Animation of VLSI CAD Algorithms – A Case Study

John A. Nestor
Department of Electrical and Computer Engineering
Lafayette College

Abstract

The design of modern VLSI chips requires the extensive use of Computer-Aided Design (CAD) tools. Undergraduate VLSI Design courses typically teach the use of these tools to create designs, but provide little or no information about how the tools work, which makes it difficult to use them effectively. The goal of the CADAPPLETS project is to provide a set of Java animations which will aid students in visualizing the internal operation of these tools in terms of common problem formulations, classes of algorithms, and specific algorithms. This paper describes one set of animations that illustrates the operation of *placement* tools, which assign cells to physical locations in a layout. This and other animations are used in class presentations in Lafayette College's undergraduate VLSI courses and are available at http://foghorn.cadlab.lafayette.edu/cadapplets/.

1. Introduction

Modern VLSI chips are being designed at seemingly ever-increasing levels of complexity. Current chip designs commonly contain tens of millions of transistors, and larger chips are on the horizon. The design of such complex chips would be impossible without a wide range of Computer-Aided Design (CAD) tools. These tools automate different parts of the design process, making it possible to complete a large design in a reasonable amount of time while managing an enormous amount of detail.

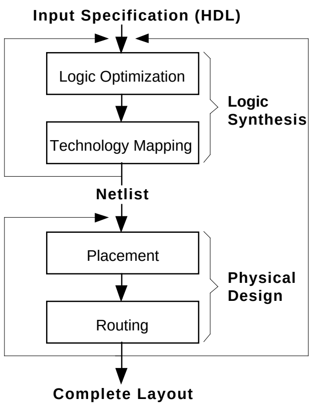


Figure 1 – Simplified Design Flow for Semi-Custom Chips

Figure 1 shows a simplified diagram of CAD tools used in the design of semi-custom VLSI chips – a popular style in which a pre-designed library of *standard cells* is used to speed the design process. Input is in the form of *a Hardware Description Language*

(HDL), which specifies the desired function of the chip. Producing a complete chip layout from this input is a complex task, so the design problem is usually broken into distinct tasks performed by separate CAD tools. *Logic Synthesis* tools translate this input into hardware, optimize it to reduce its area and meet timing constraints, and map the design into a *netlist* that specifies a set of standard cells and connections (nets) that will implement the desired function. Next, *Physical Design* tools assemble the cells and connections into a layout that is suitable for fabrication as an integrated circuit. This process is typically done 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 on the chip.

Designers working with these tools must control them properly in order to trade off constraints on area, timing, and power consumption. Because chip timing depends on the structure of the chip and its interconnections, exact timing information is not known until placement and routing tools have been run. At this point, timing information can be extracted from the layout and checked against timing constraints. Constraint violations often require that the design be modified and the tools re-run, making design an *iterative* process. Completing a design in a reasonable number of iterations requires insight into how the CAD tools operate: the *problem formulation* and *cost function* used by each tool to estimate the quality of its output, and the type of *algorithms* used to generate a solution.

VLSI Design is typically taught as a senior-level course in undergraduate Electrical and Computer Engineering programs. A typical course covers a wide range of topics, starting with integrated circuit processing, device physics, transistor characteristics, layout rules, common circuit structures, timing and power dissipation, and the use of CAD tools to produce chip designs. Given the broad coverage of material in such a course, it is difficult to adequately cover the details of how each CAD tool operates. Instead, CAD tools are usually treated as “black boxes” which automatically generate a design, with little or no explanation of the capabilities and limitations of each tool. Courses in CAD Algorithms[1,2] can provide this expertise, but may not be available to all students and are often offered at the graduate level.

The goal of the CADAPPLETS Project[3] at Lafayette College is to address this problem by providing visualization tools that can used to quickly present information about common CAD tools and algorithms to students in undergraduate VLSI design courses. This makes it easier to rapidly give the students a qualitative understanding of how these tools work, while at the same time providing a vehicle for more detailed study.

Techniques drawn from *software visualization* and *algorithm animation*[4,5] are used in developing these animations, which are implemented as Java applets. Each applet may be run as a classroom demonstration by an instructor or interactively by individual students. Moreover, the applets can be embedded into web pages of explanatory text, forming an “active textbook” with animated figures.

Each animation is intended to provide students with an understanding of CAD algorithms at three levels: (1) the general problem formulation, illustrating how the problem is represented, the cost functions it uses, and any imprecision or drawbacks; (2) different classes of algorithms used to find solutions for a given problem formulation; and (3) the details of specific algorithms. At this writing, animations have been developed to support physical design algorithms for placement and routing. The resulting applets can be used in undergraduate VLSI Design courses to provide students with a qualitative understanding of the problem formulation and the algorithms used in these tools. It can also be used in advanced courses that study these algorithms in more depth.

This paper presents a case study that explores the CADAPPLETS approach as it is applied to the *placement* problem. Specifically we use a set of applets to visualize the problem formulation for placement, the class of *iterative improvement* algorithms (one set of algorithms that can be used to solve the placement problem), and a detailed animation of a specific iterative-improvement algorithm known as *simulated annealing*[6,7].

This paper is organized as follows. Section 2 reviews the ideas of software visualization that are used to produce animations and surveys other efforts to animate CAD algorithms. Section 3 reviews the placement problem and discusses how the problem formulation for placement is visualized. Section 4 discusses how the class of iterative improvement algorithms can be applied to placement and how the basic transformations (moves) that are used by this type can be visualized. Section 5 discusses the detailed visualization of the simulated annealing algorithm in particular. Section 6 provides conclusions and suggestions for future work.

2. Background

Algorithm animation and software visualization have been used in teaching software algorithms for a number of years. In the mid-1980's Brown and Sedgewick developed the BALSA system[4] as a framework for software algorithm animation and used this framework in courses at Brown University. More recently, many other systems have been developed that extend and improve upon this work; a good overview is provided in[5]. BALSA and similar systems animate software algorithms by presenting *views* - graphical renderings of data structures and execution history. As an animation proceeds, the position, shape, and color of objects in the 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 graphical views are tied to the algorithm itself using the *interesting events paradigm*, in which algorithm code is annotated to identify important steps in the algorithm. As these events are reached during execution, the graphical views are updated accordingly. The software visualization community has used these techniques to animate a wide range of software algorithms. However, relatively few animations of VLSI CAD algorithms have been developed. For example, the N_ABLE project[8] presents applets that illustrate concepts in switching and automata theory and logic synthesis. A few physical design animations have also been developed as part of student projects in VLSI CAD courses[9,10].

Visualization aids are also built in to commercial CAD tools. For example, placement tools often display a placement using a *rat's nest* diagram[2], in which nets are displayed as line segments laid over a diagram of cell placements. This 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 visualization aids focus more on the end result than the operation of the underlying algorithm.

In contrast, the visualization aids in the CADAPPLETS project are intended to provide insight about the operation of the algorithms as well as the end result. Displays like the rat's nest diagram are used in these visualizations, but are augmented to illustrate the dynamic behavior.

3. Visualizing the Placement/Floorplanning Problem Formulation

The goal of placement is to assign physical locations and orientations to a set of interconnected cells on a chip layout while minimizing a cost function that measures chip area and routing quality. The input to placement is a netlist that specifies the dimensions of each cell used, the locations of the terminals on each cell, and a list of the nets (wires) that connect the terminals of each module. 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 if that results in a better placement, but are not generally allowed to overlap.

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 some combination of chip area and routing quality. Chip area is easily measured by finding a bounding box that contains all the cells. Routing quality, on the other hand, is more difficult to measure. The ultimate measure of routing quality is the length of the routed nets, but it is not practical to run a routing tool each time that a placement is changed. Instead, *estimates* of routing quality are used that predict the *approximate* length of routed nets as well as the *congestion* of the areas that will be used for routing.

Common net length estimates include the length of a minimum spanning tree that connects the terminals of the nets, the length of a Steiner tree that connects the terminals, or one-half the perimeter of a bounding box that contains the terminals of the nets. A common congestion measurement is the count of the number of nets that cross a set of cut lines on the layout surface. The cost function often also includes a measure of cell overlap as a *penalty function*. Although cell overlaps are not allowed in the final placement, this approach is used in placement methods that may consider illegal placements as intermediate states before reaching a final configuration. These measures of area, congestion, wirelength, and overlap penalty are combined in a weighted sum to create the complete cost function.

To allow students to visualize this problem formulation, a view has been developed that displays a placement to the user as a rat's nest diagram, as shown in Figure 2. The main body of this *placement view* displays cells as yellow rectangles, terminals as small black rectangles, and nets as red lines between the terminals. A status bar at the top of the display shows area, overlap, estimated wirelength (currently using the half-perimeter estimate), and overall cost, which is a weighted sum of these terms.

The placement view also allows the placement display to be manipulated by the user to explore the impact of changes - each cell can be rotated or mirrored by clicking the mouse over the cell, and each cell can moved to a different position by clicking and dragging. The display is updated immediately as these operations occur. The placement view is implemented as a set of Java classes that can read a netlist from an input file or web connection. The following sections describe how this code is extended to illustrate the operation of iterative improvement algorithms and simulated annealing.

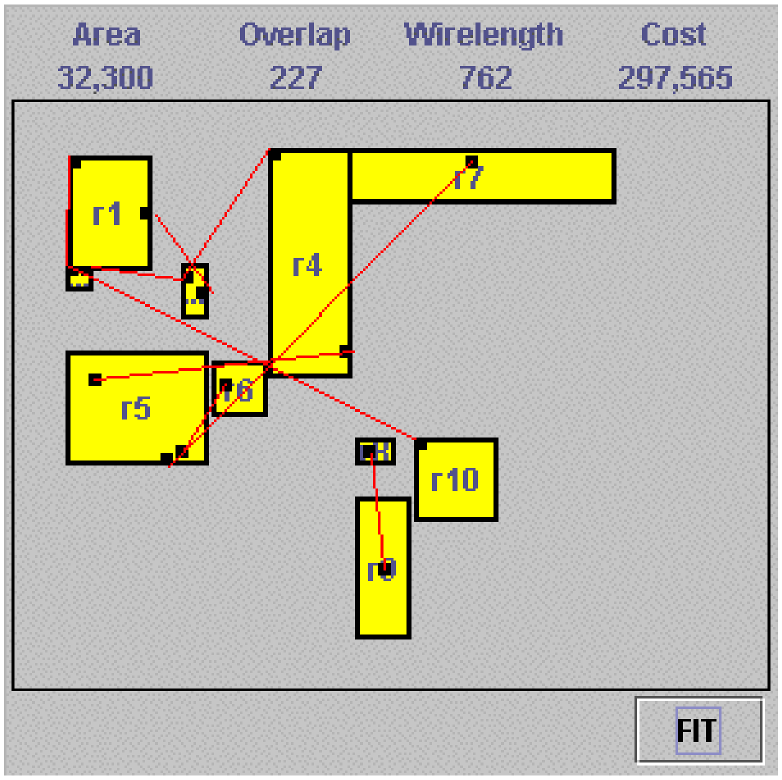


Figure 2 – Placement Display

4. Visualizing Iterative Improvement Placement Algorithms

Given the formulation of the placement problem described in the previous section, there are a number of common classes of algorithms that are used to search for low-cost placements. These include *iterative improvement* techniques, which attempt to find a low-cost placement by making a sequence of small changes, *partitioning-based* techniques, which recursively break the a design into minimally-connected groups of cells, and *analytical* techniques which attempt to minimize a cost function (e.g., the square of the wire length) directly by solving a system of equations. This paper focuses on visualizing iterative improvement algorithms.

Iterative improvement algorithms search for a good placement configuration by applying a set of simple *moves* – transformations that alter the placement in some way. While the details of iterative improvement algorithms vary, the key idea is to repeatedly apply randomly selected moves, while evaluating the impact of each move of the cost function. "Downhill" moves that decrease the cost function are *accepted*, since they improve placement quality. On the other hand, "uphill" moves that increase the cost function are usually *rejected* and reversed, since they reduce the quality of the placement. Table 1 shows a typical set of moves that might be used with the problem formulation described in the previous section.

Move	**Effect**
Move-H	Translate horizontally by a random amount
Move-V	Translate vertically by a random amount
Rotate	Rotate cell 90 degrees
Flip-H	Flip cell horizontally
Flip-V	Flip cell vertically

Table 1 – Typical Moves for Iterative Improvement

In visualizing iterative improvement algorithms, a key goal is to demonstrate how a placement changes as moves are applied. This is accomplished by extending the placement view described in the previous section to display the "interesting events" of iterative improvement: *selection* of a cell and move, *application* of the move, and either *acceptance* or *rejection* of the move. Each of these events is displayed using highlighting of cells drawn in the placement view, overlaid arrow graphics that indicate the direction and distance of the move, and text which displays the name of the move and, after it is applied, its impact on the cost function and whether it is accepted or rejected.

Figure 3 shows how these events are displayed in sequence for a "MoveH" move (that translates a cell horizontally). Before the move is applied, the cell selected for the move is highlighted in orange, and an arrow is drawn originating from the center of the cell and ending at the point where the cell will be moved. At the same time, the name of the move and the amount of the offset are displayed in text at the bottom of the view. After the move is applied, the cell is shown in its new position at the end of the arrow, and text at the bottom of the view again indicates the name of the move and the offset, as well as the change in the cost function (dCost) and an indication of whether the move was accepted or rejected. If the move is accepted, the selected cell is highlighted in green, while if the move is rejected, it is highlighted in red. Rotation and mirroring moves are displayed in a similar fashion.

With these extensions, the placement view allows animation of iterative improvement algorithms through a sequence of these displays. The arrows and highlighting are most useful for higher-speed animation, since they allow the moves to be comprehended at a glance. The text part of the display is most useful for slower animation and single-stepping, where each move is studied carefully.

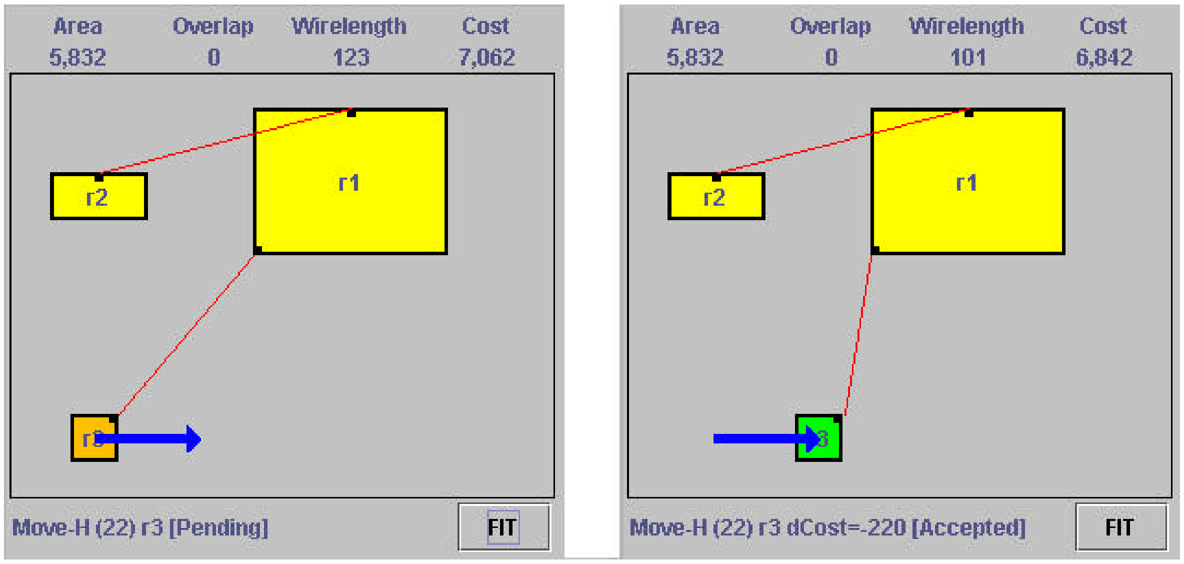


Figure 3 – Visualizing Horizontal Move – Before and After Application

5. Visualizing Placement with Simulated Annealing

Straightfoward iterative improvement algorithms accept only downhill moves. While simple to implement, the drawback of this approach is that the algorithm may become "trapped" in a local minimum in the cost function. The simulated annealing algorithm[6,7] is a popular approach to optimization that accepts uphill moves in a controlled fashion in an attempt to avoid this problem. This algorithm has been used very successfully in placement, and for a number of other CAD tools as well.

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

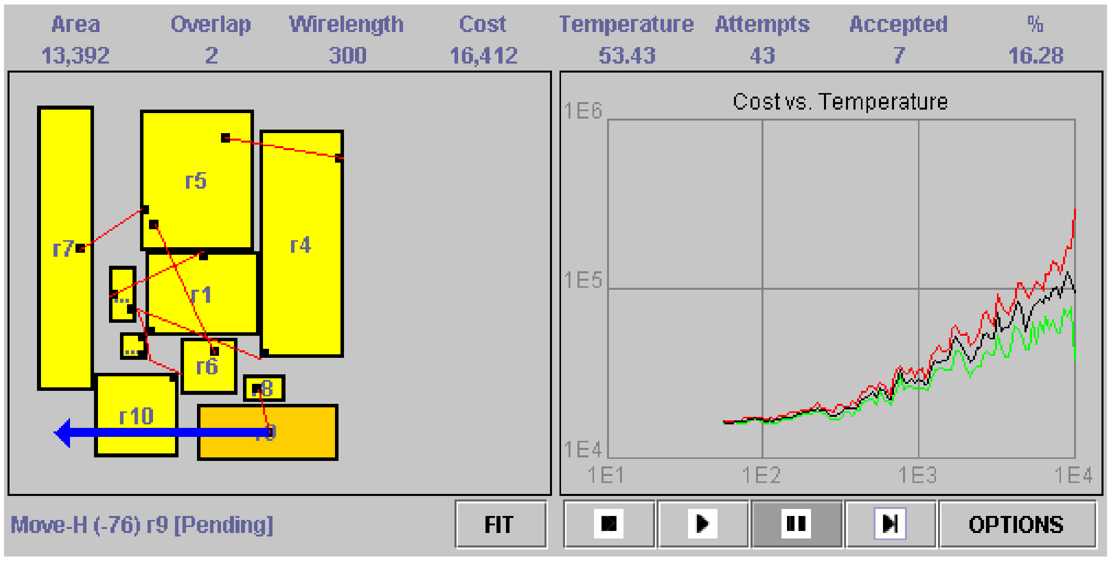
The algorithm operates in two nested loops. In the outer loop, the value of T is gradually lowered from an initial value which allows most uphill moves to be accepted down to a final value where no uphill moves are accepted. In the inner loop, a large number of moves are attempted at each temperature value. Each time a move is attempted, the change in the cost function is used to calculate the acceptance probability P. P is compared to a random number r generated between 0 and 1. If $P > r$, the move is accepted, otherwise it is rejected and reversed.

To animate the Simulated Annealing algorithm, the placement view described in the previous sections is used as one view of the algorithm's operation, as shown in Figure 5.

A second view is used to explore how placement cost varies as a function of temperature – the classic "annealing curve"[6]. This display plots the minimum, maximum, and average cost placement accepted at each temperature versus temperature and scales the display as necessary as annealing proceeds.

While simulated annealing is conceptually simple, there are a number of issues that make the creation of an effective implementation difficult[7], including the determination of the initial temperature, number of move attempts per temperature, stopping criteria, and weighting 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 speed of the animation.

The Simulated Annealing animation allows for animation in either a fine-grain or coarse-grain mode. In the fine-grain mode, each move attempt is displayed following the conventions described in the previous section. This gives the user a good idea of how individual moves are attempted, accepted, and rejected at each temperature. However, since even small placement problems require hundreds or thousands of move attempts at each temperature, this mode is not appropriate for visualizing the overall operation of the algorithm. Thus the coarse-grain mode updates the display once at the end of each temperature, showing the placement and cost information and updating the temperature curve. This allows the user to visualize the change in placement at each temperature. This display mode is also set using the options control panel.



Area	Overlap	Wirelength	Cost	Temperature	Attempts	Accepted	%
13,392	2	300	16,412	53.43	43	7	16.28

Move-H (-76) r9 [Pending]

Figure 5 – Simulated Annealing Placement Applet

6. Conclusions

This paper has discussed the use of animation techniques to illustrate the operation of the VLSI placement problem as part of the CADAPPLETS project. This animation illustrates the placement problem formulation and cost function, the operation of iterative improvement algorithms, and the specific use of simulated annealing to solve the placement problem. It is currently being used in VLSI Design lectures at Lafayette and can can be viewed on the web at: http://foghorn.cadlab.lafayette.edu/cadapplets/.

There are a number of areas for future work in this project. For example, it should be possible to select different wirelength estimates during animation, and congestion metrics should included in both the cost function and the placement display. In addition there should be an option to support row-based placement of fixed-height cells, since this approach is used for standard-cell designs. Some display of the timing effects should also be added, as this is a major concern in modern placement tools. Finally, additional animations should be developed for other algorithms and other CAD tools.

References

1. Saraffzadeh, M., and Wong, C., *An Introduction to VLSI Physical Design*, McGraw-Hill, 1996.

2. Sherwani, N., *Algorithms for VLSI Physical Design Automation*, 3rd. ed., Kluwer Academic Publishers, 1999.

3. Nestor, J., "Web-Based Visualization Tools for Teaching VLSI CAD Algorithms", *Proceedings International Conference on Microelectronic Systems Education*, pp. 100-101, June 2001.

4. Brown, M. and Sedgewick, R., “Techniques for Algorithm Animation”, *IEEE Software*, Vol. 2, No. 1, January 1985, pp. 28-39.

5. Stasko, J., Domingue, J., Brown, M., and Price, B., ed., *Software Visualization: Programming as a Multimedia Experience.* MIT Press, 1998.

6. S. Kirkpatrick et. al., “Optimization by Simulated Annealing, *Science*, Vol 220, No. 2298, pp. 671-680, 1983.

7. Rutenbar, R. “Simulated Annealing Algorithms: An Overview”, *IEEE Circuits and Devices Magazine*, pp. 19-26, January 1989.

8. Moon, I., “Action Based Learning for Switching and Automata Theory”, http://vlsi.colorado.edu/~mooni/N_ABLE/N_ABLE.html.

9. Rutenbar, R., “18-760 Projects”, http://www.ece.cmu.edu/~ee760/760projects.html.

10. Szollar, S and Young, J., “The Incredible Anneal-O-Matic”, http://www-cad.eecs.berkeley.edu/~jimy/classes/ee244/hw2/index.htm, November 1996.

Biographical Information

JOHN A. NESTOR
is an Associate Professor in the ECE Department at Lafayette College in Easton, Pennsylvania. Prior to joining Lafayette in 2000, he an Associate Professor and Associate Chair of Computer Engineering with Illinois Institute of Technology in Chicago. His teaching and research interests include VLSI Design, Computer Engineering, and Computer-Aided Design for VLSI.