# Caesar: An Interactive Editor for VLSI Layouts

**John K. Ousterhout**, University of California at Berkeley

Caesar is an interactive color graphics system for editing hierarchical VLSI layouts. Although its overall functions are like those of several previous systems, Caesar provides a novel user interface based on a structureless mechanism similar to painting. The painting paradigm permits powerful circuit manipulations to be performed with a simple mode-less command set. Caesar is evaluated based on its use in two large processor designs.

The EECS Department of the University of California at Berkeley has a long-standing program to develop computer aids for designers of electrical circuits (Newton, *et al.* 1981). Caesar is an interactive VLSI layout system that was developed as one piece of the program. It is written in C, runs on a VAX-11/780 under the Berkeley version of Bell Labs' UNIX operating system (Joy and Fabry 1981), and uses an AED 512 color display with an attached tablet.

Caesar is a layout geometry editor. As such, it provides overall functionality similar to that of earlier systems like ICARUS (Fairbairn and Rowson 1980). Each design consists of a hierarchical collection of cells. Each cell contains a description of patterns on several mask layers, as well as "child cells" that contain more information of the same sort. Like most VLSI layout systems, Caesar does not maintain information about design rules, electrical rules, or schematics. It produces CIF (Caltech Intermediate Form) files that can be processed by other design aids such as design rule checkers, plotters, and circuit extractors.

What distinguishes Caesar from other VLSI layout systems is its user interface. Caesar's interface has two novel facets. First, all commands are invoked using a cursor that selects a rectangular area, rather than a traditional cursor that selects a single point. Second, geometry is specified with a mechanism like *painting*. The rectangular cursor and painting paradigm results in a simple command interface that lets complex manipulations be performed easily and naturally.

Caesar was used in the development of two VLSI implementations of the Reduced Instruction Set Computer (RISC) (see article by Fitzpatrick *et al.* in this issue; Patterson and Séquin 1981). The RISC implementation provided the motivation to develop Caesar from an experimental system into a usable product. The RISC designers were a constant source of ideas for enhancements. The RISC chips demonstrate that Caesar is a viable tool for large designs.

## The Basic Command Interface

A Caesar workstation consists of a standard video terminal, an AED 512 color display, and a tablet. The video terminal is used for command input and also to display command menus, statistics about the circuit, and error messages.

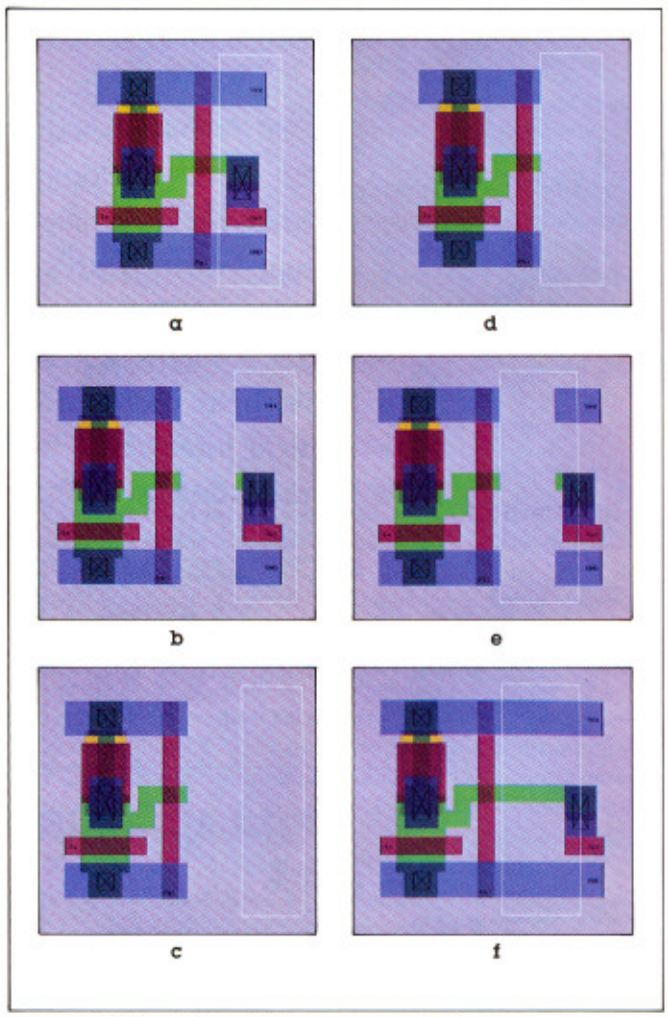



**FIGURE 1. Using the painting commands to stretch an nMOS cell:**
**a) before ERASE; b) after ERASE; c) before PUT; d) after PUT; e) before FILL RIGHT; f) after FILL RIGHT.**

Almost all graphics systems except Caesar are point-based. A pointing device is used to select a point on the screen, and then a command is invoked to process that point. Because IC mask features consist of areas rather than points or lines, Caesar's command interface is area-based. Using a tablet, the user positions a rectangular cursor over a portion of the

circuit, and then issues a command to process the area enclosed by the cursor. The rectangular cursor leads to a natural implementation of commands. A single point provides less information than an area; in point-based systems it is necessary to enter special modes during some commands so that the user can select more points. A rectangle provides just the right amount of information for virtually all Caesar commands and avoids the need for modes. (See Tesler 1981 for a discussion of the advantages of modeless user interfaces.)

For example, a rectangular cursor can be used to zoom in on a piece of the circuit. The area selected by the cursor is then expanded to fill the entire screen. Another use is to specify a grid pattern; the dimensions and location of the cursor determine the grid spacing and alignment. Other uses for a rectangular cursor are described below.

The use of a rectangular cursor for all command invocations means that Caesar is a *Manhattan* system: all feature edges are vertical or horizontal. This restriction, and its impact on the design environment, are discussed below.

## The Painting Paradigm

This section compares Caesar's interface for specifying mask geometries to the interface of other layout systems. The following section discusses the facilities for building hierarchical cell structures.

The user interfaces provided by most VLSI layout systems for editing mask geometries are direct reflections of the underlying circuit-specification languages (CIF, in many cases). Because the languages provide primitives for specifying rectangles, wires, and polygons, the layout systems allow users to do the same sorts of things. These interfaces are *object-based*; that is the user views a circuit in terms of the objects (*e.g.*, polygons) that can be created, modified (in some systems), and deleted.

Unfortunately, object-oriented interfaces force the designer to deal with structure that is irrelevant and often invisible. It makes no difference how a particular mask pattern is composed of boxes, polygons, and wires: when viewed on a color display, the pattern will probably appear as a solid mass with no evident structure. When fabricated, the circuit's behavior depends only on the pattern, not on how it is formed.

In object-based systems, modifications of patterns are object-specific; that is, commands to modify rectangles are different from commands to modify polygons. Because of these differences, it is difficult to provide commands to manipulate large collections of objects, for example, to stretch a cell. Object-based command interfaces tend to be either complicated, not very powerful, or both.

Caesar provides a structure-less paradigm for editing mask geometries. The system maintains internally a structure based on rectangles, but the user manipulates a mask pattern as an amorphous blob of *paint* with no apparent structure.

There are five basic commands: PAINT, ERASE, YANK, PUT, and FILL. The PAINT command paints the area enclosed by the cursor with one or more mask layers, and ERASE removes one or more mask layers from the cursor area. Caesar automatically splits rectangles when necessary, and merges them again whenever possible. The user is not allowed to see the rectangle structure. The YANK command treats the cursor like a cookie cutter, and presses it into the "dough" of the underlying paint. All information lying underneath the cursor is

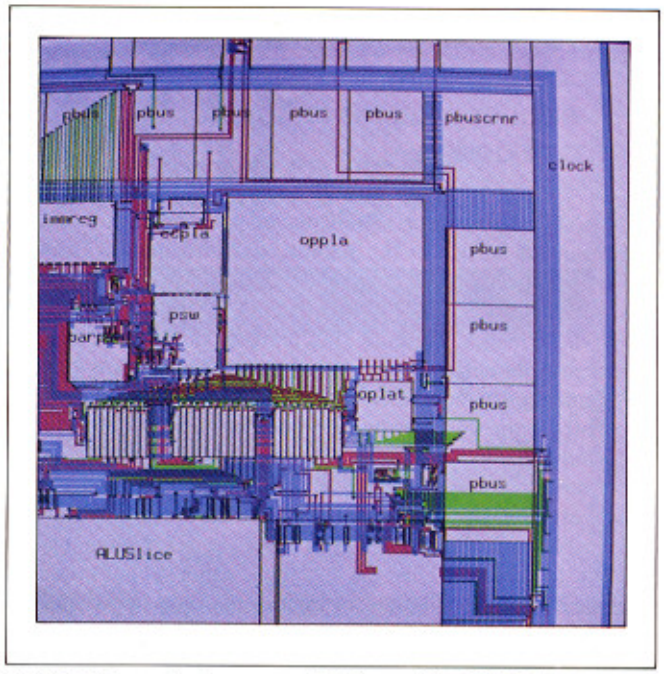



**FIGURE 2. A partially expanded view of the RISC Gold chip.**

copied into an area called the "yank buffer," without affecting the existing paint. The PUT command copies the information from the yank buffer back to the current cursor location. The FILL command takes an imprint of all paint lying underneath one edge of the cursor, then uses that imprint as a "paint brush," sweeping the brush across the cursor to the opposite edge (see Figure 1).

Although conceptually simple, the painting commands permit powerful operations on large areas of design. Figure 1 shows how the commands can be combined to stretch a cell. Stretching is accomplished with three commands. ERASE is used to delete one side of the cell (as part of ERASE, the deleted information is automatically saved in the yank buffer); PUT is used to restore the information at its stretched location; and FILL RIGHT is used to cover the gap. The various rectangles are automatically split, clipped, and merged without the assistance, or even the knowledge, of the user.

## Cell Manipulation

Although Caesar eliminates object structure within the mask patterns of individual cells, it permits cells to be arranged hierarchically into larger designs. Cells are objects that can be selected, moved, rotated, mirrored, copied, arrayed, and deleted. Each cell may be displayed either in *expanded* form (where its paint, and subcells are visible), or in *unexpanded* form (where only a bounding box and name are displayed). The default case is for subcells to be unexpanded, to speed up redisplay and highlight the logical structure of the circuit. Most of Caesar's cell facilities are similar to those of other layout systems. Figure 2 shows a partially expanded view of a portion of the RISC Gold chip.

One unusual feature of Caesar's cell commands is the ability to edit a cell in context. Modifications of a subcell often depend on the way the cell is used in its parent. For example, when a subcell fails to connect properly to other features in its parent, it must be modified to do so. In Caesar, the subcell can

Although conceptually simple, the painting commands permit powerful operations on large areas of design.

be modified *in context*: it is displayed with the orientation and location of its use in the parent, and information from the parent cell is displayed around it. Caesar translates commands into the coordinate system of the subcell, so as to carry out the changes. Without this feature, the subcell must be edited in its native coordinate system. This means that the designer must remember the context of usage and mentally transform the desired changes into the coordinate system of the subcell.

Another convenient use of the rectangular cursor is to convert a single instance of a cell to an array. The cursor's dimensions determine the spacing of the array elements in $x$ and $y$. In other systems, the spacing is either set automatically to the size of the cell (thereby making overlaps impossible), or specified numerically.

## Implementation

A cell definition is represented internally as a list of rectangles for each layer, and a list of subcell uses. Each subcell use contains a transformation and a pointer to the definition for the subcell. This implementation is similar to the implementations of several other systems, and will not be described in more detail here. The following paragraphs discuss a few aspects of the implementation that are less obvious, or that worked out especially well.

### Bounding Boxes

Any single command has bearing on only a small area in the design, usually the area displayed on the screen or the area selected by the cursor. When scanning the data-base for elements in this area, Caesar uses cell bounding boxes to limit its search: if a cell's bounding box doesn't overlap the area of interest, then that cell's paint and children are not examined. When a cell is stored on disk, the bounding boxes for all of its children are stored with it. This allows Caesar to load its main-store data-base incrementally. At the beginning of an editing session only the root cell is loaded into main memory. Subcells are not expanded and need not be loaded until they are expanded. If only a small piece of a large circuit is edited, most of the circuit is never loaded from the disk. For the RISC chips, where the data-bases each contain megabytes of information, this is an important optimization.

### Transparent and Opaque Layers

The simplest and most time-efficient way to use the raster memory of the color display is to allocate one of the bits of each pixel for each mask layer. The AED 512 display contains 8 bits per pixel, of which two are used for bounding boxes, labels, the grid, and the cursor. If the simple approach is taken, the remaining 6 bits per pixel are barely enough for the simplest nMOS process, and are insufficient for more complex processes, such as CMOS. To support processes with more than six layers, Caesar divides the mask layers into two groups, *transparent* and *opaque*. There may be up to 5 transparent and 32 opaque layers in any given technology. Each opaque layer obscures all other layers underneath it but transparent layers do not blot out underlying details. Thus, at points where only transparent layers are present, it is possible to see all overlaps. The densest layers, and those whose overlaps are important, are typically made transparent. A display with more bits per pixel would, of course, reduce this problem greatly.

### Display Resolution

Although there has been much hullabaloo of late about the "need" for high-resolution color displays with 1000 or more lines (Rosenberg and Fuchs 1981), we have been satisfied with the 512x512 resolution of the AED display. In typical usage, the features are 5 to 20 pixels across. Therefore, a higher-resolution display would not improve picture quality very much. With more pixels to modify, 1024-line displays are liable to be substantially slower than 512-line ones, unless special attention is given to their implementations. Furthermore, to keep memory costs down, 1024-line designs occasionally reduce the number of bits per pixel as they increase the number of pixels. Our experience suggests that this is exactly the wrong trade-off to make. Our satisfaction with 512-line resolution is partly a result of our restriction to Manhattan geometries. The aliasing problem makes 512-line displays less attractive for non-Manhattan shapes.

### Interface to the Color Display

Communication between the VAX-based software and the color display is over a 9600-baud link. Although we feared that this would be too slow to provide good interactive response, it turned out to be quite adequate. The AED display contains a microprocessor that accepts commands of the form "draw a filled rectangle" or "draw a line." Thus, only small amounts of data need be shipped over the link to specify complex operations. For typical usage, the system is well-balanced with respect to VAX processing time, character transmission time, and command processing time in the AED microprocessor.

## Development and Performance Statistics

Caesar was implemented by one person working about half time over about a year. The program is written in C, and contains about 12,000 lines of code and comments. It has been distributed to about a dozen sites outside Berkeley.

Running on a VAX-11/780 with the 4.1 release of Berkeley UNIX, Caesar requires a total of about 100 kbytes of virtual storage to edit small cells. For the RISC Gold design (a microprocessor containing 44,500 transistors), about 1 megabyte of storage is required if all of the cell definitions are loaded. (In practice, the incremental loading feature discribed in the Bounding Boxes paragraph above, reduces this figure substantially).

Caesar is instrumented to record its usage on a session-by-session basis. Table 1 summarizes the data for two groups of sessions. The first column summarizes data from sessions in the last month of development of the RISC Gold chip, when Caesar was being used intensively. The second column represents sessions during which the system was being used for development and small-cell layout; this information indicates its performance during the early stages of a project.

| | Heavy Usage | Light Usage |
|---|---|---|
| Number of Sessions | 784 | 595 |
| Average CPU Usage (execution time divided by elapsed time) | 6.9% | 2.6% |
| Averaged elapsed time to process each command | 1.0 sec. | .9 sec. |

**TABLE 1. Usage Statistics.**

Even under intensive use, Caesar requires only about 7% of the CPU. For small cells, Caesar's CPU needs are about 3%, or about the same as a screen-based text editor. On the average, commands were completed in about one second of elapsed time, and the system spent about three-fourths of its time waiting for user input. It should also be noted that statistics in the "Light Usage" column were gathered on a loaded machine (typically 15 to 30 users logged in), whereas the machine used for the "Heavy Usage" statistics had only a few users.

## Evaluation

I was fortunate to have a group of ambitious and adventuresome users: the RISC designers. In addition to uncovering bugs and suggesting enhancements, their usage of Caesar for two large designs brought out some of the advantages and disadvantages of the system. This section evaluates Caesar based on the RISC experience.

### Things That Worked Well

The general user interface worked out very well, especially for low-level cell design. This is due to the combination of the rectangular cursor and the painting paradigm. Together they eliminate the sub-modes and irrelevant structure present in other interfaces, and provide powerful facilities with a simple command set.

The restriction to Manhattan designs was also successful. I expected designers to chafe at this restriction, and there was some initial discomfort. In retrospect, however, the designers are happy with a Manhattan design style. By tailoring the design tools to Manhattan structures, we have achieved much greater efficiency: the circuit extractor, for example, sped up by a factor of 10 when recoded for Manhattan designs (Fitzpatrick 1981). This speedup made possible a degree of thoroughness in circuit simulation that would have been out of the question if arbitrary orientations had been present. Furthermore, designers at Berkeley and elsewhere agree that non-Manhattan designs are more error-prone than Manhattan ones. The RISC designers estimate that less than 10% in circuit density was lost due to the Manhattan restriction.

The incremental loading features and the use of bounding boxes to reduce search time proved important for large designs, such as the RISC chip. The ability to edit cells in context has only recently been added to Caesar, but initial reaction is extremely positive. This feature increases Caesar's efficiency, encourages a greater degree of modularity in designs, and makes it easier for different designers to work on different pieces of a circuit at the same time.

### Inadequacies

Although Caesar's painting paradigm works well for creating cells, its mechanisms (like those of most other layout systems) are inadequate for connecting cells. The problem stems from the fact that cells are *placed* in absolute space, not *connected* to each other. Thus, if a wire is drawn between two points, the layout system does not record the fact that the wire is intended to connect them. If one of the points moves, the designer must be careful to modify the wire so that the connection is maintained. The layout system should incorporate the notion of "connection," with tools to help the designers create and maintain connections. The solution is not a return to the old style of designing with CIF wires. Rather, we must develop a new higher-level kind of object, the connection. Work is already underway at Berkeley and elsewhere to explore connection-based (or constraint-based) systems.


### Acknowledgments

Good design tools cannot be developed in isolation: they require constant usage and evaluation. For Caesar, Dave Patterson and the RISC designers filled this role. They were ambitious enough to stress the system to (and occasionally past) its limits, and patient enough to wait for bug fixes and enhancements. The current system bears only slight resemblance to the system they began using in January 1981.

The transparent/opaque layer division and technology independence of Caesar developed from discussions with Ken Keller, and from comparisons between early versions of Caesar and Keller's KIC system (Keller 1981).

The work described here was supported in part by the Miller Institute of the University of California, and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3803, monitored by the Naval Electronic System Command under Contract No. N00039-78-G-0013-0004. The views and conclusions contained in this article are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advance Research Projects Agency or the U.S. Government.


### References


Fairbairn, D.G. and J.A. Rowson. 1980. *ICARUS2 User's Manual.* Xerox Palo Alto Research Center internal document.

Fitzpatrick, Daniel T. 1981. Private communication.

Joy, W.N. and R.S. Fabry. 1981. "Berkeley Software for UNIX on the VAX". Computer Science Division Internal Memo, University of California, Berkeley.

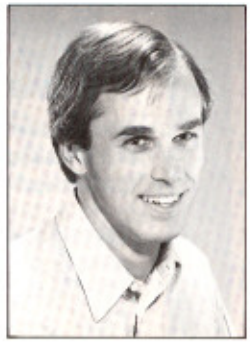
Keller, Ken. 1981. "KIC: A Graphic's Editor for Integrated Circuits." Master's thesis, EECS Department, University of California, Berkeley.

Newton, A.R., D.O. Pederson, A.L. Sangiovanni-Vincentelli, and C.H. Séquin. July 1981. "Design Aids for VLSI: The Berkeley Perspective," *IEEE Transactions on Circuits and Systems*, Vol. CAS-28, No. 7, pp. 666-680.

Patterson, D.A. and C.H. Sequin. May 1981. "RISC I: A Reduced Instruction Set VLSI Computer." *Proceedings of the Eighth International Symposium on Computer Architecture.*

*Rosenberg, J.P. and H. Fuchs. Second Quarter 1981. "Survey and Evaluation of Color-Display Terminals for VLSI." LAMBDA*, Vol. II, No. 2, pp. 58-60.

Tesler, Larry. August 1981. "The Smalltalk Environment," *Byte*, Vol. 6, No. 8, pp. 90-147.


### About the Author

**John K. Ousterhout** received the B.S. degree in physics from Yale College in 1975 and the Ph.D. in computer science from Carnegie-Mellon University in 1980. He is currently an assistant professor of electrical engineering and computer sciences at the Berkeley campus of the University of California. His research interests include computer aided design, VLSI architecture, and operating systems.