EXPERIENCES IN PORTING AND EXTENDING CAD SOFTWARE

R. H. Katz
J. Moran
U. Ramschandran
D. Schuh

Computer Sciences Technical Report #484

September 1982

Experiences in Porting and Extending CAD Software

R. H. Katz, J. Moran, U. Ramachandran, D. Schuh
Computer Sciences Department
University of Wisconsin-Madison
Madison, WI 53706

Abstract: We describe our experiences in transporting CAD software developed at other universities to our own somewhat different computing environment. The tools were the CAESAR Graphics editor developed at Berkeley [OUST81] and the MIT raster scan design rule checker [BAKE80]. We have combined these two packages into a single system, and have experimented with implementing the DRC inside an intelligent graphics terminal (the Chromatics 7900). Our observations are that an ideal design environment should exhibit a high degree of terminal independence, integration of independent design tools subsystems, and migration of design tool functions from mainframe processors to intelligent workstations.

1. Introduction

The university research community has responded to the complexity of the task of VLSI design by developing powerful tools to aid in the layout and verification of integrated circuits [CONW80]. These tools, developed by various groups at different universities, have been readily distributed to other institutions. Two of the most widely used tools, in part because they have been written for the popular UNIX operating system and undoubtedly because of their high utility, are Berkeley's CAESAR system for integrated circuit layout [OUST81] and MIT's design rule/electrical rule checker and switch simulator [BAKE80].

However, once the tools were acquired, we encountered immediate problems. CAESAR is designed for an AED 512 Color Graphics terminal with eight bit planes. We had an AED 512 with six planes available to us, but it is owned by the Computer

Center, and not our department. Thus it was not feasible to upgrade it with more memory planes. The terminal was located in another end of the building, with no provision for the video terminal needed by CAESAR for textual input/output (it was never clear to us why a separate terminal for text interaction was needed in the first place). Further, we felt that we could obtain a better color graphics terminal for roughly the same price as the AED. Instead of acquiring our own AED 512 to maintain compatibility with CAESAR, we opted for a Chromatics 7900 with a user programmable 68000 in the terminal and a higher resolution screen (768 x 1024 pixels instead of 512 x 512). Of course, we were forced to transport CAESAR to this new hardware.

The MIT tools were not trouble free either. A design specified in the Caltech Intermediate Form (CIF) first has to be converted to a form that is acceptable to these programs. Although CIF is meant to be a standard language, in fact different tool sets use different escapes in CIF to communicate important information. For example, CAESAR specifies the location of a label via a 94 escape, while the MIT tools expect a 0 escape. Integrating tool sets is remarkably difficult. Further, even though the MIT DRC reported violations in terms of lambda coordinates, it was difficult to correlate these error reports with features within the circuit. A designer was forced to quit the editor and convert the design into the internal form accepted by the DRC, before the errors could be found. This broke the interactive design cycle. There was a compelling need to combine the DRC function with the

circuit editor. (Note: the DRC has subsequently been integrated with the Berkeley version of CAESAR [OUST82]).

In a graduate seminar led by Randy Katz, two (competing) groups of students (Joe Moran and Kishore Ramachandran; Dan Schuh) undertook to make the tools work in our computing environment, and to integrate layout and design rule checking. Dan Schuh took the project one step further by implementing the design rule check within the terminal itself. These two efforts resulted in two systems: Caesar ("Big Caesar"), which implements a terminal independent version of CAESAR, and caesar ("Little Caesar") which is a version of CAESAR that takes advantage of the capabilities of an intelligent workstation.

The remainder of the paper is organized as follows. In section 2, we briefly describe CAESAR and the MIT DRC programs. Section 3 details our experience with making a terminal independent CAESAR, and section 4 describes our experiments with migrating functions into a workstation. Section 5 contains our conclusions.

2. CAESAR and DRC

In this section we briefly review the capabilities of these two systems. Our purpose is to set the context for our work. More detailed descriptions of these systems are found in [OUST81] and [BAKE80].

2.1. CAESAR: An Interactive Editor for VLSI Layouts

CAESAR allows a designer to interactively edit layout geometries. It provides a two dimensional cursor, with the ability to "paint" under the cursor. CAESAR does not support such CIF primitives as polygons and wires. A flexible set of commands make it possible to select an area of a layout under the cursor, and to move it, delete it, or replicate it any number of times. Only Manhattan features are supported. CAESAR supports the notion of a cell and allows the user to build a design hierarchy. It is possible to obtain a CIF representation of a design for input to other tools.

The system, as it is distributed from Berkeley, is configured for a video terminal at which commands are typed, and an AED 512 Graphics Terminal for graphical interaction.

2.2. Tools for Verifying Integrated Circuit Designs

The MIT verification tools include a design rule checker (DRC), electrical rule checker (ERC), and several switch level simulators. We were primarily concerned with the DRC. The checker works by converting a CIF description of a design into a rasterized image, based on a lambda grid. Templates are passed over the rasterization to check for violations, the position is advanced by one lambda position, and the process is repeated. The checker reports the coordinates of the error.

The major difficulty in using the DRC is that the CIF representation has to be converted into a different internal form understood by these tools, called CED. This involves invoking several mapping programs to put the design into the required format. In addition, it is difficult to correlate the error report of the DRC with circuit features displayed on the graphics terminal.

One of us (Dan Schuh) observed that the representation of a design in a terminal's frame buffer was essentially the same as that used by the MIT DRC, assuming that one lambda square was mapped to one pixel. This was the basis of his implementation of the DRC within the terminal.

3. Big Caesar: A Terminal Independent Caesar (Moran, Ramachandran)

We started with the graphics editor CAESAR and introduced the following changes, modifications, and improvements:

1. terminal independence
2. improved efficiency
3. correction of bugs
4. added graphics commands like wire layout and joystick control
5. integration of design rule checker into graphics editor

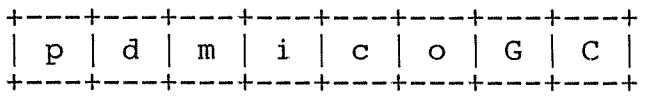
Building terminal independence into graphics tools is important, since it makes it possible to use the existing software with new graphics devices as they become available. We were unwilling to lock ourselves into AED terminals. Integration of the design rule checker and the editor makes it possible for the

user to invoke the check from within the editor and see the output graphically displayed on the terminal, thus avoiding a break in the interactive design cycle.

3.1. Terminal Independence

Big Caesar currently supports two graphics terminals, the AED 512 and the CHROMATICS 7900, and has the ability to work in either two terminal or single terminal mode of operation. The terminal handling software has been modularized to allow for easy adaptation to new terminal types.

The original CAESAR was designed for an AED terminal with eight bit planes. The mapping of layers to bit planes is shown in figure 1. One of our goals was to make CAESAR independent of the number of planes in the terminal. A first step in making CAESAR work for terminals with fewer bit planes was to move the



p - polysilicon
d - diffusion
m - metal
i - implant
c - contact cut
o - overglass
G - Grid, Bounding Box, and Labels
C - Cursor

Figure 1 - Original Bit Plane Mapping

important layers, such as the cursor, to the lower numbered planes. Big Caesar automatically maps a normally solid box of a non-existent higher bit plane into a hollow box on a lower plane. The remapping of planes is shown in figure 2. Note that the less interesting layers, namely implant and overglass, are assigned to the higher planes (which may not be present). For a terminal with fewer than eight planes, boxes on bit plane seven are mapped to hollow boxes in bit plane n-1, boxes on bit plane six are mapped to hollow boxes in bit plane n-2, etc., where n is the number of planes available. CAESAR in single terminal mode maps the text to the (C)ursor layer, or can have text directed to a special layer supported by the terminal (the CHROMATICS has a separate "overlay" layer for text).

All bit plane dependencies were removed from the original CAESAR, although CIF output is still produced in the original order. The code was also modified to make it possible to write text to the graphics device. The graphics package within CAESAR was extensively restructured to enhance terminal independence. An internal table describes the capabilities of each terminal type

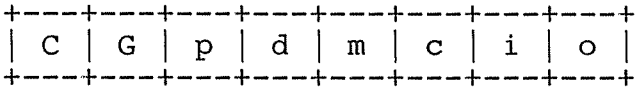


Figure 2 - New Bit Plane Mapping

supported by Big Caesar. Information included in this table is the terminal's name, color map, and device specific graphics routines. Also stored is information about the screen resolution and number of bit planes. Adding a new terminal type is a matter of making the appropriate entries in the terminal table, and providing the routines for the graphics primitives.

3.2. DRC Integration

The long command ":CHeck" invokes the design rule checker. The section of the design visible under the CAESAR cursor is passed to the design rule checker. The resulting error codes are displayed on the screen where they occur within the design. The long command ":ERRor err_num" prints out a textual explanation of the error on the screen.

When the DRC is invoked, the rectangles under the cursor are written to a file scaled up to quarter lambda coordinates (this corrects certain roundoff errors in our original version CAESAR). The file is sorted by maximum y coordinate and converted to the CED format expected by the DRC program. The file of sorted CED rectangles are read in and checked by the DRC. It announces an error by calling a routine to display an error code at the x and y coordinates where the error was detected.

3.3. Additional Enhancements

For the AED and CHROMATICS, the joystick may be used to move the CAESAR cursor with the additional short commands "x" (for

lower left) and "r" (for upper right). Necessity is the mother of invention, and these commands were added because we do not have a graphics tablet available to us.

New wire commands have also been added to aid in routing wires, a feature strongly suggested by users of our Caesar. The long command ":Wire layer width" is used to set the wire layer and width. The short command "b" is used to (b)egin a wire at the current joystick position. After moving the joystick, the short command "h" draws a (h)orizontal wire from the last point to the current joystick position. If "H" is used, then the wire is drawn in two segments if necessary to join the two points, drawing the (H)orizontal wire first. The short command "u" draws an (u)p-down wire, while "U" draws a two segment wire, with the (U)p-down segment being drawn first. After an "h", "H", "u", or "U" command, the joystick position becomes the new starting point for the next wire, thus making it easy to continue laying out the same wire over a longer path (see figure 3).

Additional changes were made to enhance the efficiency of CAESAR. Storage is allocated in large chunks, and managed internally, rather than calling the operating system to allocate storage for each new rectangle created by CAESAR. Further, the graphics primitives were structured in such a way that the state of the terminal is remembered between operations, reducing the amount of information that has be communicated to the terminal.

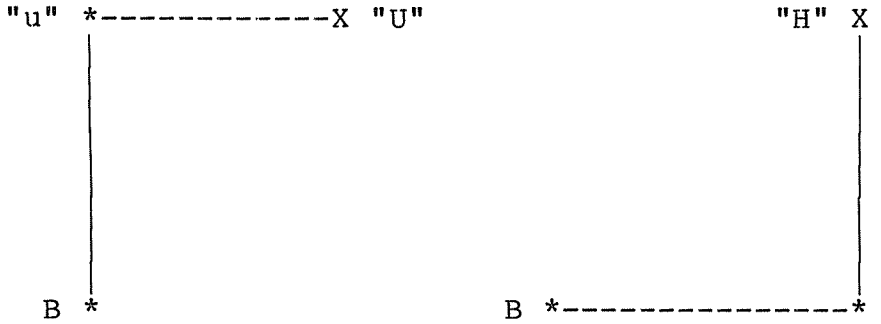


Figure 3 -- Wire Routing Commands
(B = Begin point; X = Current Cursor Position)

4. Little Caesar: An Intelligent Workstation Caesar (Schuh)

Many of the enhancements to CAESAR described above were also included in Little Caesar. The major difference was that this Caesar achieved terminal independence by making extensive use of an existing terminal independent graphics package called Spider-Graphics [HANR82]. The advantage is that it is possible to bring the editor up on new terminals very rapidly, as long as the primitives have already been implemented for the desired device. For example, CAESAR was modified to drive a pen plotter by recompiling the system with the graphics package for the plotter. The disadvantage is that there is one executable copy of the editor for each terminal type.

Our experiments with CAESAR indicate that a major bottleneck is the flow of characters to the graphics terminal. The size of this stream can be greatly reduced if the program keeps track of the state of the terminal between interactions.

The CHROMATICS 7900 terminal has a resident 68000 microprocessor that is user programmable. While it is not easy to download programs into the terminal (because of incomplete documentation and bugs in our version of the terminal), we have been able to surmount the problems. For example, we were able to write a new routine for drawing filled rectangles that outperformed the version supplied with the terminal. Migrating other graphics functions into the terminal also reduces the amount of information that has to be communicated to the device, thus improving the terminal's ability to change the image on its screen.

A major effort was dedicated to making the MIT DRC program run in the terminal. DRC is well-suited for local execution because its input is similar to the format of graphics display data in the terminal's frame buffer. By properly scaling the graphics display, it is possible to make pixels in the frame buffer have a 1-to-1 correspondence with a one lambda grid. The DRC program can obtain its data directly from the frame buffer, without requiring any interaction with the mainframe.

The original program obtains a line by line bitmap rasterization of the layout by examining layout rectangles sorted in raster order. We can obtain the data by simply looking at the corresponding pixels in the frame buffer. Design rule checks are done on a window by window basis, as before. Errors are flagged by highlighting the grid and cursor layers in windows where they are found. To make the error visible, we have prepared a special color map in which the grid and cursor layout appear

"translucent". This is implemented by converting the color map of the first six layers to a <hue, value, saturation> representation. Pixels under the highlighted grid and cursor layers have their value and saturation lowered, but the hue is maintained. This allows the paint to be seen through the solid rectangles of the cursor and grid layers.

Unfortunately, all errors are highlighted in the same way, making it difficult to detect which error occurred. It is impractical to type a message because of the small number of pixels associated with the error (i.e., a 4 pixel by 4 pixel window). We rewrote the DRC to do one check over the whole design at a time, rather than all checks within a window at a time. A different method of extracting the rasterization was employed to reduce the performance penalty. Instead of constructing 4 x 4 and 3 x 3 windows for all layers bit by bit, 4 x 32 bit windows for the critical layers in a given check were extracted into register variables. The frame buffer memory is structured so this can be done efficiently. This allows a rapid scan through most windows, since the next window can usually be obtained by shifting a few registers. Since some checks require looking at as many as 4 layers, not all of the rasterization fits in registers. However, an initial screening can be done on the basis of one or two layers.

The user interface has two parts. Caesar can execute the drc by sending a sequence of characters to the terminal to invoke the program, and describing the area of the frame buffer and the

kinds of rules to check. The drc is invoked by the command ":drc", and the area checked is approximately the area under the cursor. Currently, all checks are made, but it would be simple to restrict the checks to a requested subset.

The second part of the interface is the display of error messages. If a violation is found while checking a particular error type, the program waits for a character from the keyboard. Four actions are possible, depending on the character: the sequence of checks is continued, either with the (1) cursor layer, (2) grid and cursor layer, or (3) no layers erased, or (4) the drc is aborted.

We have found that on our heavily loaded mainframe, the drc can run many times faster when performed in the terminal. However, we have not conducted extensive performance tests at this time.

5. Conclusions

The ability to obtain university software from other institutions is a mixed blessing. While it does not cost much (if anything), it may not be suited to a user's environment. Further, the original authors are not interested in supporting a user community. To take maximum advantage of the opportunity, the recipient should be prepared to modify the code to make it work effectively in his environment. Often this can have a synergistic effect. Since the "not invented here" syndrome does not apply in this situation, tools from different places can be combined

within an integrated system. Very often the system is more powerful than its individual parts. By taking advantage of new hardware, such as a more powerful terminal, an existing tool can be made even more effective. Users should not be afraid to modify the tools they receive.

Our experience with migrating the design rule check into the terminal is a prototype for the design environment of the future. Many design tasks will be offloaded into the terminal. The result will be a shorter modify-test cycle, with the ultimate result being lowered costs for circuit design.

6. Acknowledgements

First and foremost, we wish to thank the authors of these programs, John Ousterhout and Clark Baker, for making the source code available to us. Our colleague, Don Neuhengen, helped in getting the MIT software running, and also discovered the problems in making CAESAR run on "our" AED terminal. The chip designers in our seminar course were responsible for "shaking out the bugs" and for suggesting several features, such as the wire commands, which were incorporated into our CAESARs.

7. References

[BAKE80] Baker, C. M., C. Terman, "Tools for Verifying Integrated Circuit Designs," Lambda, Fourth Quarter 1980.

[CONW80] Conway, L., "VLSI Design in the Universities," in University Scene, Lambda, Fourth Quarter 1980.

[HANR82] Hanrahan, P., "SpiderGraphics User Manual," Internal Memo, Computer Sciences Department, University of

Wisconsin-Madison, 1982.

[OUST81] Ousterhout, J. K., "Caesar: An Interactive Editor for VLSI Layout," VLSI Design, 4th Quarter 1981.

[OUST82] Ousterhout, J. K., personal communication, 1982.