

An Integrated UNIX-Based CAD System for the Design and Testing of Custom VLSI Chips

L. J. Deutsch

Communications Systems Research Section

This article describes a computer-aided design (CAD) system that is being used at the Jet Propulsion Laboratory for the design of custom and semicustom very large scale integrated (VLSI) chips. The system consists of a Digital Equipment Corporation VAX computer with the UNIX operating system and a collection of software tools for the layout, simulation, and verification of microcircuits. Most of these tools were written by the academic community and are, therefore, available to JPL at little or no cost. Some small pieces of software have been written in-house in order to make all the tools interact with each other with a minimal amount of effort on the part of the designer.

I. Introduction

The design of VLSI chips would be impractical without the use of CAD tools. This is because the typical VLSI chip that is designed at JPL currently comprises at least 5,000 transistors and in some cases this number is as large as 50,000. In addition to this, each of the six to ten mask layers that correspond to steps in the fabrication process must be defined for these chips. In order to facilitate the design of VLSI chips on a computer, a standard file format for the description of these mask layers was developed at the California Institute of Technology. This description is called the "Caltech intermediate form" (CIF) (Ref. 1). Many chip fabricators will now accept a computer file in CIF format and implement the design on a silicon wafer. In fact, CIF has recently become an industrial standard as well as a university standard for chip description.

The primary purpose of a VLSI CAD system is to allow a designer to specify a chip in some convenient fashion and to produce a file in CIF format that may be sent to a fabrication service. The process of defining the CIF file is called "layout." In addition to layout, there are secondary functions that a

good CAD system should perform. One of these is design verification. This consists of an automated check, performed on the chip description, against a set of rules that are fabrication-dependent. These rules might include both "design rules" and "electrical rules." Design rules refer to geometrical constraints on the masks themselves and arise from tolerances and material limitations in the fabrication process. Electrical rules refer to the interconnection of transistors. One example of an electrical rule would be a limitation on device fan-out.

A good CAD system should also contain simulators that allow the designer to "run" a chip in software and hence verify that it performs the desired function. Simulators can take many different forms from low-level programs that model only the digital switching of a transistor, to high-level programs that use very complex models of the transistors. Since the execution time of a simulator depends on its level, several different levels of simulation should be present on the CAD system.

Tools should also be present that allow the same tests that are performed on the chip design with the simulators to be

used on the finished chip itself. This involves a hardware interface from the CAD system computer to a chip testing device.

In addition to the above tools, a complete system will include programs for creating commonly-used subcircuits with a minimum amount of effort. Such subcircuits might include input and output pad drivers, programmable logic arrays, read-only memory (ROM), and simple logic gates.

Much CAD tool development has already been done by universities in support of their own VLSI designer training efforts. This software development is sponsored in part by the Defense Advanced Research Program Agency (DARPA). DARPA also supports the MOSIS (Ref. 2) chip fabrication service that is run by the Information Sciences Institute of the University of Southern California. Because of JPL's standing as a research facility and its involvement with certain DARPA contracts, the university tools and MOSIS fabrication are available to users here at no cost. The CAD tools are written to run on Digital Equipment Corporation VAX computers with the UNIX (Ref. 3) operating system.

A CAD system has been assembled using these university tools as part of the DSN Advanced Systems Program. A VAX 11/750 computer was purchased along with an inexpensive binary license for UNIX (A binary UNIX contains no source code and hence no ability to modify system software). Software was obtained from many sources including Caltech, MIT, the University of California at Berkeley (UC Berkeley), Carnegie-Melon University, and the University of Washington. Some additional pieces of software were written in-house so that the various tools can communicate with each other. This CAD system is used to support both the design of custom and semicustom chips and for the evaluation of new CAD tools as they become available. Some of the chips that have been designed on this system include a multicode convolutional encoder (Ref. 4), a Reed-Solomon encoder (Ref. 5), a syndrome generator for a Reed-Solomon decoder (Ref. 6), a Fermat number multiplier (Ref. 7), and a Massey-Omura multiplier (Ref. 8).

In the following sections, the various software tools in the system will be examined in more detail. They will be grouped according to their function. An overall block diagram of the software CAD system is shown in Fig. 1. Several file formats are used to represent data on the system. One is the CIF format mentioned above. In UNIX, the format of the file is indicated in its name as a string of characters at the end of the name preceded by a period. Hence the computer file "file.cif" would be in CIF format. Other formats used on the system include ".ca" or "caesar format" for a layout graphics file and ".sim" for a file containing wirelist data. These conventions will be used in the remainder of this article.

II. Layout

The primary layout tool on the CAD system is a program called "caesar" (Ref. 9). It was written at the University of California at Berkeley. Caesar requires the use of two computer terminals. One terminal may be one of a variety of high resolution (at least 512×512 pixels) color terminals. An Advanced Electronics Design AED-512 terminal is used on this CAD system. The second terminal is a regular CRT-type such as the DEC VT100. In addition to the terminals, caesar may optionally employ a digitizing tablet to help speed up data entry. All commands are entered either from the tablet or the text terminal. The graphics terminal is used only to display the portion of the chip design that is being edited.

Caesar is basically a powerful graphics editor. It allows the user to place, move, modify, copy, and delete colored rectangles on the graphics screen. These colored rectangles form a pictorial representation of the various mask layers in the designer's chip. Caesar allows the editing of arbitrarily large designs by including the ability to zoom in and out of the design and to pan across it. The user can save any part of the design being edited in a file on the computer disk. This file is in .ca format. These .ca files can be called back to the graphics screen to create large hierarchical designs. In addition, the designer may place textual labels on the masks so that they may be referred to in verification and simulation software. Figure 2 is a photograph of the caesar graphics display. It exhibits the hierarchical nature of the caesar program.

The digitizing tablet greatly increases the speed at which designs may be accomplished with caesar. There are four buttons on the tablets "cursor" (a device similar to the "mouse" on many personal computer systems). Two of these are used to position the lower left and upper right corners of a white box on the graphics screen. A third is used to fill in that box with a color. The fourth is used to locate subcells in the chip design and to descend into the hierarchy for editing purposes.

Caesar has been used by various universities to design successful chips with as many as 100,000 transistors. There is no theoretical limit to the size of a chip that may be designed. Caesar is also "technology independent." This means that it may be used to design chips for fabrication in any VLSI process (e.g., NMOS, CMOS, and GaAs). Caesar accomplishes this by reading data files for each of these technologies. New data files may be easily created by the designer as new fabrication processes become available. This means that caesar will never be obsolete. In fact, two additional technologies have been added to caesar here at JPL. These are the Sandia National Laboratory's CMOS process and a GaAs process that is compatible with the Rockwell fabrication line.

Caesar produces a .cif file as output. This file may be sent directly to fabrication, or subsequent software testing may be performed on it. In addition, an in-house program called "ca2cif" has been written to convert .ca files to .cif format without having to run caesar. This means that the user need not have access to a high-resolution graphics terminal to perform this function.

Caesar has one important limitation as a layout tool. Because caesar creates all designs out of rectangular elements that are aligned with the axes of the graphics display, chips may not contain any curved or slanted geometry. Such chips are said to be of "Manhattan geometry." Chips that are designed with curves and bends in them are considered to have "Boston geometry."

Any .cif file that represents a Manhattan design may be edited with caesar. This is accomplished by running the program "cif2ca." Cif2ca produces a .ca file that corresponds to the original .cif file.

There are many programs available for creating hardcopy plots of chip designs from a .cif file. Such plots are invaluable in checking long line interconnections on large chips. Two programs have been evaluated on this CAD system. One of these, "mcp" from UC Berkeley, is used to create color plots on a Trilog color printer/plotter. This device is a slow dot matrix printer that uses a four-color ribbon. It takes about 20 minutes to produce one page of a low-resolution plot with mcp. Also, since the "m" in mcp stands for Manhattan, mcp can plot only Manhattan designs. The second program is called "cifp" and comes from Caltech. This program drives a Hewlett Packard eight-color pen plotter on the JPL CAD system (it can also drive other devices). Cifp is capable of plotting Boston geometry chips.

III. Design Rule Checking

Two programs that perform design rule checking have been evaluated on the CAD system. The first is a program from Carnegie Mellon University called "drc." This program works only on NMOS chips and so it has been replaced by the second program, "lyra" (Ref. 10), from UC Berkeley.

Lyra checks a file in .ca format against a set of rules that describe the limitations of a particular fabrication process. These rules describe, for example, the minimum sizes of transistors and interconnections, the minimum distances allowed between various structures, and the allowed dimensions of via cuts between interconnection layers. Notice that these are all rules that pertain only to the geometric properties of the design. They have nothing to do with the functional behavior of the circuit that is represented by this geometry.

Lyra rules for new technologies may be written by the user. Hence lyra is technology independent. A set of rules for the Sandia CMOS process have been written here at JPL. The program itself is an expert system that checks the .ca file against the set of rules. Any errors are recorded in a new .ca file that includes the old one as a subcell. When this new file is examined using caesar, the design errors appear as black rectangles that are labeled with the appropriate error message. Alternatively, lyra may be run directly from caesar. In this mode, the tablet cursor is used to position caesar's white box around the area to be checked. A command is then typed from the text terminal and, after a short delay, the errors are displayed on the graphics screen. Figure 3 shows a photograph of a caesar graphics screen that has been checked with lyra.

Lyra, like caesar, runs on chips with Manhattan geometry only. This allows lyra to execute very quickly — as much as an order of magnitude faster than more general design rule checkers. Because of this constraint, lyra needs only to check the rules at the corners of the colored rectangles.

IV. Extraction

In order to perform either electrical rule checking or simulation on a chip design, a functional description of the chip must be derived from the .cif file. This is done by a program called an "extractor." On this CAD system, the extractor program is called "mextra" and it comes from UC Berkeley. The "m" in mextra stands for Manhattan so mextra operates under the same constraint as caesar and lyra. Mextra is not technology independent, though. It can only extract NMOS and CMOS chips. The ability to switch easily between NMOS and CMOS extraction was added here at JPL. The output of mextra is a file in .sim format. Each line of this file describes a device (e.g., transistor or capacitor) and how it is interconnected with other devices in the design. Mextra labels all the electrically different nodes of the design. This is done arbitrarily except for those that have been previously labeled with caesar. These retain their previous labels.

Since many of the tools that use .sim files as input report errors by node number, there is a program called "mexnodes" from the University of Washington that allows the designer to see all the node numbers using caesar.

V. Electrical Rule Checking

The software for electrical rule checking on the CAD system is called "erc" and it comes from the Boeing Corporation. It works only on NMOS chip designs. The rules that it checks are not easily modified. This is not as serious a problem as one might expect since electrical rules do not vary

much between different NMOS fabrication processes. Also, CMOS circuits are far more robust with respect to electrical rules than NMOS circuits. It would be preferable, however, to have an electrical rule checker with the same kind of flexibility as the lyra design rule checker.

The erc program needs both a .cif file and a .sim file for the design that is to be checked. In addition, the user can supply a third file that locates the inputs and outputs to the design. The presence of this last file can prevent many erroneous errors. There are many options that may be used when running erc. It is possible to have erc search for only selected rule violations in this way. Erc produces two outputs. The first is a text file that explains each error that it has found. The second is a .cif file that contains the original .cif input file with added labels that indicate the errors. This latter file may be examined by running cif2ca and looking at the result with caesar.

The erc program has been helpful in detecting errors on ratioed NMOS logic. These errors, if not detected by erc, could have been found only by running a very-high-level simulation program or by testing the finished chip.

VI. Simulation

There are many simulation programs on the CAD system. This reflects the fact that careful simulation of chip designs in software can lead to chips that work on the first fabrication iteration. The time that is invested in a thorough software simulation is more than made up for in reductions in chip testing and redesign time.

There are five simulation programs that are currently used by chip designers on the CAD system. In addition, several others have been evaluated. These programs vary in the detail with which they perform their function.

The lowest level simulator on the system is called "esim." Esim is an "event-driven" simulator that was written at MIT (Ref. 11). The version that is installed here can simulate only NMOS circuits although a new version exists (modified at UC Berkeley) that can perform CMOS simulations as well. Esim takes a .sim file for the design as input. In addition, esim implements a comprehensive editing language for the definition of test vectors and the display of output sequences. Esim models transistors as switches that are either open or closed. Signals in esim may have one of three values at any time: 0, 1, or X. An X corresponds to an unknown logical state. Esim ignores resistors and models capacitors as if they have an infinite charge decay time. For this reason, esim runs at a very high speed. A chip with 3,000 transistors can be tested with thousands of bits of test vectors in less than one minute using esim on a moderately loaded VAX computer.

Because of its limitations, however, esim is only good for checking the logic design of the chip.

A newer simulator by the same author as esim is "rsim" (Ref. 12). Rsim is really a set of software tools that implement a complete simulation system. The first of these is a program called "netlist." Netlist allows the user to define a circuit by specifying interconnections between predefined logic primitives. Netlist could be used, for example, by a logic designer for software simulation of circuits that consist of off-the-shelf, small-scale integrated (SSI) circuits. Netlist is fully interactive and technology independent. Its output is a file in a format called .1 that is used by the rest of the rsim system. A .1 file for a chip design can be extracted from a .sim file by running an rsim program called "presim." Once a .1 file exists, one of two types of simulation may be run. The first is a program called "net." Net performs a simulation similar to esim. The advantage to net is that the user may run a simulation using a description generated by the netlist program without actually having performed a layout of the circuit. In this way, algorithms may be verified even before layout begins. The second simulator in the rsim package is called "rnl." Rnl has a more sophisticated model of devices than does esim. In particular, rnl has models for resistors and capacitors and it differentiates between sizes and types of transistors. It produces a much more precise simulation than rsim but it takes about five to ten times as long to run. In addition, rnl uses the "lisp" programming language (an artificial intelligence language) as a user interface. This makes it rather difficult to use for a first-time designer. Rnl can simulate both NMOS and CMOS devices.

Because of the need to use lisp to communicate with rnl, a user-friendly front-end program called "rsim" was written here at JPL. Rsim takes a file in .rsim format that describes the test vectors that are to be run through the rnl simulator. Rsim translates this file into the lisp instructions that are needed to run rnl. Rsim then runs rnl and produces an output similar to that produced by esim. In addition, a second program called "makesim" was written that allows the designer to use a subset of the esim editor to create .rsim files. These new programs have allowed even first-time users to take advantage of the power of rnl.

The program "spice" (Ref. 13) has been around for a long time. It was written at UC Berkeley and it has had numerous updates and revisions. Version 2G6 of spice is on the CAD system. Spice is the highest level simulator that is on the system. Spice takes an input file that is written in a fortran-like language as input. This file contains both the circuit description and a description of input waveshapes. Spice performs an analog-level simulation that models the entire circuit as a set of linear equations. It is very accurate and

very time consuming. A spice simulation run for a circuit with 200 transistors for 32 clock periods (and a 1-MHz clock frequency) takes about an hour to complete on the VAX 11/750 computer with a floating point arithmetic accelerator board. In addition, spice will very often fail to complete at all if there are more than 500 transistors in the circuit. Spice is, therefore, most applicable to the simulation of small subcircuits within a chip. A spice-compatible input file (.spc format) may be created from a .sim file by running the program "sim2spice" from UC Berkeley. Spice is technology independent since new device models may be added fairly easily (as long as the new device is similar to one in a catalog of standard device types). New device definitions usually consist of calls to existing device models with a set of specialized parameters.

The timing analyzer program "crystal" is a good compromise between the high detail of simulation that spice provides and the high speed of esim. Crystal has models for most VLSI devices, but it does not keep track of signal values within the circuit. Instead, crystal determines the speed at which a signal wavefront propagates within the circuit. Crystal is a very powerful program. It can be used to find critical paths (those paths that have the longest delays) in a circuit. These paths may be displayed with caesar (see Fig. 4.). In addition, spice input files that describe these critical paths can be generated automatically by crystal. In this way, spice can be used to simulate only the most time-critical parts of a design. This drastically reduces simulation time.

In addition to these simulation programs, a program called "slang" from UC Berkeley was also evaluated. The people at Berkeley have simulated chips with more than 100,000 transistors using slang. However, slang is very hard to learn to use. The user interface is cumbersome and the documentation is inadequate. In addition, rsim seems to perform all of the same functions as slang and is easier to use. For these reasons, slang is not used by most of the designers.

VII. High-Level Design Aids

The software described up to this point is all that is needed to design and verify custom VLSI chips. However, the design process can be made much easier with the addition of certain tools called "high-level design aids." These tools generate and assemble common circuit elements from high-level commands that are issued by the designer. They free the designer from the tedium of actually performing a detailed layout of these basic cells and hence speed up the layout process.

One common circuit element is the "programmable logic array" (PLA) (Ref. 1). A PLA consists of a set of AND gates followed by a set of OR gates and it may be used to imple-

ment any combinatorial logic operation. If some of the outputs of the PLA are fed back to its inputs through a set of delay elements, then the result is a "finite state machine" (FSM). Any digital system may be implemented as an FSM. For the remainder of this report, the term PLA will be used to represent both PLAs and FSMs. Despite the apparent power of the PLAs, they are not used for all possible functions because they are inherently large and slow compared to fully custom logic. However, they are extremely useful in parts of a large chip design that are not time critical. There are many programs that have been evaluated on the CAD system for the design of PLA structures. In fact, there is a comprehensive set of tools that address all aspects of their definition and design. All of the PLA tools are from UC Berkeley.

There are several ways that the designer can specify the functionality of a PLA. There is a .pla file format on the system for defining PLAs with text lines that describe the inputs, outputs, and AND-OR connections, as well as labels and clocking considerations. The designer may elect to design a PLA by simply writing such a file with a text editor. In addition, the designer can define the PLA in terms of an equivalent set of Boolean equations. This is done by running the program "eqntott." Eqntott translates the equations into a .pla file. The designer may also describe the PLA with a state diagram. This is done by running "peg" (Ref. 14). Peg also creates a .pla file. The .pla file can be optimized to some extent by running the program "presto." Presto eliminates redundant logical elements and rearranges intermediate products in an attempt to minimize the number of gates in the PLA.

Once a .pla file exists, then a PLA layout may be generated. The program "mkfsm" converts a .pla file into a .cif file that contains a layout of the PLA. Mkfsm, however, will produce only an NMOS layout and, in fact, this layout is incompatible with current MOSIS processes. Mkfsm was modified here at JPL to resolve the process incompatibilities and it has been used in several successful chip projects. It has been replaced recently with a new program called "tpla." Tpla is a PLA layout generator that is written with the "tpack" cell assembly package that will be described below. It is technology independent and new technologies may be easily added by the user. Tpla produces a .ca file as output. Tpla has some additional features that make it a very good PLA generator. Tpla minimizes the capacitance within the PLA (and hence maximizes the speed of the PLA) by reducing the number of interconnections that are made with high-capacitance materials. It also calculates the power consumption of the PLA and generates power distribution lines that are the appropriate size to deliver this power. The outputs of the PLA may be optionally placed on the same or opposite side as the inputs. Figure 5 shows a layout of a PLA that was designed

in less than 5 minutes using eqtott and tpla. It implements a 4-bit binary counter with reset.

The subroutine package tpack that was mentioned above is a set of programs that may be called from a user-written C program. They are used to place, stretch, and interconnect predefined cells that are placed in a template file. This is a very powerful tool for a designer who is also a good C programmer. Once basic cells have been designed, they may be assembled into larger functional blocks (or even into complete chips) under software control. Programs that are written using tpack are actually silicon compilers (Ref. 15). One example is the tpla program.

Another high-level tool is the UC Berkeley program called "quilt." Quilt is used to assemble arrays of rectangular circuit elements. Quilt has been used in almost every chip design that has been performed on the CAD system. Many high-level design aides have been written that are basically sophisticated calls to the quilt program. One of these is the UC Berkeley program "vlsifont" which allows the designer to place macroscopic identifying text on a chip using any of the more than one hundred fonts in the UNIX font catalog. Another program that uses quilt is called "pads." Pads was written here at JPL and it allows the designer to define a ring of input and output pads to place around a chip. Pads is technology independent and templates have been written for three magnifications of NMOS and two of CMOS.

The program "rom," which was written here at JPL, allows the designer to define a read-only memory block by creating a text file (in .rom format) that describes the desired contents of the memory. Rom processes the .rom file and runs tpla to generate the layout. Rom is, therefore, technology independent.

One last high-level design aid that was written here at JPL is called "title." Title, which was written a year before vlsifont, is also used to generate text on a chip. It is still used because it is at least ten times faster than vlsifont. It has only one font and it can be used only for NMOS and CMOS chips.

VIII. Standard Cell Libraries

Once a subcircuit, or cell, has been designed, fabricated, and tested successfully, there is no need to redesign it each time it is needed in subsequent chip projects. For this reason, libraries have been set up on the CAD system, so that all designers can access .ca files for these cells. There are separate libraries for NMOS, MOSIS CMOS, and SANDIA CMOS on the system at this time. Each library is write protected so that only the system manager can add to or edit the cells. Each

designer, however, has read access to the libraries. In fact, when caesar is run, a pointer to the appropriate technology library for the current design is generated. This is done through the use of a caesar configuration file in each designer's directory.

The use of predesigned cells can reduce the time needed to complete a complex chip design by orders of magnitude.

Figure 6 shows two standard cells from the SANDIA CMOS library. The cells in this library (as well as the MOSIS CMOS library) each have the same height, the same power distribution bus locations, and inputs and outputs that are accessible from both the top and bottom of the cell. These cells are designed to be assembled into rows with intercell routing performed in the channels between the rows. This type of design is often referred to as "standard cell" design and it is an example of semicustom chip layout.

IX. Testing

Hardware and software are currently under development here at JPL to allow the designer to stimulate a chip with the same test vectors that were used for the software simulation described in Section V. These will be described in a subsequent issue of the *TDA Progress Report*. Currently, testing must be performed with standard test equipment such as signal generators, oscilloscopes, and logic analyzers. The new test station consists of a microprobe station for generating and acquiring signals directly on the fabricated wafer and a digital interface for connecting the chip to the CAD system computer. A photograph of the test station appears in Fig. 7. The microprobe equipment is complete and the hardware interface is currently being installed. The software that will drive the tester is being written at this time. When completed, the test station should greatly enhance a designer's ability to qualify chips quickly and easily.

X. Conclusion

The UNIX-based VLSI CAD station described in this report is a complete, integrated system for the design and verification of custom and semicustom VLSI chips. It has been used in the design of several successful chip projects in both NMOS and CMOS. In fact, most of the chips that are designed on the system work with the first fabrication. The University CAD tools have proven to be robust and adaptable to the needs of many JPL projects. A minimal amount of in-house software was needed to integrate these tools. Also, the establishment of standard cell libraries has reduced the design time needed for new chips substantially.

Since most of the tools are technology independent, the CAD system will be useful far into the future. Some improvements will be required, however. A technology independent extractor will be needed. Also, software for automatic routing

of interconnect signals would further reduce design time. Both these problems are currently being addressed at UC Berkeley. In fact, a new version of caesar called "magic" includes place and route functions; it will be released this year.

References

1. Mead, C. and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Menlo Park, California, 1980 pp. 115-127.
2. The MOSIS Project, *The MOSIS System (What It is and How to Use It)*, Publication ISI/TM-84-128 Information Sciences Institute, Marina del Rey, California, 1984.
3. Bourne, S. R., *The UNIX System*, Addison-Wesley Publishing Company, Menlo Park, California, 1983.
4. Deutsch, L. J., "A VLSI Implementation of a Multicode Convolutional Encoder," *TDA Progress Report 42-72*, Jet Propulsion Laboratory, Pasadena, California, February 15, 1983, pp. 61-69.
5. Truong, T. K., Deutsch, L. J., Reed, I. S., Hsu, I. S., Wang, K., Yeh, C. S., "The VLSI Design of a Reed-Solomon Encoder Using Berlekamp's Bit Serial Algorithm," *Proceedings of the Third Caltech Conference on VLSI*, California Institute of Technology, Pasadena, California, 1983, pp. 303-330.
6. Shao, H. M., Truong, T. K., Deutsch, L. J., Yuen, J. H., and Reed, I. S., "A Systolic Design of a Pipeline Reed-Solomon Decoder," *TDA Progress Report 42-76*, Jet Propulsion Laboratory, Pasadena, California, February 15, 1984, pp. 99-113.
7. Chang, J. J., Truong, T. K., Reed, I. S., and Hsu, I. S., "VLSI Architectures for the Multiplication of Integers Modulo a Fermat Number," *TDA Progress Report 42-79*, Jet Propulsion Laboratory, Pasadena, California, November 15, 1984, pp. 136-141.
8. Wang, C. C., Truong, T. K., Shao, H. M., Deutsch, L. J., Omura, J. K., and Reed, I. S., "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *TDA Progress Report 42-75*, Jet Propulsion Laboratory, Pasadena, California, November 15, 1983, pp. 52-64.
9. Ousterhout, J., *Editing VLSI Circuits with Caesar*, Computer Science Division, University of California, Berkeley, California, 1983.
10. Arnold, M. H., *Specifying Design Rules for Lyra*, Computer Science Division, University of California, Berkeley, California, 1983.
11. Baker, C. and Terman, C., "Tools for Verifying Integrated Circuit Designs," *Lambda*, Fourth Quarter, 1980.
12. Terman, C. J., "RSIM — A Logic-Level Timing Simulator," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, IEEE Computer Society Press, Silver Spring, Maryland, 1983.

13. Negal, L. W. and Pederson, D. O., "SPICE -- Simulation Program with Integrated Circuit Emphasis," Memorandum No. ERL-M382, Electronics Research Laboratory, University of California, Berkeley, California, April, 1973.
14. Hamachi, G., *Designing Finite State Machines with PEG*, Computer Science Division, University of California, Berkeley, California, 1983.
15. Losleben, P., "Computer Aided Design for VLSI," in Barbe, D. F. (editor), *Very Large Scale Integration: VLSI*, Springer-Verlag, New York, 1980, p. 108.

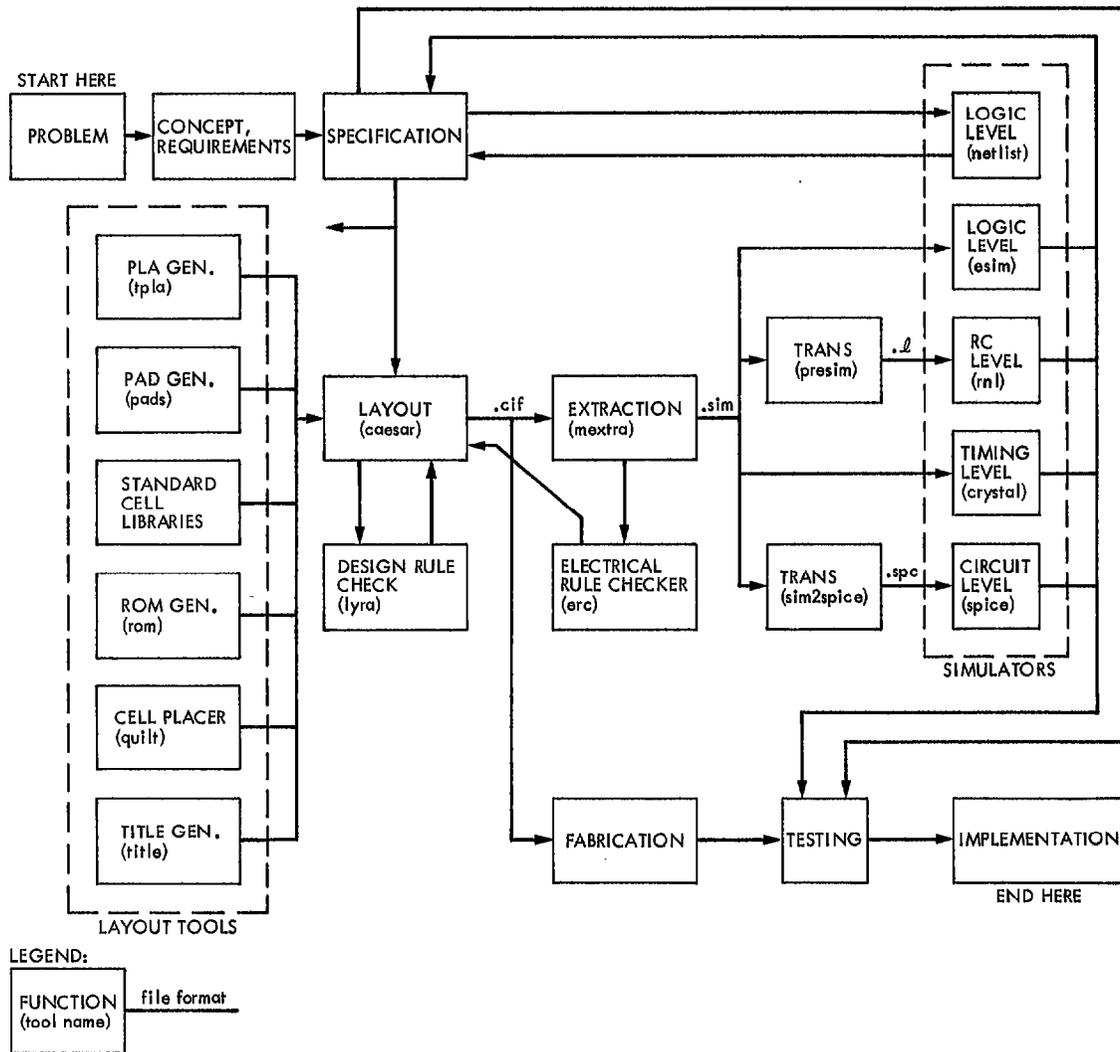


Fig. 1. Integrated UNIX-based VLSI CAD system

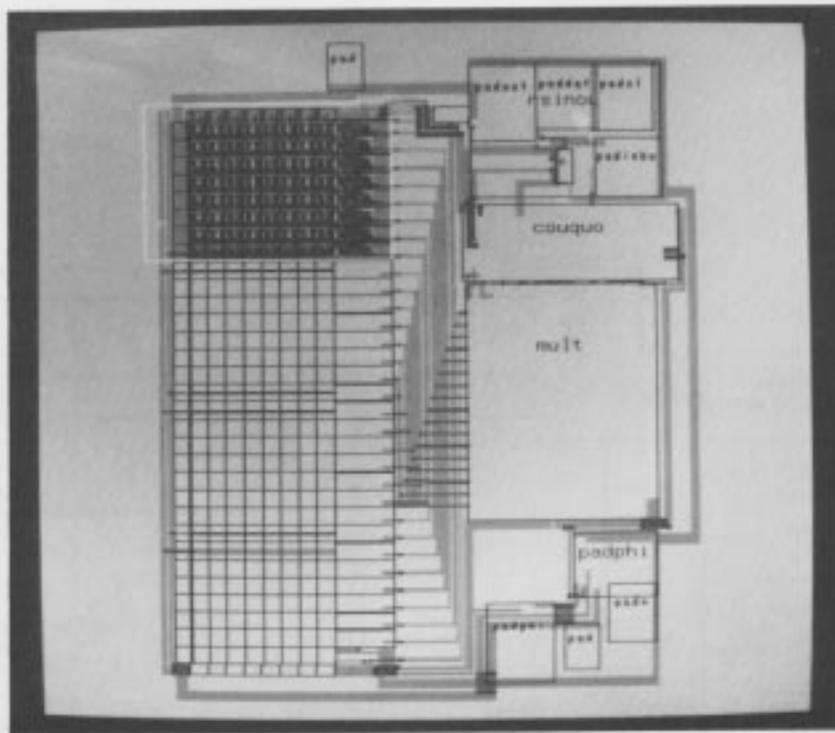


Fig. 2. The caesar graphics display

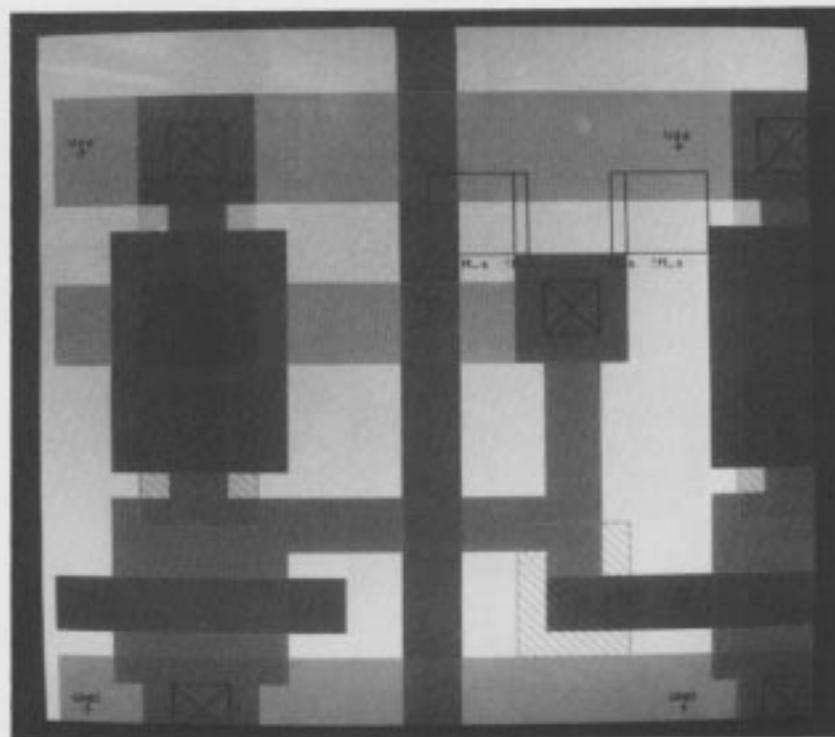


Fig. 3. Design errors located by lyra are displayed interactively with caesar. The black rectangles indicate the magnitude of the violations.

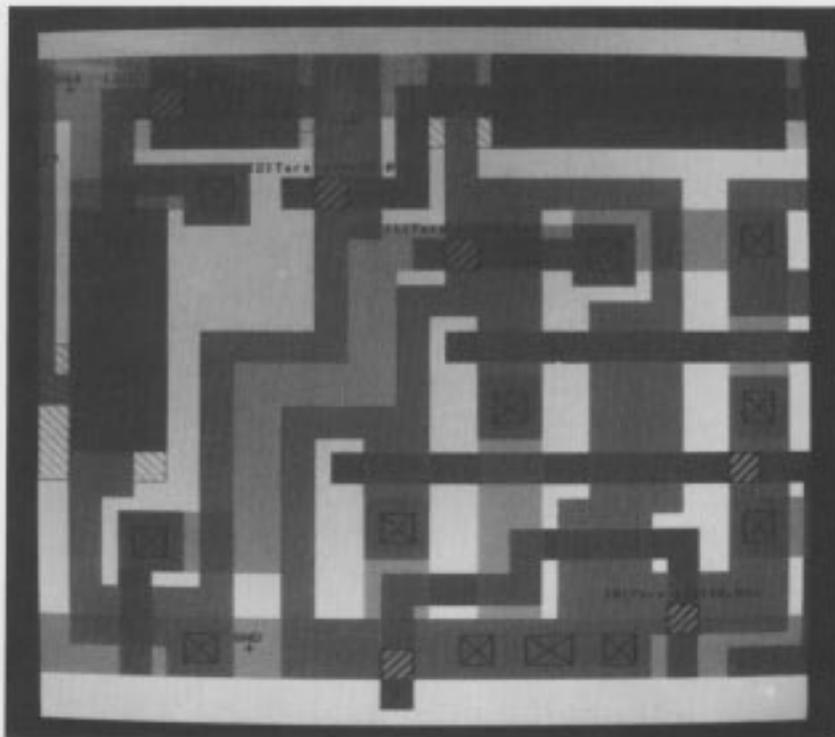


Fig. 4. A caesar display of the output of the crystal timing analyzer

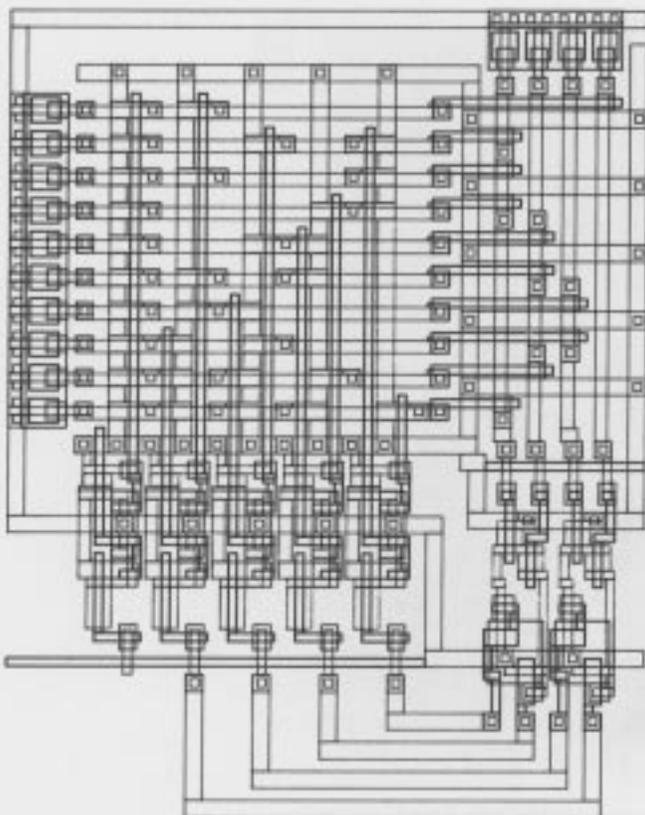


Fig. 5. A 4-bit binary counter with reset implemented as a PLA. The design, accomplished by defining the PLA with Boolean equations, was completed in 5 minutes using eqntott and tpla.

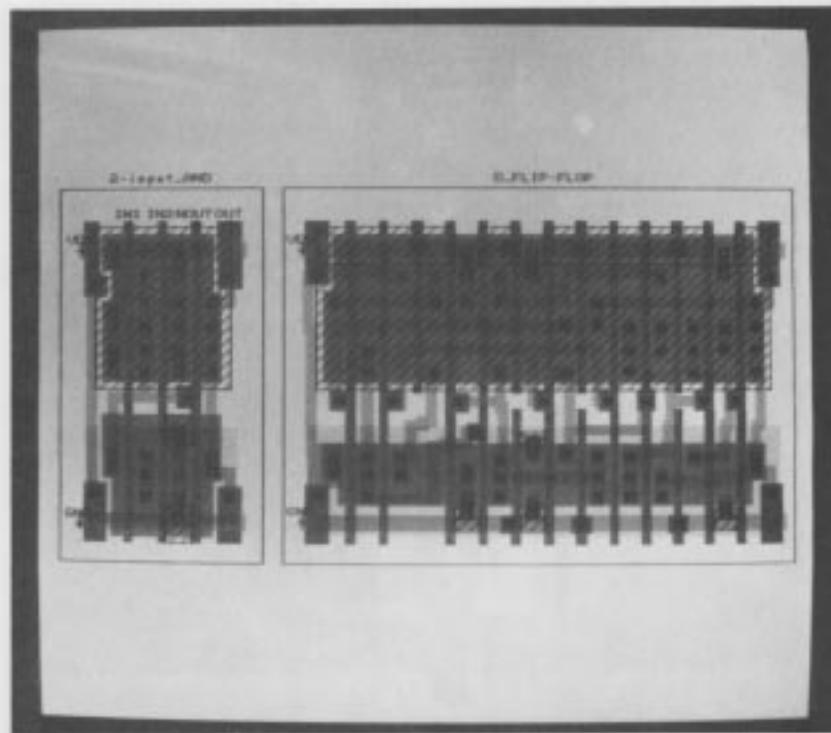


Fig. 6. Two examples of CMOS standard cells



Fig. 7. The testing station for the integrated UNIX-based CAD system