Visualization in a VLSI design automation system

by D. L. DeMaris

Problems unique to the visualization of complex, partially automated design tasks such as VLSI system design are reviewed, and approaches are described. The design domain used to illustrate the approaches is chip-level "floor-planning," an iterative-refinement design methodology for VLSI layout, routing, and timing control. The general view structure and control structure are described. Other visualization topics addressed are display of evolving data, sequencing of overlay data, an interleaved temperature–color metaphor for view consistency and clarity, and dynamically generated iconic measurement tools.

Introduction: From CAD to full-lifecycle design automation frameworks

Over the last decade, designers of various products have come to rely increasingly on computer-aided design (CAD) tools for editing and visualization. The initial design tools supported direct editing and straightforward visualization of spatial design and text annotation in domains such as mechanical design and mask layout for integrated circuits. Second-generation software began to support applications involving simulation, such as digital schematic entry coupled with behavioral simulation. Typically, the focus of these tools was narrow, with separate programs and interfaces used for each successive stage of design.

The current generation of CAD products supports complex, partially automated design tasks (such as computer system design) consisting of many processing steps performed in sequence. In such applications, several trends have emerged to increase the need for tool integration and visualization of results of what were previously considered separate design activities. Designer productivity has been raised by automating a range of design tasks; but the information generated must still be reviewed to balance requirements, invoke and control constructive automation tools, and make manual changes and edits where necessary to improve designs or compensate for imperfect automation. The amount of information generated per designer is generally greater, so that the cross-referencing previously acceptable among different design and analysis tools would be unmanageable today. Competitive pressures have also forced designers (and CAD software developers) to take many more factors into account to maximize performance and minimize both design and manufacturing costs.

Current logic circuit technologies offer such high circuit density, small devices, and narrow interconnection linewidths that system performance is increasingly determined by the quality of the layout and routing. Since a greater portion of computer system delay is in wire interconnections, there is increased emphasis on early prediction and rapid convergence of feasible layout and routing which obey timing constraints. System

designers must use logic synthesis and automatic placement and routing techniques for design productivity, yet the use of these automation tools makes early system timing more difficult when they are applied to an entire chip at once to achieve the best possible automatic result. To better predict and control system timing and design feasibility, many chip-design groups are adopting a structured chip-design methodology, in contrast to flat logic synthesis or complete "bottom-up" macrocell designs with no global physical or timing influence. A floor-plan editor and a visualization environment are critical to the productive use of a structured strategy. Abstract design analysis tools such as floor-planning can provide feedback from early stages in the development cycle, permitting design changes before much manual work has been done and eliminating the need for many attempts to do the designs with traditional CAD tools [1, 2].

Views and subviews

Designers must manage simultaneous and often conflicting requirements, so they must look at several dimensions or domains of data at the same time. Design subsystems using general optimization techniques such as simulated annealing attempt to balance multiple constraints and objectives [3]. When all goals cannot be met, designers resort to visualization of results to make judgments about what to change. To assist in this analysis, we employ many views of the data. Subviews within floor-planning (which is, in effect, an abstract physical view) include object attributes and geometry, network topology, and measurement subviews such as routing congestion maps, timing problems, and net "global" routes. Such displays may be composed in one window, or in multiple windows with less visual density.

Figures 1–3 are screen displays composed of subviews within a single window. Typical subviews presented are the basic layout of circuit region outlines (the actual "floor-plan" of the chip), and the connections between circuits (referred to as nets). Circuit outlines are presented by default as a porosity attribute, which indicates whether those circuits have space designed in for nets to pass over them. Other attributes displayed in the layout subview include population numbers and connectivity (a summary of the connections between circuits,

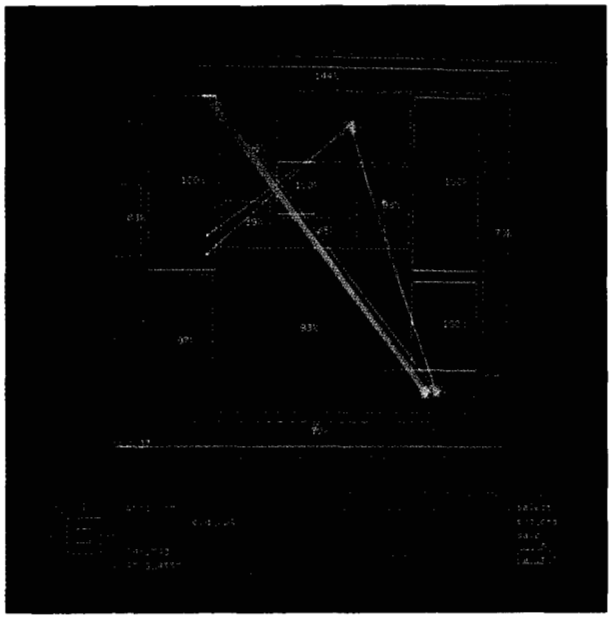

Figure 1

Network subview overlaid on layout subview. Nets not meeting performance targets are adaptively displayed as "flylines" prior to routing. Note randomization for easier viewing of problem severity, and use of cues to indicate design status. Solid lines or dashed distinguish macrocells or random logic regions; small rectangles or asterisks correspond to exact or estimated contact points for nets. Circuit layout edges are colored (green or red) to show routing availability at the global level. Local population numbers are displayed as percentages inside the circuit region, indicating placement and routing feasibility for the many small circuits within that region.

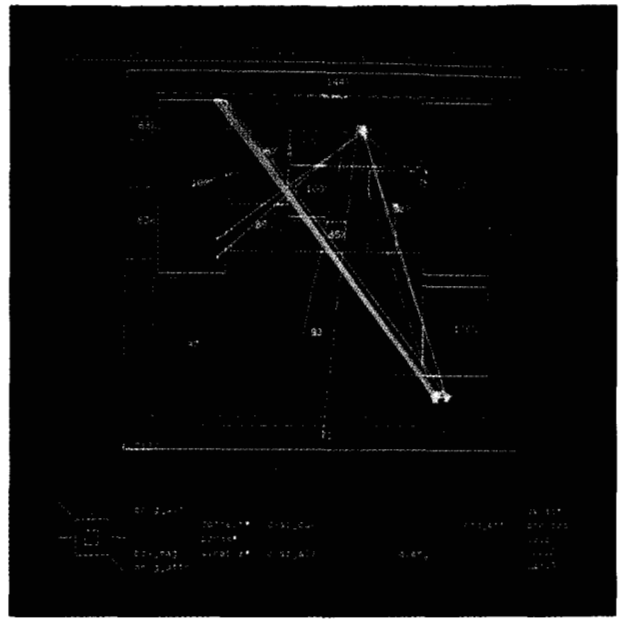

Figure 2

Fixing timing problems using network and layout subview. When layout adjustments are made, population and porosity layout subviews are automatically updated. The timing problem subview was regenerated after some changes reduced the number of problems. A proposed change is evaluated by turning on the "gross connectivity" subview, scope to the current object, and selecting affected objects in turn.

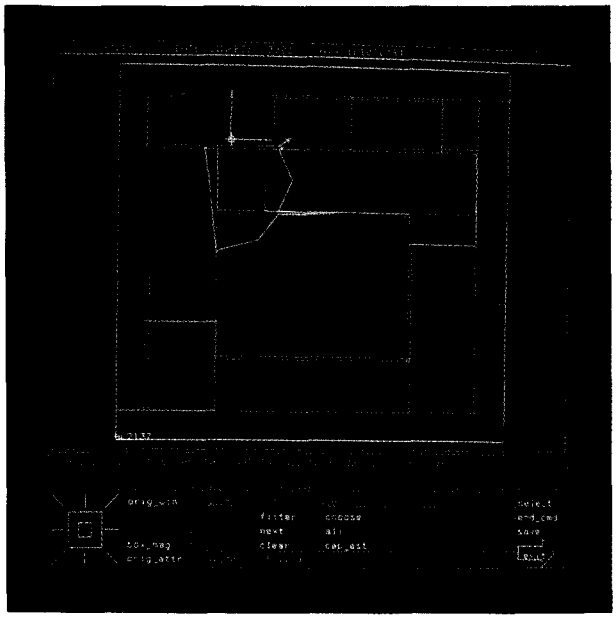

Figure 3

Slow paths overlaid on congestion map. After routing, exact paths are shown, rather than the "air line" approximations. Paths are randomized for better presentation of patterns. Known macrocell pins are denoted by blocks; unassigned pins on random logic (gate) macros are denoted with asterisks. Use of the interleaved spectrum clarifies the interpretation of vectors in the display, so that congested edges are not mistaken for components in the net path.

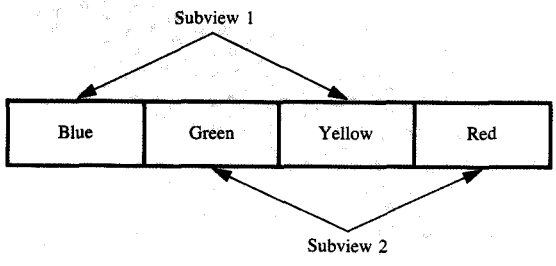

Figure 4

Interleaved spectrum.

represented as a band with width corresponding to the number of individual nets).

The attributes and subviews are controlled or "scoped" by activating them for all displayed design objects, for a selected group of objects, or only for the currently selected object. If multiple floor-plan views in separate windows are used, each window has independent scoping and control information. When an object is modified, the view update command is broadcast to all copies of the application. All copies use the same model data in memory, maintaining consistency. The actual display of objects is controlled by hierarchy-scoping commands. After an object has been selected, its components may be displayed (if subviews are activated) or hidden. This hierarchy scoping is maintained independently for each window.

The floor-planner can also be used as a control environment to invoke more detailed design tasks such as gate-level automatic and interactive placement and routing. The visual analysis provided by the floor-planner reveals the high-risk areas which should be explored first in the detailed design tools or in other views. (Gate-level synthesis and path tuning, or more fundamental structural changes, are accessed through other views for problems not solvable in physical design.)

Adaptive presentation of incrementally refined design

In this new design environment, the design data evolve incrementally over the full product lifecycle. Design is a flexible, iterative process using constructive automation actions, visualization, and analysis, with editing and parameter selection between steps. Software must take into account the incremental process and partially known state, adaptively showing the data in the most well-defined representation available. Forcing designers to invoke more commands to see each increment would complicate an already difficult task. Over time, we have reduced the number of required subviews designers must select, unifying them and automatically presenting them as they are generated. This is an important distinction between a design automation environment and other CAD or scientific visualization tasks that present complex, but complete and well-defined or measured data [4].

Figures 1–3 illustrate the adaptive refinement of net path presentation, with cues to indicate the state of each design object. Solid lines in the default outline and porosity views indicate reusable circuit components; dashed lines represent regions where many small circuits should be placed, indicating that designers have flexibility in the layout of that function.

Consistency and color mapping

Another challenge in complex design applications involves consistent visual presentation of various subviews. The primary technique we employ is the systematic use of a color-mapping metaphor to indicate the quality of some measured result. A "temperature spectrum" metaphor is used, with cool colors (blue–

green) representing feasible designs or measured good results and hot (yellow–red) indicating problems or marginal results. This mapping is applied uniformly in our application to diverse visualization elements: local wire demand, global wire availability, routing congestion, and net timing-target goal completion. Because in some cases two dimensions of information must be displayed in the same space, we modify the color strategy to an "interleaved hot–cool" metaphor (**Figure 4**). Experience has shown us that a consistent purist use of a "traffic light" metaphor is confusing when the subviews are overlaid.

Interleaved spectrum

When the same scale was used for both subviews, we found that routes or portions of routes might be confused with cuts used to measure congestion. We then settled on the interleaved scheme, so that one subview of a concurrently displayed pair might use green–red, while the other used blue–yellow. For example, in views of global routes to manage both routing congestion and timing problems, congested edges use the green–red coding, and overlaid routes for individual nets use blue–yellow coding for timing problems. The complementary colors used are also easily distinguished [5].

Figure 3 illustrates the use of alternating color mapping for concurrent subviews. Proposed connection routes which may fail timing constraints are displayed against the background of a routing congestion map.

When both dimensions of the design can be visualized effectively, trade-offs can be made and problems in one domain which impact another can be identified. Often, a few timing problems arise because a local wire congestion problem results in "roundabout" or indirect wiring paths. If chip population permits, more space can be allotted locally, allowing the timing-sensitive paths to follow more direct routes. Designers may also invoke the global router with different weighting factors on timing or routing congestion [6].

Color mapping for program control

The same color-mapping scheme is employed in cases where controls for automated subsystems are interactively specified. Cool colors indicate that programs are free to modify some value or use some resource; hot colors indicate that some value cannot be changed or is blocked for the program.

Figure 5 illustrates the use of this visual indicator to control and display the state of global routing setup parameters.

Dynamic icon generation

A technique used in the floor-planner which we believe to be novel is the construction of iconic measurement

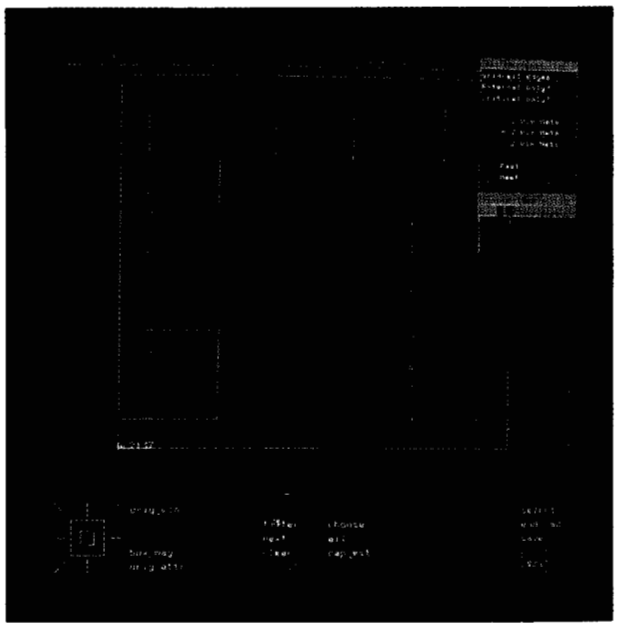

Figure 5

Controlling global routing through regions. Green circles in a region indicate that the routes may pass through the region; red circles mean that region is forbidden unless an explicit connection to the region is required. The large circle in the lower left is a "global" controller which toggles the state of all the regions.

tools based on measured data. The designer can use a customized "ruler" built for each slow net to make a floor-plan correction based on the maximum distance the connection can span while meeting its performance target. This translates the abstract electrical performance data into a form directly related to the effective actions which can be taken in the interactive floor-planning environment.

Figure 6 shows a ruler icon, displayed together with the route chosen for that circuit connection.

Interactive checking

The floor-planner is designed to support a range of design styles, including full custom, and to work in an exploratory fashion with early (possibly incomplete) data; as a general philosophy, all constraints on layouts and legality conditions are optional. The many interactive operations supported allow a good deal of robustness, which is useful for getting good results when developing new design approaches or dealing with inconsistent or incomplete data. Checks for illegalities can be turned on during interactive commands, or they may simply be invoked after many edit operations as a "debugger." In this case, floor-plan violations are highlighted visually for ease in correcting overlap or local wiring (population) errors.

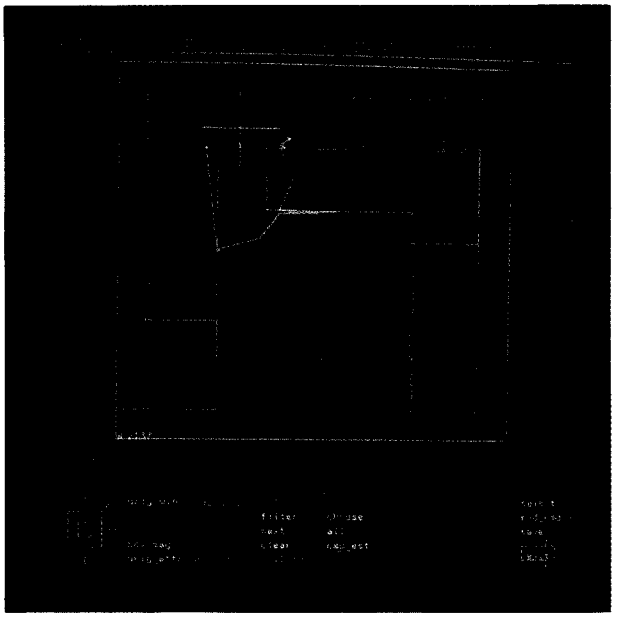

Figure 6

Slow path with iconic timing ruler. A net with a timing problem is displayed. The cursor (red crosshair) is displayed as a timing "ruler" sized to the maximum length the net could span horizontally or vertically and still meet a performance goal. Rulers for all nets reviewed may be placed into the timing view display and left until that subview is cleared. The designer can use these as guides for floor-plan changes such as moving circuits or changing the shapes of flexible circuit regions.

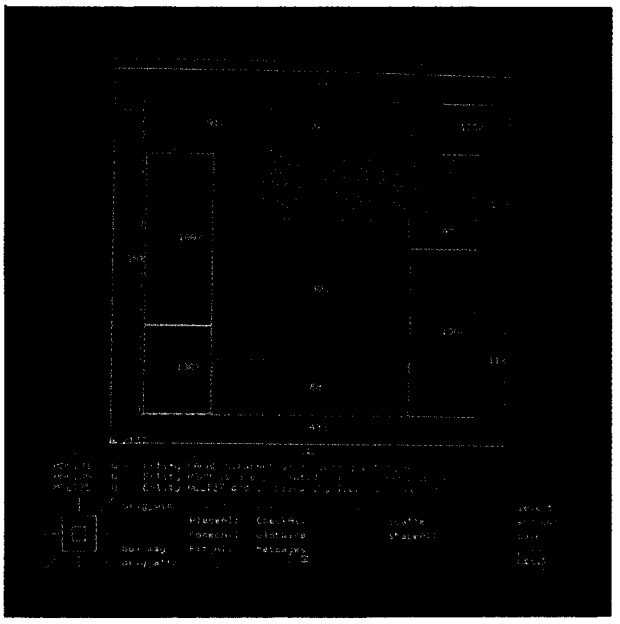

Figure 7

Error checking. Floor-plan errors are highlighted for debugging. Crosshatched areas indicate a violation of design rules, such as overlap or regions too small to contain the logic specified for them.

Figure 7 illustrates the use of crosshatched fill patterns to indicate design objects which violate design constraints or overlap rules.

Data overload: detail hiding, sequencing

Perhaps the most challenging visualization problem to address is that of data overload. While we have touched on this regarding the use of multiple views to manage conflicting requirements, the need for greater detail within a single analysis subview such as routing or timing poses special requirements.

In some cases we show abstract maps with optional detailed attributes, such as exact routing supply and demand numbers. In this case, numerical values are shown only for problem regions. It is still possible to generate busy overlaid displays in which individual paths cannot be distinguished, or interfere with other important data in making adjustments.

The only alternative to simultaneous presentation is sequential presentation; we have employed a uniform approach based on data filters and queues. After selecting a set of interesting objects (such as slow paths or nets in a congested region), a design may iterate through the queue one at a time, draw all at once, or select any subset. When drawing the queue one at a time, each object may be overlaid on the previous object, or each subview may be cleared after it has been displayed.

We have also used animation on subview sequences such as timing problems or congestion to give an overview leading to conclusions or further queries. When presenting data sequences, we have developed heuristics to draw attention, so that users are aware that a new item is displayed even though the actual "data points" may be the same as those for the item just previously displayed. For example, we randomize points within the legal range of unassigned values when displaying net paths, so that a number of similar global routes are distinguished. The randomization technique is also used to spread out the display points in simultaneous or overlaid displays.

Figures 1 and 3 illustrate the use of randomization in network displays to help distinguish individual data objects with similar or identical data.

Figure 8 shows a selection menu used for filtering network views.

Summary and future work

While the visualization aspects of the floor-plan system are still undergoing refinement, it has already proven useful to designers. It has also been of value to algorithm developers, who obtain much better feedback from users

on problems than was obtained with previous batch-oriented systems, in which only final detailed results, such as exact routes on plots of finished designs, could be examined.

Future developments are expected to take the form of generalizing the techniques and data structures described here and incorporating them into an object-oriented application framework, helping to enforce consistency and reduce development time. Given that many previous object-oriented frameworks and view/control paradigms [7] originated on monochrome displays, the systematic use of color at the framework level still needs definition and standardization. Other features discussed here, such as stepping through queued subviews and dynamically built iconic measurement tools, should be provided as reusable framework components.

Acknowledgments

The interactive and visual aspects of the Maple floor-plan editor owe much to the efforts of Dan Mainiero, Tim Holohan, John Thorvaldson, William Livingstone, and particularly Kevin McCullen. We also appreciate the comments and patience of early users; Patrick Lampin and Juergen Koehl should be singled out for their help and encouragement.

References

1. Kevin McCullen, John Thorvaldson, David DeMaris, and Patrick Lampin, "A System for Floorplanning with Hierarchical Placement and Wiring," *Proceedings of the 1990 European Design Automation Conference,* March 1990, pp. 262–265.
2. Ravi Nair, C. Leonard Berman, Peter S. Hauge, and Ellen J. Yoffa, "Generation of Performance Constraints for Layout," *IEEE Trans. Computer-Aided Design* **8,** No. 8, 860–874 (August 1989).
3. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science,* May 1983.
4. Bruce H. McCormick, Thomas A. DeFanti, and Maxine D. Brown, "Visualization in Scientific Computing," *Comput. Graph.* **21,** No. 6, 3 (November 1987).
5. Gerald M. Murch, "Physiological Principles for the Effective Use of Color," *IEEE Comput. Graph. & Appl.* **4,** No. 11, 49–53 (November 1984).
6. W. K. Luk, Paola Sipala, Markku Tamminen, Donald Tang, Lin S. Woo, and C. K. Wong, "A Hierarchical Global Wiring Algorithm for Custom Chip Design," *IEEE Trans. Computer-Aided Design* **CAD-6,** No. 4, 518–533 (July 1987).
7. Glenn E. Krasner and Stephen T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm," *J. Object-Oriented Programming* **1,** No. 3, 26–49 (August 1988).

Received November 16, 1989; accepted for publication September 7, 1990

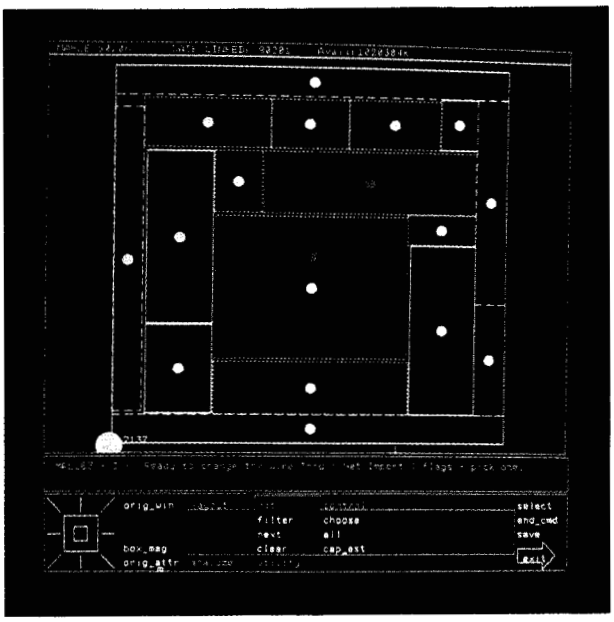

Figure 8

A filter presented for selection. The net view filter allows graphics in the congestion subview to be queried to select a subset of interesting nets, such as slow nets through a congested area. A queue of nets is returned which can be reviewed, edited, or passed as input to other views and commands in the environment.

David L. DeMaris *IBM General Technology Division, Burlington facility, Essex Junction, Vermont 05452.* Mr. DeMaris joined the IBM Burlington laboratory in 1982 after receiving his B.S.E.E. from the University of Illinois. He has worked on logic modeling, RISC microprocessors, VLSI design methodologies, and computer-aided design software. Mr. DeMaris' technical interests include computer architecture, design methodologies, design and programming frameworks, and applications of discrete complex systems dynamics in pattern recognition and data compression.