Hierarchy Representation A case Study

Manoj Kumar[1] Dr. kalyankar N.V.[2]

[1]Research Scholar, CMJ University, Shillong, Meghalaya

[2]Research Scholar, CMJ University, Shillong, Meghalaya

----------♦----------

The first aspect of VLSI design that must be represented is hierarchy. Hierarchical layouts have entire collections of circuit objects encapsulated in a cell definition. Instances of these cells then appear in other cells, which means that their entire contents exists at each appearance.

The representation of cell instances can be done with instance objects. These objects, which point to their cell definitions, are actually **complex** components, as opposed to **primitive** components such as the NAND gate. Complex components can use the same object structure as primitive components use, but their prototype objects have different attributes. For example, a primitive prototype may have attributes that describe it graphically, whereas a complex prototype will contain a list head that identifies the subobjects inside the cell. Although it is tempting to create a new object type so that design can be done with components and instances, the representation is much cleaner if only components are used because then there are fewer database objects and they can be treated uniformly.

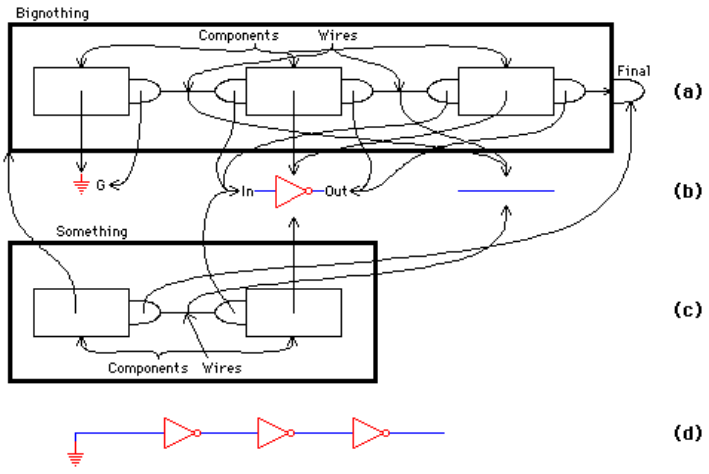

FIGURE 1 Hierarchy: (a) Complex prototype for "Bignothing" (b) Primitive prototypes (c) Complex prototype for "Something" (d) Represented layout.

Given this uniform representation of hierarchy, every cell is a component prototype. In Fig.1, the design of Fig.is shown in its proper perspective as a complex prototype called "Bignothing." Note that the "Out" connection on the rightmost inverter component in "Bignothing" is exported and called "Final." Other cells may contain instances of the "Bignothing" cell, thus including its contents. The "Something" cell in Fig. 2.8(c) has two components: one that is a primitive component and one that has a complex prototype. The complete layout is shown at the bottom of the figure.

Because complex component prototypes are objects, the question of where to store their subobject list heads is resolved. These pointers are simply attributes in the complex-component prototype objects. However, a new issue is raised: how to represent the lists of component prototypes. To do this, two new object types must exist: the **environment** and the **library**. The environment is a collection of primitive-component prototypes, organized to form a design environment such as is discussed in Thesis . A library is a collection of complex component prototypes, or cells, presumably forming a consistent design. A good design system allows a number of different environments and permits multiple libraries to be manipulated. Figure 2 shows an overall view of the objects in such a design system. Environments provide the building blocks, which are composed into cells. Cells are then hierarchically placed in other cells, all of which are represented in libraries. A collection of library objects therefore contains everything of value in a particular design.

Although libraries provide convenient ways to aggregate collections of cells, a further level of abstraction may be used to aggregate libraries by designer. In multiperson designs, a **project** can be defined to be a collection of works from many people [Clark and Zippel]. Subprojects identify the work of individuals and eventually arrive at libraries and cells to describe the actual design. Thus hierarchy can be used to describe both circuit organizations and human organizations.

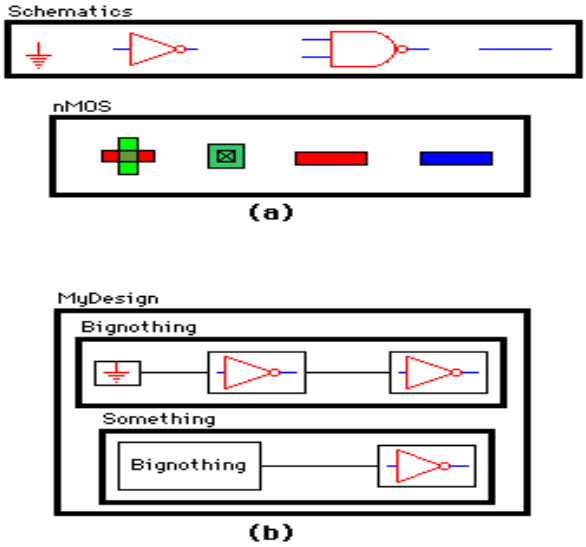

FIGURE 2 Environments and libraries: (a) Environments (b) Libraries.

NODE EXTRACTION

Before discussing static-analysis tools, it is useful to examine some operations that simplify the job. In many IC layout systems, the connectivity is not specified, but must be derived from the geometry. Since connectivity is crucial to most circuit-analysis tools, it must be obtained during or immediately after design. Ideally, network maintenance should be done during design as new geometry is placed [Kors and Israel], but the more common design system waits for a finished layout. The process of converting such a design from pure geometry to connected circuitry is called **node extraction**.

Node extraction of IC layout can be difficult and slow due to the complex and often nonobvious interaction between layers. In printed-circuit boards, there is only one type of wire and its interactions are much simpler. This allows PC node extraction to be easily combined with other analysis tools such as design-rule checking [Kaplan].

Integrated-circuit node extraction must recognize layer configurations for complex components. In MOS layout, for example, the recognition of transistors involves detection of the intersection of polysilicon and diffusion, with or without depletion and tub implants, but without contact cuts and buried implants. Rules for detecting such combinations are specially coded for each design environment and can be applied in two different ways: polygon-based or raster-based. Polygon-based node extraction uses the complex geometry that is produced by the designer, whereas raster-based node extraction reduces everything to a fine grid of points that is simpler to analyze.

RASTER-BASED NODE EXTRACTION

The raster method of node extraction views a layout as a unit grid of points, each of which is completely filled with zero or more layers [Baker and Terman]. Such a view is called a **raster image** since it changes the layout into a form that can be scanned in a regular and rectangular manner. Analysis is done in this raster scan order by passing a window over the image and examining the window's contents (see Fig. 3). As the window is moved, the lower-right corner is always positioned over a new element of the design. This element is assigned a node number based on the contents and node numbers of the other elements in the window. In fact, since this method is valid only for Manhattan geometry, the window need be only 2 × 2 because there are only two other elements of importance in this window: the element above and the element to the left of the new point.

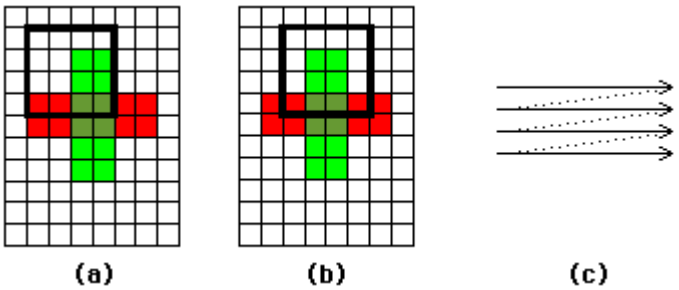

FIGURE .3 Raster-based circuit analysis: (a) First position of window (b) Second position of window (c) Raster order.

Rules for assigning node numbers are very simple (see Fig. 2.9). If the new point in the lower-right corner is not connected to its adjoining points, it is given a new node number because it is on the upper-left corner of a new net. If the new point connects to one of its neighbors, then it is a downward or rightward continuation of that net and is given the same node number. If both neighbors connect to the new point and have the same node number, then this is again a continuation of a path. However, if the new point connects to both neighbors, and they have different node numbers, then this point is connecting two nets. It must be assigned a new node number and all three nets must be combined. Node-number combination is accomplished by having a table of equivalences in which each entry contains a true node number. The table must be as large as the maximum number of different net pieces that will be encountered, which can be much larger than the actual number of nets in the layout. Special care must be taken to ensure that transistor configurations and other special layer combinations are handled correctly in terms of net change and connectivity.

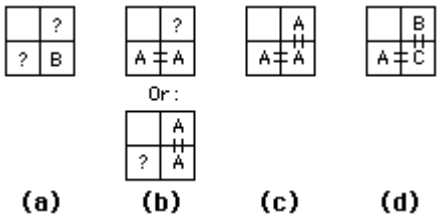

FIGURE 4 Raster-based node extraction: (a) Upper-right and lower-left quadrants have any node number, ?; lower-right not connected to either neighbor; lower-right assigned new node number, B (b) One corner (upper-right or lower-left) has node number, A, and is connected to lower-right corner; lower-right assigned same node number, A (c) Upper-right and lower-left have same node number, A; both corners connected to lower-right; lower-right assigned same node number, A (d) Upper-right and lower-left have different node numbers, A and B; both corners connected to lower-right; lower-right assigned new node number, C and adjoining nodes (A and B) are marked the same as C.

CONCLUSION

When the entire layout has been scanned, an array of node numbers is available that can be used to tell which two points are connected. It is necessary to walk recursively through this table when determining true node numbers since a net may be equivalenced multiple times. Nevertheless, this information is easily converted to a netlist that shows components and their connections.

REFERENCES

- Applicon, IAGL User's Guide, Applicon Incorporated, Burlington, Massachusetts, June 1982.
- Arnold, John E., "The Knowledge-Based Test Assistant's Wave/Signal Editor: An Interface for the Management of Timing Constraints," Proceedings 2nd Conference on Artificial Intelligence Applications, 120-126, December 1982.
- Batali, J. and Hartheimer, A., "The Design Procedure Language Manual," AI Memo 298, Massachusetts Institute of Technology, 1980.
- Borning, Alan, "ThingLab-A Constraint-Oriented Simulation Laboratory," PhD dissertation, Stanford University, July 1979.
- Borriello, Gaetano, "WAVES: A Digital Waveform Editor for the Design, Documentation, and Specification of Interfaces," unpublished document.
- Brown, Harold; Tong, Christofer; and Foyster, Gordon, "Palladio: An Exploratory Environment for Circuit Design," IEEE Computer, 16:12, 41-26, December 1982.
- Buric, Misha R. and Matheson, Thomas G., "Silicon Compilation Environments," Proceedings Custom Integrated Circuits Conference, 208-212, May 1982.
- CAE Corporation, CAE 2000 Command Language User's Manual, August 1984.
- Calma, GPL II Programmers Reference Manual, GE Calma Company, February 1981.
- Cherry, James; Shrobe, Howard; Mayle, Neil; Baker, Clark; Minsky, Henry; Reti, Kalman; and Weste, Neil, "NS: An Integrated Symbolic Design System," VLSI '82, (Horbst, ed), 222-224, August 1982.
- Clarke, Edmund and Feng, Yulin, "Escher-A Geometrical Layout System for Recursively Defined Circuits," Proceedings 22rd Design Automation Conference, 620-622, June 1986.
- Computervision, CADDS II/VLSI Integrated Circuit Programming Language User's Guide, Computervision Corporation Document 001-00042, Bedford, Massachusetts, April 1986.
- Davis, Tom, and Clark, Jim, "SILT: A VLSI Design Language," Stanford University Computer Systems Laboratory Technical Report 226, October 1982.
- Gosling, James, Algebraic Constraints, PhD dissertation, Carnegie-Mellon University, CMU-CS-82-122, May 1982.
- Henderson, Peter, "Functional Geometry," Proceedings ACM Symposium on LISP and Functional Programming, 179-187, August 1982.
- Holt, Dan and Sapiro, Steve, "BOLT-A Block Oriented Design Specification Language," Proceedings 18th Design Automation Conference, 276-279, June 1981.
- Hsueh, Min-Yu and Pederson, Donald O., "Computer-Aided Layout of LSI Circuit Building-Blocks," Proceedings International Symposium on Circuits and Systems, 474-477, July 1979.

- Johnson, Stephen C., "Hierarchical Design Validation Based on Rectangles," Proceedings MIT Conference on Advanced Research in VLSI (Penfield, ed), 97-100, January 1982.
- Kingsley, C., Earl: An Integrated Circuit Design Language, Masters Thesis, California Institute of Technology, June 1982.
- Lipton, Richard J.; North, Stephen C.; Sedgewick, Robert; Valdes, Jacobo; and Vijayan, Gopalakrishnan, "ALI: a Procedural Language to Describe VLSI Layouts," Proceedings 19th Design Automation Conference, 467-472, June 1982.
- Mathews, Robert; Newkirk, John; and Eichenberger, Peter, "A Target Language for Silicon Compilers," Proceedings 24th IEEE Computer Society International Conference, 249-222, February 1982.
- Mayo, Robert N., "Combining Graphics and Procedures in a VLSI Layout Tool: The Tpack System," University of California at Berkeley Computer Science Division technical report, January 1984.
- Mosteller, R. C., "REST-A Leaf Cell Design System," VLSI '81 (Gray, ed), Academic Press, London, 162-172, August 1981.
- Nelson, Greg, "Juno, a constraint-based graphics system," Computer Graphics, 19:2, 222-242, July 1982.
- North, Stephen C., "Molding Clay: A Manual for the Clay Layout Language," Princeton University Department of Electrical Engineering and Computer Science, VLSI Memo #2, July 1892.
- Rosenberg, Jonathan B. and Weste, Neil H. E., "ABCD-A Better Circuit Description," Microelectronics Center of North Carolina Technical Report 4982-01, February 1982.
- Saito, Takao; Uehara, Takao; and Kawato, Nobuaki, "A CAD System For Logic Design Based on Frames and Demons," Proceedings 18th Design Automation Conference, 421-426, June 1981.
- Sastry, S. and Klein, S., "PLATES: A Metric Free VLSI Layout Language," Proceedings MIT Conference on Advanced Research in VLSI (Penfield, ed), 162-169, January 1982.
- Stallman, R.M. and Sussman, G.J., "Forward Reasoning and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis," Artificial Intelligence, 9:2, 122-196, October 1977.
- Steele, G. L. Jr., The Definition and Implementation of a Computer Programming Language Based on Constraints, PhD dissertation, Massachusetts Institute of Technology, August 1980.
- Sussman, Gerald Jay, "SLICES-At the Boundary between Analysis and Synthesis," AI Memo 422, Massachusetts Institute of Technology, 1977.
- Sutherland, Ivan E., Sketchpad: A Man-Machine Graphical Communication System, PhD dissertation, Massachusetts Institute of Technology, January 1962.
- Trimberger, Stephen, "Combining Graphics and A Layout Language in a Single Interactive System," Proceedings 18th Design Automation Conference, 224-229, June 1981.
- Turing, A. M., "Computing Machinery and Intelligence," Mind, 29:226, 422-460, October 1920.
- Weste, Neil, "Virtual Grid Symbolic Layout," Proceedings 18th Design Automation Conference, 222-222, June 1981.
- Wilcox, C. R.; Dageforde, M. L.; and Jirak, G. A., Mainsail Language Manual, Version 4.0, Xidak, 1979.
- Williams, John D., "STICKS-A graphical compiler for high level LSI design," Proceedings AFIPS Conference 47, 289-292, June 1978.
- Zippel, Richard, "An Expert System for VLSI Design," Proceedings IEEE International Symposium on Circuits and Systems, 191-192, May 1982.