1994.
[23] K. Walrath and M. Campione, *The JFC Swing Tutorial*. Reading, MA: Addison-Wesley, 1999.

John A. Nestor (S'78–M'87–SM'91) received the B.E.E. degree from the Georgia Institute of Technology, Atlanta, and the M.S.E.E. and Ph.D. degrees from Carnegie Mellon University, Pittsburgh, PA, in 1979, 1981, and 1987, respectively.

In 1987, he joined the Illinois Institute of Technology, Chicago, where he was Assistant Professor from 1987 to 1993 and Associate Professor from 1993 to 2000. In 2000, he joined the Electrical and Computer Engineering Department, Lafayette College, Easton, PA, where he is currently an Associate Professor. His research interests include VLSI CAD, VLSI design, FPGA-based design, reconfigurable systems, and visualization aids for electrical and computer engineering education.

Dr. Nestor received a Best Paper award at the 22nd International Workshop on Microprogramming and Microarchitecture in 1989, a NSF Research Initiation Award in 1990, and the John A. Curtis Lecture Award from the Computers in Education Division of the American Society of Engineering Education in 2002. He has served as a member of the Program Committee for the International Conference on Microelectronic Systems Education in 2003, 2005, and 2007 and as Poster Chair at the same conference in 2007.