Palo Alto Research Center

A New Approach to Imaging IC Layout and Schematics

Giordano Bruno Beretta

XEROX

A New Approach to Imaging IC Layout and Schematics

Giordano Bruno Beretta

EDL-88-3 April 1988 [P87-00013]

Abstract: Workstation-based design automation systems can be built in two ways: either as a special purpose system or on top of a general purpose environment already available on the workstation. This report assumes that the second option has been selected. In the community that uses design automation tools, imaging is often confined to implementing a set of simple but fast graphic routines. This attitude is responsible for the poor integration of many design automation systems with their underlying platforms. When the information contained in a design database has to be disseminated outside the system to illustrate documentation or to print the contents of the database, design automation systems often fail to provide such services with reasonable comfort.

This report addresses the information dissemination problem by examining system integration from a document processing perspective. Three problem areas are identified: the definition of a user model, independence from imaging models, and the representation of colors.

Solutions to these problems are demonstrated in an implementation called Nectarine. Nectarine processes an image from a VLSI editor which can contain any combination of layout, schematics, and diagrams to produce a device-independent file called a master. The master—which is written in a standard page description language—can be inserted in a text document or sent to a variety of printers.

CR Categories and Subject Descriptors: B.7.2 **[Integrated Circuits]:** Design Aids–*graphics*; D.2.2 **[Software Engineering]:** Tools and Techniques–*user interfaces*; I.3.3 **[Computer Graphics]:** Methodology and Techniques–*device independence*; I.7.2 **[Text Processing]:** Document Preparation.

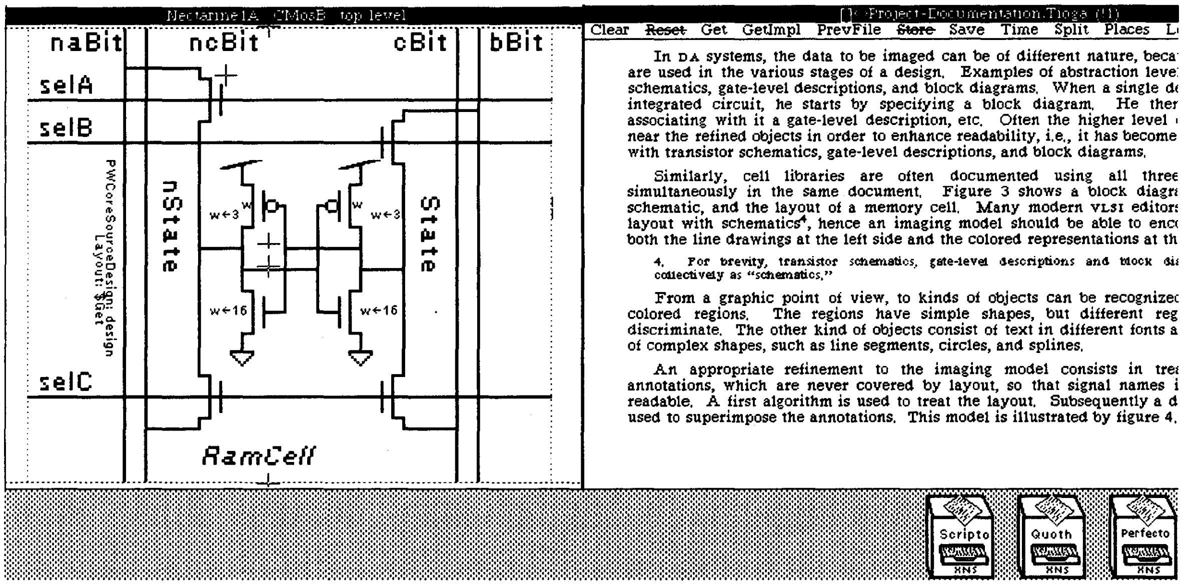
Additional Keywords and Phrases: document processing, documentation, functional colors, IC layout, IC schematics, illustration, imaging, Interpress, printing, system integration.

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

1. Introduction

Electronic publishing has come a long way from paper tape and 80-column dumb terminals to modern full-page systems that show the document exactly as it will appear in printed form. In the realm of VLSI design automation (or DA), many systems are still at the level of emulating what is done on drafting paper. In this report, we show how DA tools can be integrated into existing document processing systems. Figure 1 indicates the problem area: it shows a portion of a screen as seen by designers. Typically, they want to illustrate the text document at the right with the schematic at the left or to send the schematic to one of the printers whose icons appear at the lower right-hand corner of the screen.


naBit
ncBit
cBit
bBit
selA
selB
selC
nState
State
w←3
w←3
w←16
w←16
RamCell
PWCoreSourceDesign: design Layout: $Get
Project-Documentation.Tioga
Clear ~~Reset~~ Get GetImpl PrevFile ~~Store~~ Save Time Split Places L
Scripto
Quoth
Perfecto

In DA systems, the data to be imaged can be of different nature, beca
are used in the various stages of a design. Examples of abstraction leve
schematics, gate-level descriptions, and block diagrams. When a single d
integrated circuit, he starts by specifying a block diagram. He ther
associating with it a gate-level description, etc. Often the higher level
near the refined objects in order to enhance readability, i.e., it has become
with transistor schematics, gate-level descriptions, and block diagrams.

Similarly, cell libraries are often documented using all three
simultaneously in the same document. Figure 3 shows a block diagr
schematic, and the layout of a memory cell. Many modern VLSI editor
layout with schematics[4], hence an imaging model should be able to enc
both the line drawings at the left side and the colored representations at th

4. For brevity, transistor schematics, gate-level descriptions and block di
collectively as "schematics."

From a graphic point of view, to kinds of objects can be recognize
colored regions. The regions have simple shapes, but different reg
discriminate. The other kind of objects consist of text in different fonts a
of complex shapes, such as line segments, circles, and splines.

An appropriate refinement to the imaging model consists in tre
annotations, which are never covered by layout, so that signal names i
readable. A first algorithm is used to treat the layout. Subsequently a d
used to superimpose the annotations. This model is illustrated by figure 4.

Figure 1. Snapshot of a portion of a workstation's screen. On the left is a window displaying a schematic. The objectives are to be able to illustrate the text document in the right window with this schematic, and to send the schematic to any of the printers represented by icons at the lower right-hand corner of the screen.

DA tools started with the goal of being able to edit very large numbers of colored rectangles efficiently. Therefore, the focus of imaging problems in the DA tool community was the implementation of a set of fast graphic routines for displaying data such as schematics or layout on a monitor. Later, as designs became more complex, the emphasis shifted to providing more and more synthesis and analysis tools, leaving the imaging aspect at its primordial state.

System integration has always been a major problem in the area of DA tools. For example, considerable work has been put into attempts to standardize the interfaces between tools, so that at least the tools in a single system can be integrated. The tools, however, are rarely integrated with the underlying system. In particular, VLSI editors tend to use their own imaging models (as defined in section 2), which leads to problems when information has to be disseminated, whether in internal documentation, external publications, or visual aids for meetings. Problems can range from a complete inability to insert an image produced with a VLSI editor into a document produced with a text editor, to the more subtle problem of being able to print an image that remains intelligible on a printer.

Indeed, when it comes to document preparation, only a few systems, such as Intergraph's DP/Publisher [Nobes, 1985], go beyond the creation of checkplots. A *checkplot* is the plot of the layout for a chip. The hardcopy of a checkplot is often produced on a black-and-white plotter using different line styles to draw the outlines of the various layers. It is used to check whether any design rules[1] have been violated. Besides being of little use for electronic publishing, because of the complexity of today's VLSI systems, checkplots have become obsolete and programs are used to verify the layout. This very complexity renders accurate documentation of primary importance for the successful organization and management of a project.

Many VLSI design cell libraries have become so rich that the bandwidth of the screen real estate of even the fastest workstations is too small. Hence, designers like to keep printouts of cell libraries and logic diagrams relevant to their project on their desk or wall.

As powerful general purpose workstations become increasingly available, they are becoming more and more popular for VLSI design systems. Since most the environments already available on these workstations offer sophisticated document processing tools, it makes more sense to improve the integration of a DA system with its platform than to attempt to write from scratch a document processing program for the VLSI system. A *platform* is a scheme of principles on which the components of a system are based. Examples of platforms are operating systems and shells. To determine where integration is needed, the requirements for information dissemination peculiar to VLSI design must be assessed. The following requirements have been identified:

1. Creating documentation must be easy, so that designers will generate it as often as the need arises, instead of working around it.
2. Integration should not hamper portability to other platforms.
3. The solution must be complete, so that all printing needs are satisfied.

The first requirement means that the same user model employed by the existing document processing tools should be used by the DA tools. The user interfaces of many DA systems are still far behind the state of the art found in other fields of computer science.

The second requirement means that the integration must be achieved at a high level, not by espousing unique imaging paradigms chosen by the designers of the platform. In particular, the implementation must be such that it does not assume a given imaging model and it remains compatible with a large number of platforms; portability is thus retained.

1. An example of a design rule is the minimum distance between two wires to avoid a short circuit that could be caused by a misregistration occurring between different lithographic exposures.

The third requirement means that a vast array of printing technologies must be supported. In particular, publication quality must be supported in order to write reports, documentation, and other high quality documents. As in other applications of electronic publishing, the rendition of color is a problem. VLSI editors usually exploit the peculiarities of a specific display device to obtain the shortest painting times on the screen. Colors are determined by using appropriately tailored color maps, a facility that cannot be used effectively on a printer. When the screen is shared with a document editor, it may expect a different color map. A color representation must be found that allows discrimination of the features of a VLSI layout.

Many printing programs currently available in DA systems use paradigms tailored to a given printer technology, i.e., a specific "driver" is written for every printer used. For instance, they may image layout by drawing the contours of wires if they are "targeted" to a pen plotter. Or they may image layout by "BITBLT-ing"[2] patterns or stipples for each rectangle if they are "targeted" to a raster or bitmap device. These printing programs contain many optimizations to handle the large amount of data common in VLSI design while using as little CPU time as possible.

Their major drawback is that they are too specialized and have to be almost fully rewritten for every printer type. A subjective disadvantage is that the output is not necessarily pleasing to the eye, because the stipples or outlines used exhibit a so-called "checkplot look." Only a few programs allow VLSI designers to produce, unaided, color separations of high enough a quality to meet the printing industry standards for documents.

From an implementation point of view, a substantial method to achieve device independence is to use a page description language. It is not wise to store illustrations in the form of bitmaps, because bitmaps depend on the resolution of a device. Also files containing a geometric description of the illustration are not a good choice, because they contain control characters for specific devices and cannot be scaled easily.

Page description languages allow to generate a program that draws the illustration. In order to avoid the necessity of writing drivers for the required devices, an existent language should be used. Furthermore, if a standardized language is chosen, the generated data is persistent. Such a standardized page description language will be described in section 6.1.

The next section presents a suitable imaging model for IC layout and schematics. Section 3 discusses the user interface. Section 4 describes the method used to achieve independence from underlying imaging models. In section 5, a solution is presented to the problem of producing colors that a reader can discriminate, as well as the methods by which this solution is obtained. Before the conclusions, an implementation sustaining the claims made in this report is described.

2. Read "bit-blitting." BITBLT is a common name for the operations performed by the RasterOp procedure described by Newman and Sproull [1979, p. 263].

2. An Imaging Model for IC Layout and Schematics

An *imaging model* describes a conception of how the operations of an imaging system affect the rendering of an image. As mentioned earlier, portability necessitates that no assumptions of a given underlying imaging model be made. Before this can be discussed, an adequate imaging model must be found for the application at hand. First a model suitable for VLSI layout is discussed, then a more sophisticated model is presented, based on the Interpress imaging model [Xerox, 1986], which allows mixing of layout and schematics.

2.1. Imaging Layout

Printing VLSI images[3] in a device-independent manner poses new imaging problems, the most important of which is to render the layout in such a way that enough information is preserved in the hard-copy output. In editors for VLSI design layout, an abstraction of the physical IC layers is commonly represented by using different colors. In conventional imaging, accurate tone rendition is of primary importance, while the information contained in a single fragment of the image is less important. In the rendition of a VLSI image, however, individual tones are irrelevant, since they are simply abstract properties of the physical layers in the VLSI image. At the same time, the IC designer must be able to discern all the layers in the VLSI image, including those contained in overlaps. The polygons representing the various features must be transparent [Trimberger, 1987], while, at the same time, the different intersections must yield tones that are easy to tell apart.

This property of being transparent is different within the scope of VLSI DA tools than it is in the real world. For example, in the editor presented in section 6, the second metal layer is represented using the color "darkish vivid magenta." However, when metal-2 occurs over some material other than a well, the user needs it to be printed in a more transparent shade (in this case, "very dark vivid magenta") to prevent large power or ground wires from hiding the contents of cells over which they are routed.

As this would indicate, one of the most relevant problems in rendering VLSI images is how intersections are treated, since this determines readability. Therefore, the solution is to treat intersections explicitly, instead of trying to twist an existing imaging model to produce the desired effect. The example in the previous paragraph cannot be solved with conventional algorithms used in computer graphics for rendering realistic scenes.

By treating intersections explicitly, I mean that the intersections should be found and treated as new regions, to yield a partition of the plane. This way, the rendition of intersections can be controlled individually and independently of underlying imaging systems. Plate I illustrates some possible imaging models. To image VLSI layout, two problems have to be resolved: all intersections have to be found and a model for coloring the intersections has to be determined. I give a solution to the first

3. In the following, the term "VLSI image" will include the VLSI editor's data structure and its contents.

problem in section 4 and a solution for the second problem with the method by which it is obtained in section 5.

2.2. Refining the Imaging Model

In DA systems, the data to be imaged can be of different kinds; different abstractions are used in the various stages of a design. Examples of abstraction levels are: layout, transistor schematics, gate-level descriptions, and block diagrams. When a single designer is in charge of an integrated circuit, the process usually begins by specifying a block diagram. Each block is then refined by associating a gate-level description with it, etc. Often higher level descriptions are retained near refined objects to enhance readability, i.e., it has become customary to mix layout with transistor schematics, gate-level descriptions, and block diagrams.

Similarly, cell libraries are often documented using all three levels of abstraction in the same document. Plate II shows a block diagram element, a transistor schematic, and the layout of a memory cell. Many modern VLSI editors offer the ability to mix layout with schematics[4]; hence an imaging model should be able to encompass line drawings (shown on the left in plate II) and colored representations (shown on the right).

From a graphic point of view, two classes of objects can be distinguished. One class contains sets of colored regions. These regions have simple shapes, but different regions should be easy to discriminate. The other class consists of text, in different fonts and orientations, and lines, such as line segments, circles, and splines.

An appropriate refinement to the imaging model treats the schematics as annotations which are never covered by layout. In this way, signal names in the layout are always readable. First an algorithm renders the layout, then another algorithm superimposes the annotations. This model is illustrated by plate III.

3. Designing a User Interface

Everything should be made as simple as possible, but no simpler. (A. Einstein)

3.1. The User Model

One picture is worth a thousand words. However, designers will include in their documents the symbolic descriptions they consider relevant only if they can do so with the same limited effort required to write plain text.

In pursuit of system integration, a tool that generates IC design documents should use the same user model as the system's text editor. A *user model* is the user's conceptual model of the information he

4. For brevity, transistor schematics, gate-level descriptions, and block diagrams will be referred to collectively as "schematics."

or she manipulates and of the processes applied to this information [Newman, 1980]. While the first part of this definition has been covered in section 2.2 (annotations overlaid on layout), this section is devoted to the processes involved.

Like many CAD/CAM tools, VLSI design automation tools have a long history of inadequate user interfaces. Indeed, most of the time, the user's model of VLSI DA systems is rather a brouhaha of commands. There are many interactive commands and variants thereof that the user needs to accomplish design tasks; the tool writers are busy providing the functions of the commands and, in return, the users are willing to make a larger learning investment to "buy" more functions.

A problem arises when a new user wants to learn the system. Although the complexity of the job justifies long training, an inconsistent user model fosters unwanted behavior. The habits formed because of this absence of a consistent user model lead to avoiding some commands and replacing them by other complex command sequences that seem necessary in the user's conception. Thus, the user can take advantage of only a subset of the system. Other researchers have given examples of common errors in user interface design [Nievergelt, 1982] and have presented a detailed analysis of a VLSI editor's user interface [Card et al., 1983, chapter 10].

In summary, there are two main requirements for the design of a user interface within a tool that integrates design automation with a document processing system:

- The sequence of operations that a user must perform to include a symbolic representation in a text document should be as short and simple as possible.
- The tool that generates IC design documents should use the same user model as the text editor.

As is suggested by figure 1, both the task of coalescing a VLSI image with a text document and of printing a VLSI image can be conceived of as a sequence of copy commands. In the first case, each invocation of the command would take a specified portion of the VLSI image and insert an equivalent illustration at a selected position in the text. In the second case, all operations required to submit the specified portion of the VLSI image to a print server[5] would be performed.

Independent of the implementation complexity of a task, such as submitting an arbitrary VLSI image to a print server of any technology, the user's model is kept simple, because conceptually the task is simple.

Having defined what the user's model of the tool should be, the abstract concepts around which the tool is organized have been specified. This is known as the representation of the tool at the conceptual level [Moran, 1980]. In section 3.2, the details of the user interface are specified. If integration is to be achieved, the actual realization of the user interface must depend on the paradigms in its underlying system; a functioning implementation is presented in section 6.

5. A "server" is a device, connected to a network, which provides a service to users on the network. One should therefore think of a server as any collection of the electronics, software, and peripheral equipment necessary to deliver a service.

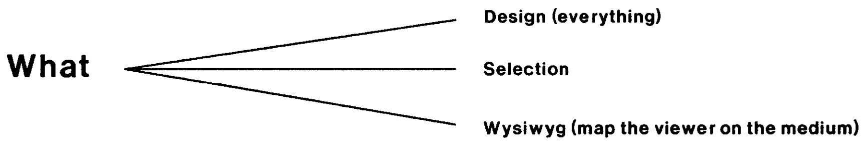
3.2. The User Interface

Exterminate features. (C. Thacker)

Two concepts mentioned above have to be refined. The first is to define "a specified portion of the VLSI image to be acted upon." The second is to specify how the choices and variants for moving a specified portion of the VLSI image are presented to the user. From an implementation point of view, this means that some parameters have to be introduced. In pursuit of clarity and orthogonality, the parameters themselves have to be chosen carefully and, furthermore, their possible values must also be taken into consideration.

Three orthogonal questions are identified: *What? Where?* and *How many?* These questions are sufficient to describe the input, the output medium, and the number of copies for a printer.

Several users have been observed and interviewed. It has been noted that the user's answers to pertinent questions generally remain constant over a session, and therefore the user may consider it a nuisance to set these parameters every time the command is issued. The parameter's values should be retained for repeated invocations of the command. In the following paragraphs, the possible values of the parameters are described.

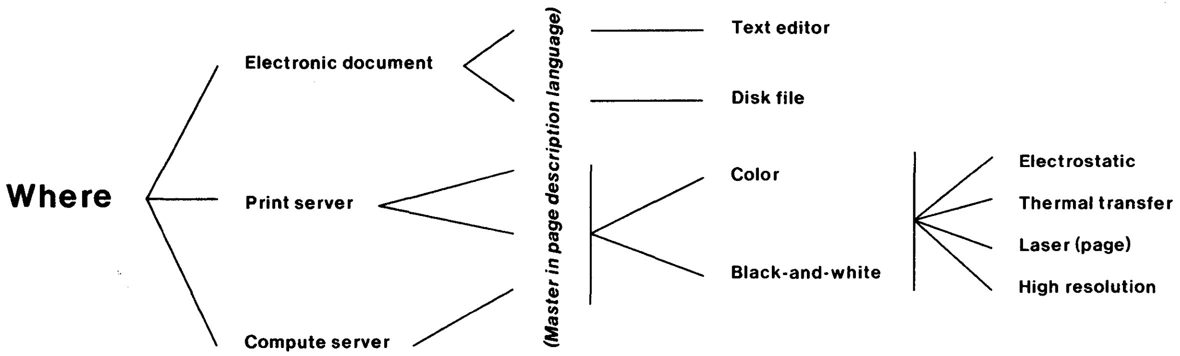


"What" circumscribes the input. Three values are sufficient to do all the required tasks: *design*, *selection*, and *WYSIWYG*. The *design* value indicates the entire content of the VLSI image. Therefore, this value is employed to draw everything. The *selection* value is used to show details in a large scale and is based on the selection concept of the VLSI editor.

The third option, *WYSIWYG* (what you see is what you get), is more subtle. Prior to the availability of the tool described here, coalescing VLSI images with text documents required the use of an additional tool and the specification of a set of parameters, including a cropping window, a translation of the origin, a scale factor, and eventually, for layout, a rotation to change the aspect ratio. However, as often is the case, a simpler method can be more powerful. The user adjusts the window of the VLSI editor so that it displays exactly what should go into the text document. When the command is executed, the tool computes the transformation and fits the image to the text layout. For instance, use of this feature maintained the same font size in the diagram above and the diagram below.

While the first parameter circumscribes the input, the second parameter controls the output. Since the destination is not necessarily a printer, the second parameter sets *goals* rather than devices. Goals can be classified into three sub-groups: electronic documents, print servers, and compute servers.

The diagram below contains the item "master in page description language." For the user, this is an implementation detail, and therefore not part of the user interface *per se*. Nevertheless, it is intentionally revealed to the user, thus it becomes part of the user model [Moran, 1980]. One reason to make the user aware of this extra step is that most systems have a *Redo* or *Repeat* command which reexecutes the last command. To support such a command efficiently, an intermediate data structure called a *master* (a program in this case) is maintained to capture the last imaged input. In this master, the illustration is stored in a normalized size, so the task can easily be redirected to a different goal.



The text editor is the most prominent member of the electronic document sub-group. For the user, the typical operation is trivial; executing the command will cause the selection in the window of the VLSI editor to be copied and fitted at the cursor position in the text document. Repeating this operation allows the user to coalesce a VLSI database with a text document performing only a simple sequence of the cycle *select input*, *select output position*, and *copy*. The implementation details are summarized in section 6.

A disk file is the other choice available in the electronic document sub-group. If the master is represented in a standardized page description language (as discussed at the end of the introduction), writing it to a disk file is obviously valuable. Such masters are useful in electronic document exchange beyond the user's environment and for archival storage, since it will always be possible to print a document in a standardized page description language. An additional application is reading the master into a general purpose graphic illustrator for further processing. This is required when illustrations for books must be improved by graphic artists.

The print servers comprise the second sub-group of output goals. The use of a standardized page description language makes available a large amount of software to drive a variety of different printers. For the purposes of printing VLSI schematics and layout, four printer technologies are particularly valuable.

Electrostatic plotters combine a reasonable resolution of 200 or 400 dpi (dots per inch) with a large paper width. This makes them the preferred output medium for complex schematics involving a large number of symbols and for layout. Thermal transfer printers have the advantage of being able to print on paper that is larger than 8.5 by 11 inches and at medium resolution. This makes them suitable for medium-sized schematics.

Laser printers are well-known devices that print on 8.5 by 11 inch sheets of paper, while the high resolution printer is an experimental laser printer with a resolution of 1200 dpi [Starkweather, 1985].

The latter prints on photographic film or paper; a page is produced for each color separation unless the printer is in black-and-white mode.

The distinction between black-and-white and color printers is made because many color print servers offer a logical black-and-white interface which translates colors to black-and-white patterns and skips the color passes. Since publishers often require black-and-white illustrations, a good tool should support this by presenting a different logical printer for the two possibilities; this is the most natural place to implement it.

Compute servers form the third and last sub-group of output goals [Hagmann, 1986]. Their purpose is to off-load the creation of complex bitmaps from the print server to more powerful or dedicated machines. When this value is selected, the master is sent to a server that performs the raster scan-conversion, producing a (possibly compressed) bitmap. After being notified by the compute server that the bitmap is completed, the tool automatically submits it to the appropriate print server.

How often ———————— **Any number**

In contrast to the first two parameters, *what* and *where*, the third parameter, *how many*, is trivial. It is needed to specify the number of copies to be printed when the goal specified in the *where* parameter is a print or compute server.

3.3. Advanced Functions

Although this user interface covers most needs, in real life the need arises to execute more complex tasks. It is important that such requirements be satisfied by the same tool, since a special tool would result in an overly complex user model. If you add tools, you add their user models, and what you get is not a series of simple user models but a single user model that is a collection of all the previous user models and hence cumbersome and unwieldy.

On the other hand, the user interface of the initial tool should be retained as closely as possible to keep the execution of simple tasks simple. A possible stratagem is to designate the seldom used functions as "advanced" and hide them from the casual user. The way this is realized depends on the paradigm of the system underlying the tool. For example, if the paradigm includes pop-up or pull-down menus, a nested menu can be presented to the user.

The advanced functions presented below cover all the electronic publishing needs of the VLSI designers currently using this tool.

3.3.1. Specified Scale

By default, the input is always scaled such that it fills a sheet of paper for a printer or the width of a column for a text document. An alternative to this scaling relative to a given bounding box is scaling the input relative to the size of the elements it contains. In conjunction with a 1200 dpi laser printer, the tool becomes valuable for the rapid prototyping of simple printed circuit boards. All the user has to do is to specify the value of λ in micrometers[6] and print the mask on a film.

3.3.2. Specified Layer

An interesting problem is the preparation of bonding specifications. These are diagrams that depict which bonding pad of a chip die is to be connected to which pin of the chip carrier. Since the production of a checkplot is an overnight job and contains a great deal of irrelevant information, the designers often draw the bonding specifications by hand on ruled paper. Due to the many pins in full custom VLSI chips, this is a tedious and error-prone task.

Because bonding pads can be distinguished from all other features in a layout by the presence of a layer called "overglass," the solution is simple. The user can limit the input parameter to include only features of a particular layer; this allows to obtain a correct bonding specification in a matter of seconds.

This feature can also be used for the rapid prototyping of simple multi-layer printed circuit boards. The user merely executes the command for each layer and obtains a set of films.

3.3.3. Automatic Page Breaking

When electronic publishing was taking its first steps, it simulated a typewriter. The user had to manually break words, lines, and pages. Similarly, the first programs for schematic capture simulated a template. The size of the template corresponded to the sheet of paper the designer had used for drafting. If a circuit did not fit in a template, the designer had to break manually it into different templates.

With the increased popularity of wide printers and plotters, programs have evolved to permit the capture of schematics on a large surface. Sometimes designers suddenly need up-to-date diagrams for unscheduled presentations. Plotters often have long job queues and, in some shops, only trained operators are allowed to cut the paper.

The user is given the possibility of overriding the default size of the print-out and obtaining a large *ad hoc* plot using a page printer. In effect, the tool scales the input to the user-specified size and subsequently cycles a virtual 8.5 by 11 inch sheet of paper from left to right and top to bottom. At each position, a page of a multi-page master is created. The resulting printed sheets of paper can then be manually taped together to obtain the desired diagram.

3.3.4. Dossiers

The use of dossiers[7] is a highly visible way in which a well-polished tool can change the daily work of designers. The concept is easy. Shortly before a meeting or a design review, the designers open their VLSI databases containing the diagrams to be discussed at the meeting. The relevant diagrams are successively selected, and each one is printed on a separate sheet of paper, scaled to fit the sheet. The

6. Because different VLSI processes permit different minimal device sizes, most VLSI editors specify the geometric dimensions in an abstract unit called λ. When the fabrication masks are created, the coordinates are scaled such that λ corresponds to a given number of micrometers.

resulting multi-page master is automatically printed in collated sequence to produce a dossier. These dossiers are more useful than overhead slides because each participant can annotate his copy, or a single dossier can be circulated for comments.

3.3.5. Ganging

When color separations are printed for book illustrations, it is desirable to place as many figures as possible on every set of separations, a technique known as *ganging* in the printing industry. The reason is that cost depends on the number of elements to be stripped and not on the number of color figures per page. A separate command can be used to specify the field in which the input has to be imaged.

4. Determining Intersections

As was mentioned in the introduction, independence should be maintained from any imaging model offered by the tool's platform, so as not to hamper portability. In section 2.1, the necessity of finding intersections and coloring them individually was explained. Now the problem of finding these intersections will be addressed.

From a theoretical point of view, plane-sweep is the most efficient algorithm to find the intersections of sets of arbitrary polygons [Nievergelt and Preparata, 1982]. Moreover, it has been shown [Beretta and Meier, 1986] that plane-sweep is suitable in practice for coloring intersections and performing raster scan-conversion.

Nevertheless, in the specific case of VLSI layout, a better algorithm is available. When simplicity is more important than a layout with maximal density, the layout may consist exclusively of axis-parallel rectangles. Layouts designed with this restriction are called *Manhattan geometry* in VLSI literature and they are most common when layout assemblers or compilers are used for full-custom and standard cell applications. This greatly simplifies the task of imaging the layout, since only one kind of object, namely rectangles, must be considered.

If plane-sweep is used to find the intersections, the output to be raster scan-converted consists of polygons, so that an algorithm for filling polygons must be used. Since most workstations efficiently support an operation known as BITBLT or RasterOp for filling rectangles [Newman and Sproull, 1979], an algorithm that outputs only rectangles is more desirable. These rectangles appear in huge quantities: a representation of a small IC chip of 35 mm^2 typically contains more than a million rectangles. This large number of rectangles makes an algorithm based on an efficient data structure imperative. With its linear expected-time complexity [Ousterhout, 1984], corner stitching presents a promising data structure for this task.

7. A bundle of documents pertaining to a particular subject.

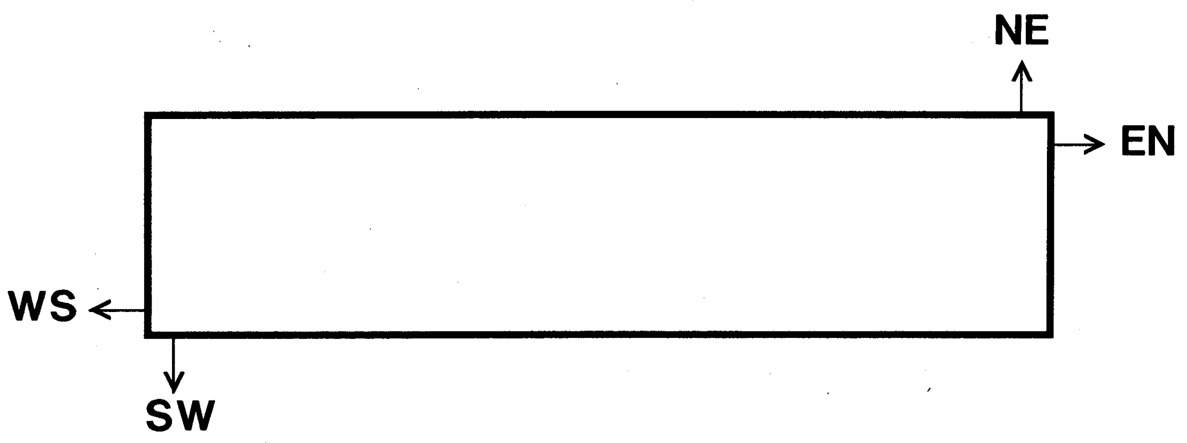


Figure 2. A tile is a rectangle in which an application has a constant value. Four stitches allow programs to visit the tile's neighbors in constant time.

4.1. Corner Stitching

Corner stitching [Ousterhout, 1984 and Shand, 1985] is a data-structuring technique that is efficient for performing local operations on large numbers of small rectangles aligned to the axes. In corner stitching, space is stored explicitly. Because of this, the data structure represents a decomposition of the plane generated by the input rectangles; this is called a *tessellation.* The elements of a tessellation are called *tiles.* The geometrical axioms for a decomposition are satisfied by defining a tile to be a set that is closed at the South and West sides and is open at the North and East sides.

The tiles are linked with four pointers, called *stitches.* Only two corners need to be stitched to accomplish all local operations. The stitches in the lower left corner of a tile are called *South-West* and *West-South,* while those on the upper right corner are called *North-East* and *East-North* (see figure 2).

Each tile has a *value,* which is application-dependent. Space tiles, i.e., tiles that do not correspond to any input rectangle, have the special value Λ. A tessellation is kept unique and minimal by requiring maximal horizontal strips. When a non-space tile is inserted into a space, horizontally adjacent space tiles are split into thinner tiles and then joined into maximal strips. When a new non-space tile collides with an existing non-space tile, a client's procedure must supply a new value for the new tile inserted into the tessellation to represent the intersection. Only then are the maximal horizontal strips formed which restore the consistency of the tessellation.

Thus each tile contains the following information:

- The coordinates of the South-West corner. The coordinates of the North-East corner are determined by the South-West corner coordinates of the neighboring tiles.
- Four stitches to the neighbors.
- The value of the tile.

In his paper on corner stitching, Ousterhout has shown that, in a tessellation with n non-space tiles, there are at most $3n+1$ space tiles. He claims that, on the average, only one space tile is needed in layout for every non-space tile. Despite this claim, the number of tiles in a tessellation is much larger than the number of rectangles in the input. This is because the intersections are decomposed to compute their color. The input for a straight transistor surrounded by white space, for instance, consists of two crossing rectangles (the example used in plate I is a representation of a transistor), while it generates five non-space tiles in the tessellation.

It might be argued that it would be better to defer finding the intersections to the moment when the bitmap is produced. A color map would be used to OR the rectangles into the bitmap. This is a time/space trade-off. Although the number of rectangles becomes much larger by representing the intersections explicitly, every intersection is handled only once at the high level of rectangles to determine the color. When the intersections are resolved at the bitmap level, the operation of OR-ing bit patterns is performed for every bit for every rectangle intersecting on that bit. Even if the BITBLT function is implemented efficiently, doing operations on a word basis as far as possible, this low level method is significantly slower owing to the large number of rectangles in a VLSI layout.

4.2. Client Implementation

To avoid the creation of unnecessary rectangles in the intermediate master (see section 3.2), rectangles that entirely share a common edge and are the same color are merged, using the maximal horizontal strip method introduced above. The basic idea is to assign every color a number which is stored as the value of a tile, as illustrated in plate IV. If two tiles with a common edge bear the same number, they are merged into a single tile with that number.

To retain independence from VLSI technologies, it is preferable not to make assumptions about the possible colors, so that the tool, for instance, can be used for both NMOS and CMOS layout without this being hardwired in the code. Therefore, a separate data structure, called a *color table*, organizes the colors. Each entry in the color table consists of a set of layer colors and a specification of a blend color. The set of layer colors contains all layers that intersect in a given tile, and is the search key into the color table. The blend color specification is the resulting color-blend for the intersecting color.

The value of a tile stored in the tessellation is a reference to an entry into the color table. The algorithm for inserting a rectangle into a tessellation is as follows:

```
for every tile intersecting the rectangle to be inserted
    if the tile already has the layer color of this rectangle do nothing
    else if the tile is empty
        find the reference to the color in the color table
        if it is not there, create a new entry
        set the value of the tile to be this reference
    else (* some rectangles already intersect at this tile *)
        add the layer color of this rectangle to the old set in the tile
```

```
find the reference to the new set of colors in the color table
if it is not there, create a new entry
replace the value of the tile by this reference.
```

After all rectangles have been inserted into the tessellation, and after the colors for the intersections have been blended as described in the next section, they are painted into the master using a page description language as described in section 3.2. Since, in the geometrical sense, the tessellation is a decomposition of the plane, a simple and fast intermediate encoding can be produced.

5. Coloring the Intersections

At the beginning of the VLSI era, when color monitors were quite expensive, the different layers of a layout were represented using different cross-hatch patterns. This method is still used in many books and conference proceedings. For VLSI image editing, the introduction of color has led to a substantial increase in the designer's productivity. When VLSI books are borrowed from libraries, the illustrations of layout are frequently hand-colored. This indicates the desirability of the use of color in all VLSI publications as well.

A chip die consists of silicon and aluminum, which are both grey. In principle, the colors representing the various layers can be chosen at random. However, when a die is observed under a microscope, colors are detected, as is demonstrated by color plate V. Visible light has a wave length between 0.38 μm and 0.78 μm, and the thickness of some of the physical layers created on a silicon wafer falls within this range. Therefore, when white light is shone through a wafer, the physical layers act as dichroitic filters. Where different layers overlap, interferences at the intra-layer boundaries create additional colors. Typically, the colors adopted in VLSI editors are inspired by these colors.

Since the thickness of one and the same layer is different from one manufacturing process to another, there is no canonical choice of colors: hence, the color conventions used vary from one VLSI editor to another. Mead and Conway [1978, p. 64] proposed the following colors for the NMOS process: green for diffusion and transistor channels, yellow for ion implantation, red for polysilicon, blue for metal, and black for contact cuts.

Plate VII shows the colors displayed on a color monitor by the VLSI editor introduced in section 6. To stress the unsatisfactory distribution of the colors, these are displayed as points in a CIE 1931 (x, y)-chromaticity diagram [Wyszecki and Stiles, 1982, Foley and Van Dam, 1982]. This sample has been taken from a CMOS (complementary metal oxide silicon) double metal layer technology. Tones of green, red, cyan, magenta, yellow, and light yellow are used to represent diffusion, polysilicon, metal-1, metal-2, transistor gates, and wells, respectively. When layout is printed, the conventions for the hues should be roughly retained to avoid confusion.

A large proportion of the colors in plate VII are nuances of green, yellow, orange, indigo, and magenta. Outside these five clusters, there are only a few instances of other colors, such as blue, cyan, and red. A plausible explanation of this state of affairs is that the implementors of VLSI editors select the color tones for the layers merely using aesthetic criteria. Subjectively, light and vivid primary colors are quite eye catching on a color monitor, therefore, the implementors may tend to select such colors, not bothering about an even distribution over the available gamut of colors. Figure 5 on page 22 shows the lightness distribution of the colors displayed in plate VII.

Color monitors have a large color gamut compared to paper. This makes acceptable the clustering of the colors used in VLSI layout editors with respect to the color spectrum. On the other hand, an accurate mapping of the colors seen on the monitor to a device such as a printer is quite difficult if not impossible, because of the completely different color gamut.

5.1. A Simple Solution

Since in VLSI layout the exact visual perception of a color of a given swatch is not as important as the fact that the swatches are easily discernible, it is possible to "spread" the clusters and to eliminate the greys. A simple heuristic approach is enough to obtain an acceptable result.

Using this simple method, the colors written in the master have a more uniform distribution over the entire color spectrum than the layer colors used by the VLSI editor, achieving more robustness in view of the mismatch of the gamut of the color monitor and of the various color printers. Designers have printed images produced with the tool described in this report on such diverse devices as high-resolution color laser printers, thermal transfer printers, and Versatec color plotters.

The difficulty of making a judgment of the relative magnitude of two color differences is reduced if the colors involved are closely similar in at least one of the three basic attributes of color perception, such as hue, saturation and lightness[8]. An adequate color model common in computer graphics is the HSL color model, although this model is used much more informally than in color science [Schwarz et al., 1987]. As shown by Foley and Van Dam [1982], it is easy to convert a color from the RGB model to the HSL model and vice versa.

In a simple implementation, the colors for the various layers present in the VLSI image are converted to HSL values. The hues for the layers are then distributed more uniformly across the spectrum and the resulting values are reconverted to the RGB model.

For the intersections, the RGB color model is used to construct a simple blending algorithm. Each layer is given a weight and the weighted mean of the RGB values of the colors for the intersecting layers is calculated, yielding the color for the intersection. The weight is not fixed, but depends on the context, i.e., the other intersecting layers. Given the small number of layers in a VLSI design, the implementation is a straightforward application of dynamic programming.

8. Hue is the attribute of color perception denoted by blue, green, yellow, red, purple, and so on. Saturation is the attribute of a visual sensation which permits a judgment to be made of the degree to which a chromatic stimulus differs from an achromatic stimulus regardless of their brightness. Lightness is the attribute of a visual sensation according to which the area in which the visual stimulus is presented appears to emit more or less light in proportion to that emitted by a similarly illuminated area perceived as a "white" stimulus [Wyszecki and Stiles, 1982].

The color values for the CMYK model used in printers are obtained with a simple approximative calculation. First cyan, magenta, and yellow values are calculated by subtracting red, green, and blue from one [Foley and Van Dam, 1982]. Finally grey, i.e., the minimum value of cyan, magenta, and yellow, is subtracted from these three colors and used as the black value. The technique of removing grey and placing it into the black separation is known as *gray component replacement* or *GCR* [Bruno, 1983]. The proposed radical degree of GCR is unusual, but it allows the size of the print files to be reduced by 25%, which in the case of VLSI layout is a large number of megabytes.

As a starting value, if there are n layers in an intersection, each is given weight $1/n$. The value is then determined from a look-up table, which returns a factor for every layer depending on the other layers present. The weights are then renormalized so that their sum is 1 and the weighted mean is calculated. Cuts and vias are exceptions because their factor is always infinite to give them weight 1.

This allows one to fine-tune the rendition of such delicate layers as wells. Physically, a well is an implant in the substrate, which means that it is below all other features on the chip. An accurate representation would be to treat it as a background color. In a straightforward implementation using a blending algorithm based on mean values, the rendering of the wells looks like a veil over the area it covers. For the designer, it is subjectively disturbing to perceive wells to be on top of the other layers when he knows that the wells are underneath. Using context-dependent weights easily solves this problem, as it solves the metal-2 problem described in section 2.1.

Some experiments with different weight strategies have been carried out. Designers have been asked to read layout produced by someone else, and the blending that permits the maximum readability has been chosen for the final implementation. Plate VI shows an inverter imaged with these colors.

Typically in VLSI layout there are millions of rectangles, while there are only around 100 possible intersection combinations. Therefore, the color table introduced in section 4 is used to cache the blended colors to avoid recalculating for every tile in the corner-stitched plane. When all rectangles have been inserted in the tessellation, the color table contains all intersection combinations appearing in the input. The color table is traversed only once and a color is blended for every entry. As the last step of imaging IC layout, the tessellation is enumerated into the intermediate master using the information in the color table.

5.2. A Methodology for Selecting Discriminable Colors

Although the simple method just described produces usable layout prints, a significant improvement can be obtained by applying the colorimetric methods developed in color science. These methods allow the achievement of significantly improved readability and the use of a smaller color palette than a physical model combined with transparent colors as described by Trimberger [1987].

While applications like avionics initially triggered the development of such methods for color monitors [Merrifield, 1987 and Silverstein, 1987], these methods have not been applied to computer generated hardcopies. The material in this section is based on the book by Wyszecki and Stiles [1982], to which the reader is referred for further details. This section is limited to the presentation of the material necessary for the task at hand.

5.2.1. Basic Concepts of Colorimetry

The process of color perception is very complex and not yet fully understood, therefore there is no simple physical model for the process. This precludes the possibility of finding a set of mathematical formulas to determine the colors to be used for imaging layouts.

The basic idea of colorimetry is color matching, whereby well-defined standard observers are asked to match a color which they are shown. In fact, *color* is defined as that aspect of visual perception by which an observer may distinguish differences between two structure-free fields of view of the same size and shape, such as may be caused by the differences in the spectral composition of the radiant energy concerned in the observation [Wyszecki and Stiles, 1982, p. 487].

If a color is to be measured, it is shown under well-defined conditions to a number of standard observers as defined by the CIE (Commission Internationale de l'Eclairage). It is possible and convenient to represent the color stimuli by vectors in a three-dimensional space, called the *tristimulus space* [ibid., p. 119]. A stimulus **Q** can be matched in color to an additive mixture in suitable amounts of three fixed primary stimuli; the scalar multipliers of the primary stimuli are called the *tristimulus values* of **Q** [ibid., p. 121].

It is natural to use **R**, **G**, **B** (red at 0.700 μm, green at 0.546 μm, and blue at 0.435 μm) as the primary stimuli. The result of a color match, then, is the proportion of red and green the average standard observer has perceived (blue is calculated by subtraction, because the values are given as proportions). The proportions of red and green for a number of stimuli can be plotted in a *(r, g)-chromaticity diagram.*

For a number of cases, the observer will have to desaturate the stimulus to be matched by adding red or green to it; thus the proportion of red and green can assume negative values [ibid., p. 124]. Having to deal both with positive and negative values has several practical disadvantages. This has led to the introduction of an imaginary tristimulus space with primary stimuli **X**, **Y**, and **Z**, such that stimulus values in XYZ-space are always positive. The axes **X**, **Y**, and **Z** do not correspond to real colors. The transformation from the RGB-space into the XYZ-space can be found at page 139 [ibid.].

The CIE 1931 (x, y)-chromaticity diagram as it is shown in plate VII is obtained by projecting stimuli in XYZ-space onto the plane $x + y + z = 1$ to normalize against luminance. The horseshoe-shaped curve is called the *spectrum locus* and represents the fully saturated spectral hues. Because printed VLSI layout is always examined at a scale where features are sufficiently large, in this report the spectrum locus is plotted from the data of the CIE 1964 Supplementary Standard Colorimetric System [ibid., p. 738], which is based on a supplementary standard observer with a 10° angular subtense (large visual field). The straight line connecting the two ends of the spectrum locus contains non-spectral colors and is called the *purple line.*

The transformation from the RGB values of a color monitor into tristimulus values in XYZ-space is linear [Meyer and Greenberg, 1987]. Thus, if the red (maximum x-coordinate), green (maximum y-coordinate), and blue (minimum x-coordinate) points in plate VII are connected to form a triangle, all the colors that can be displayed on the monitor where the values for red, green and blue were measured are inside this triangle. This triangle delimits the *gamut* of the particular monitor. For comparison, a simple approximation of the gamut of an electrostatic color printer has been plotted in plate VII.

Because the printer's gamut is smaller, colors that can be discriminated on the color monitor will be printed in colors that can no longer be distinguished. Furthermore, as the shape of the two gamuts suggests, there is no linear transformation from one gamut to the other, so that it is not possible to find a simple mapping from the colors used to display layout on the monitor to the colors used to print layout. This makes it necessary to define a new palette of colors for layout. They are called *discriminable colors* because they can be discriminated both on a monitor and on hardcopy.

The (x, y)-chromaticity diagram is useful for judging the saturation and complements of colors. Techniques using it are described by Foley and Van Dam [1982, pp. 607-610] and are not repeated here. Unfortunately, this diagram is not uniform;, the distance of two colors in the (x, y)-chromaticity diagram is not related to the perceived difference of the colors, so that it alone does not suffice to find a palette of discriminable colors for VLSI layout.

The CIE has defined two uniform color spaces which are appropriate to predict the magnitude of perceived color difference between two object-color stimuli, i.e., in which the Euclidean distance between pairs of near colors is proportional to the magnitude of perceived color difference between them. One of these spaces is called CIE 1976 $(L^*a^*b^*)$-space, abbreviated CIELAB.

The transformation from the XYZ-space into the CIELAB-space is not linear, making it hard to visualize it. The transformation and a sketch of the CIELAB-space can be found on page 167 of Wyszecki and Stiles [1982]. L^* is in the range (0, 100] and is the *lightness*, the attribute of a visual sensation by which the colored feature in which the visual stimulus is presented appears to be lighter or darker in reference to the "white" paper on which it is printed.

The a^* and b^* axes are more difficult to imagine. A useful experiment is to plot the values for a^* and b^* for a number of constant L^* values in the interval (0, 100], thus obtaining slices in the CIELAB-space. Plate VIII shows such a slice for $L^* = 50$. The colors on the horizontal a^* axis vary from "dark vivid green cyan" to "vivid magenta red" for increasing values of a^*. Similarly, on the vertical b^* axis they vary from "vivid cyanish blue" to "vivid yellow brown" for increasing values of b^*.

Plate IX shows such a slice for $L^* = 90$. The colors on the a^* axis vary from "lightish vivid green cyan" to "light vivid magenta red" for increasing values of a^*. Similarly, on the b^* axis they vary from "light cyanish blue" to "lightish vivid orange yellow" for increasing values of b^*.

Black and white are the two lightness extremes for null chromaticity, i.e., black typically has coordinates $[\varepsilon, 0, 0]$ in CIELAB-space and white [100, 0, 0], where ε is a small positive number. White is best used to represent the layout substrate, and black can be used to represent the cuts.

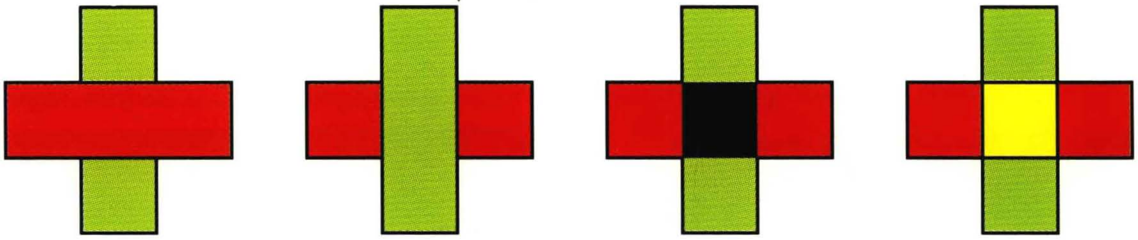


Plate I. Assume that first the vertical green rectangle is drawn, then the horizontal red rectangle is added. Four possible imaging models are: the second rectangle covers the first, the second rectangle is behind the first, the rectangles are transparent and colors are subtractive, or the rectangles are transparent and colors are additive.

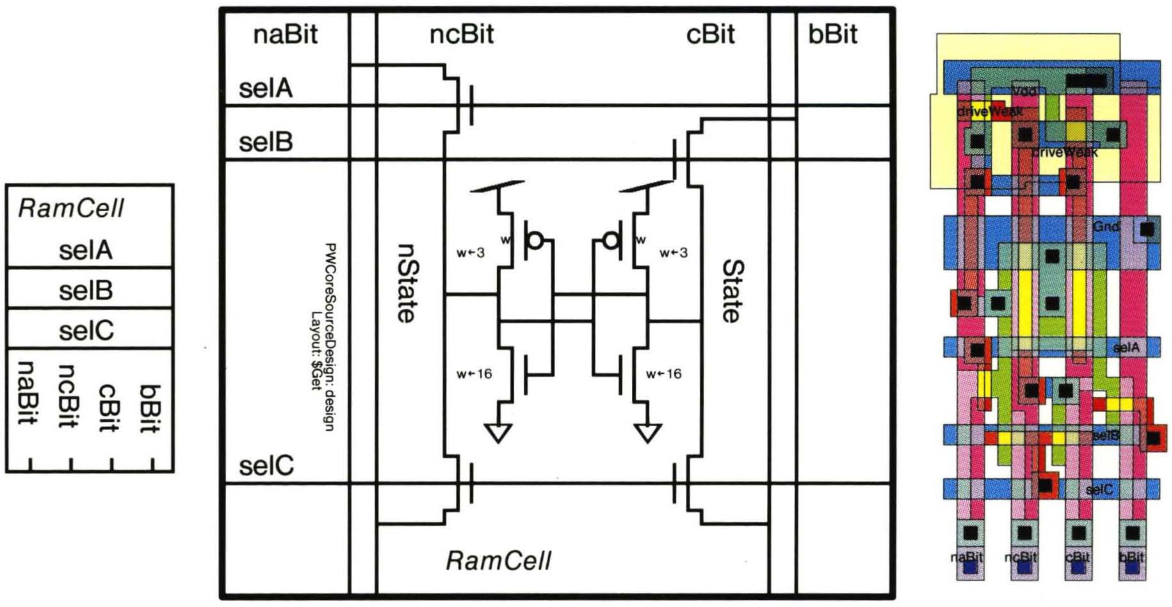
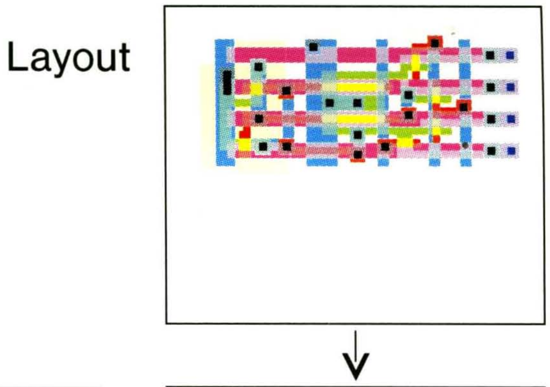
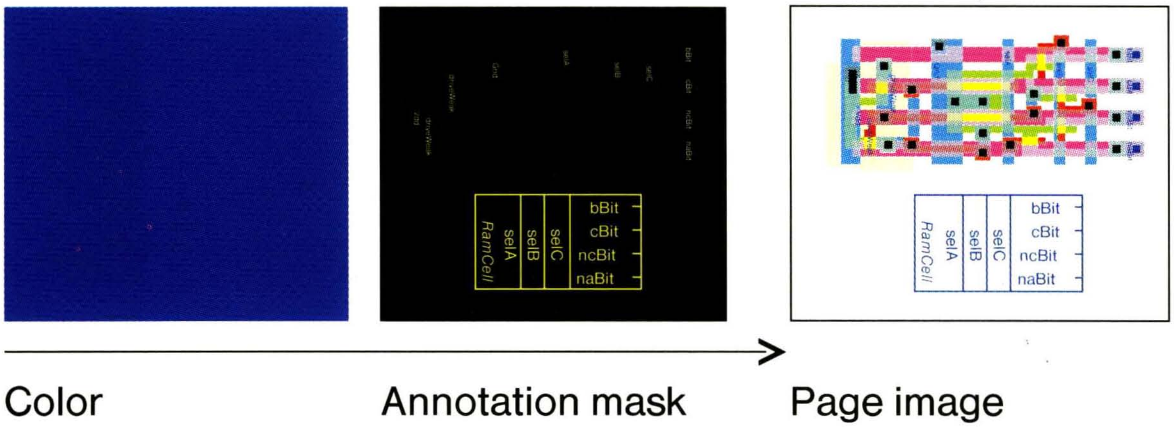


Plate II. A design database may contain any combination of representations for the data, such as block diagrams, schematics, and layout. The ability to print such files efficiently and insert them into text documents is desirable (as demonstrated by this figure).

Plate III. A desirable imaging model superimposes annotation on layout. First the layout is painted onto the page. Then a mask of the annotation is used to apply the color used for the annotation to the page image.





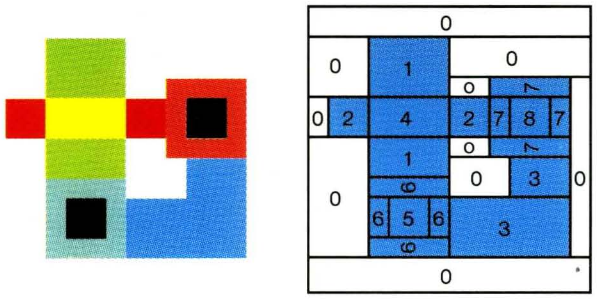


Plate IV. The layout at the left is decomposed in the tessellation shown at the right. The numbers in the tiles are their respective values, which indicate the colors to be painted into the master.

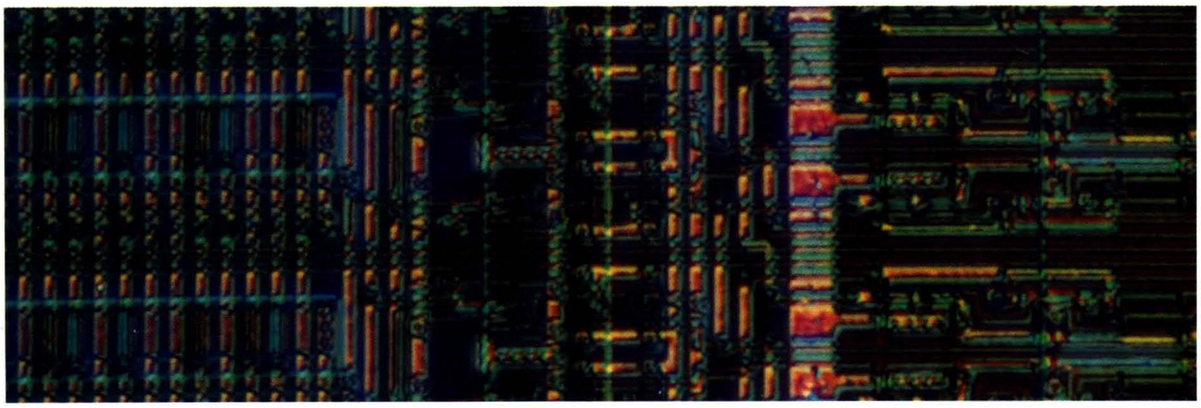


Plate V. If white light traverses a wafer, the features on the latter diffract the light acting as dichroitic filters. Where different layers overlap, interferences create additional colors.

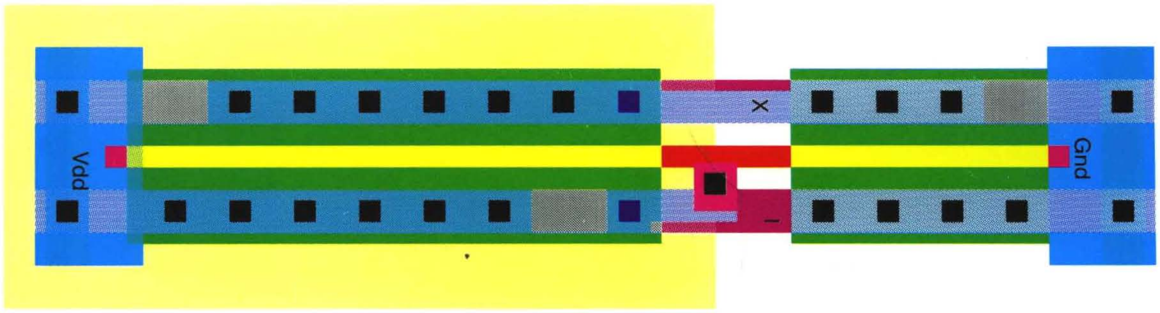


Plate VI. Layout for an inverter in CMOS technology imaged using the colors obtained with the simple method of "spreading" the hues for the layers uniformly across the spectrum.

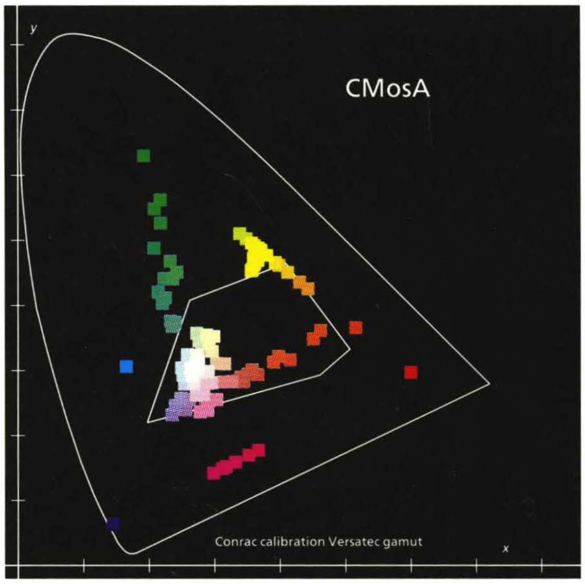


Plate VII. Colors used by a typical VLSI editor to display layout on an eight bit per pixel color monitor. The colors are represented as points in a CIE 1931 (*x*, *y*) – chromaticity diagram. The irregular hexagon approximates the gamut of an electrostatic printer.

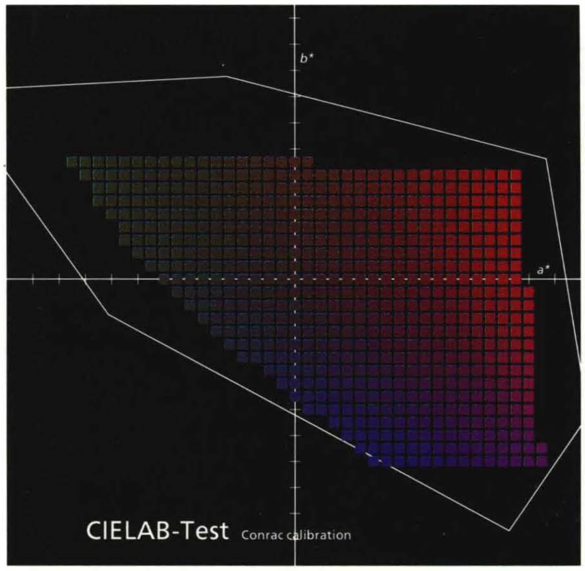


Plate VIII. A slice in CIELAB – space through the colors that a Conrac monitor can display at *L = 50. The irregular hexagon is a projection of the gamut of the monitor approximated by red, yellow, green, cyan, blue, and magenta. The abscissa is *a** and the ordinate *b**.**

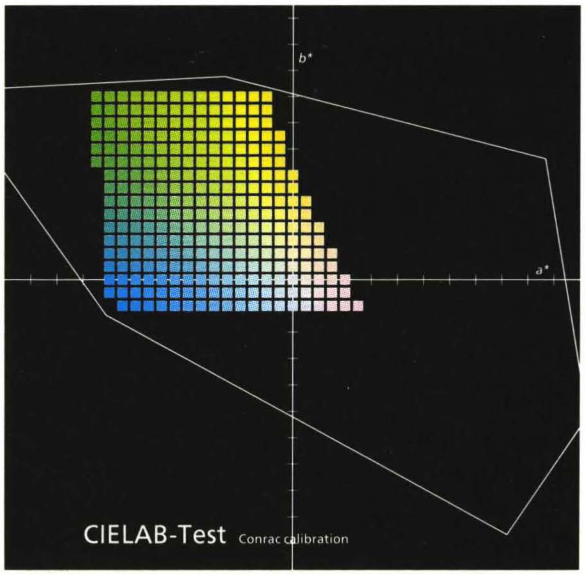


Plate IX. A slice in CIELAB – space through the colors that a Conrac monitor can display at *L = 90. Note the shift towards the second quadrant and the appearance of yellow. The abscissa is *a** and the ordinate is *b**.**

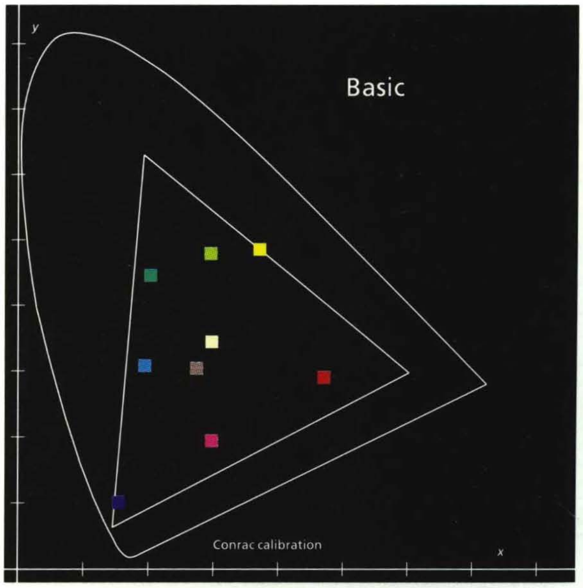


Plate X. CIE 1931 (*x*, *y*) – chromaticity diagram showing a basic palette of 9 colors that are suitable for representing VLSI layout. Blue can be used for vias, vivid yellow for gates, and light yellow for wells.

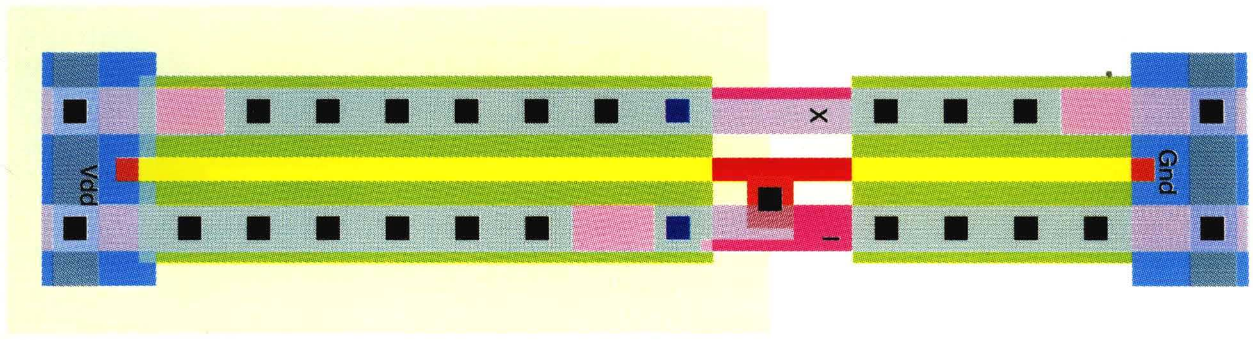


Plate XI. The layout for the CMOS inverter shown in plate VI revisited. This figure is imaged using a palette of discriminable colors.

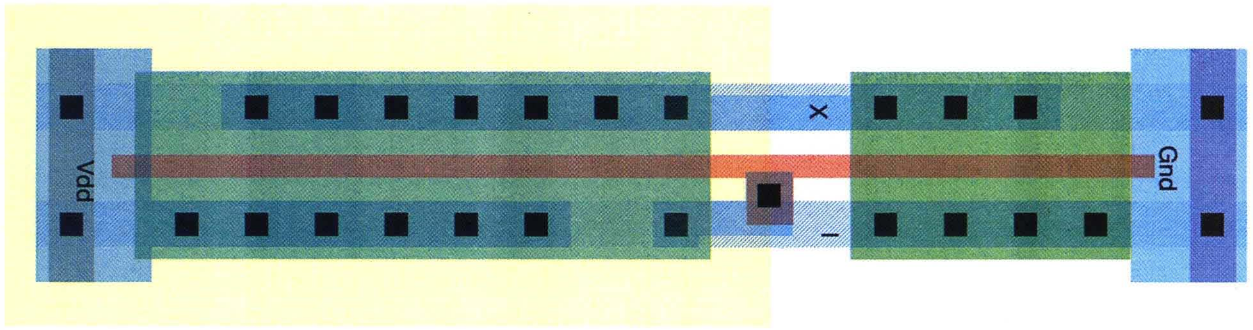


Plate XII. Same inverter layout as in plate XI, but plotted with a conventional state – of – the – art checkplot program. See page 23 for a discussion of the print quality.

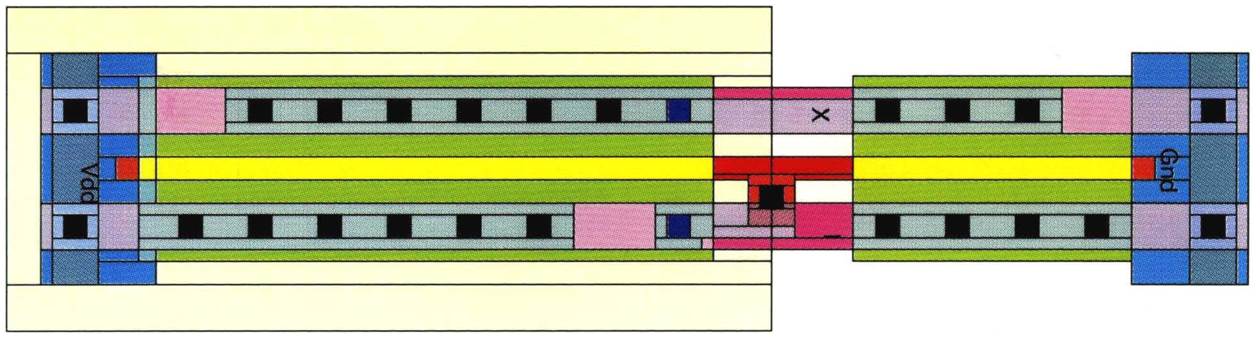


Plate XIII. The same layout as in plate XI, but with the rectangles in the VLSI image outlined. The rectangles are not unique, but depend on how the designer lays out the circuit.

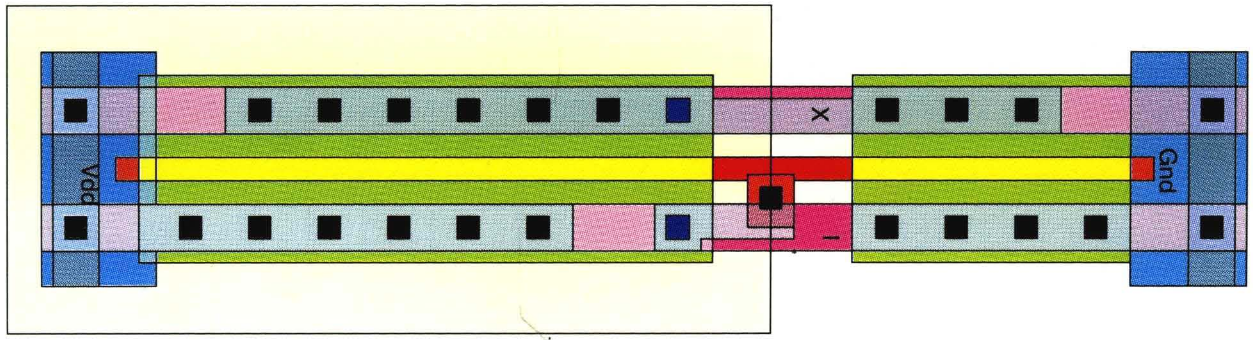


Plate XIV. The same layout as in plate XI, but with the regions outlined.

5.2.2. Problematic Colors: Blue

The rendition of VLSI layout on a printer is problematic for two colors: blue and yellow. To understand why blue is problematic, the difference between an additive and a subtractive color model must be understood (see plate I).

A color monitor is an example of the additive color model. The screen is a black matrix. There is a raster of three distinct phosphor dots for red, green, and blue. A primary color like red is generated by activating the phosphor dots for red. To generate a secondary color, for example yellow, green is added by also activating the green dots. The pitch (distance between the shadow-mask apertures) of the color monitor is sufficiently small that the human eye cannot discern the single dots and integrates the red and green stimuli into a single yellow stimulus. If the blue phosphor dots are activated too, blue is added to the yellow and white is obtained.

A set of dichroitic filters in the colors cyan, magenta, and yellow is an example of the subtractive color model. We observe a white object. A primary color, for example cyan, is generated by holding the cyan filter in front of the object. The filter reflects the complementary color, red in this case. Subtracting red from the white object, it is seen as cyan. To generate a secondary color, for example blue, a magenta filter is added to subtract green also. If the yellow filter is added too, blue is subtracted from the blue and black is left.

The reason that cyan, magenta, and yellow are used for printing is historical. In the printing industry, plates have been primarily produced by a negative/positive process. Before the availability of color scanners, the negative color separations were produced photographically using a set of filters in the colors red, green and blue. To print the plates produced with these separations, inks of the complementary colors had to be used [Strauss, 1967].

In color printing, a mixture of the additive and the subtractive color models is used. An ink has only one lightness value. Lighter colors are obtained by applying ink spots of multiple sizes and/or varying the distance between spots. Since the ink coverage is not continuous, sometimes two ink spots of different colors are printed one on top of the other and sometimes they are printed one next to the other. For a first discussion, let us assume that the inks combine ideally in the subtractive color model.

The inks are not light emissive, but filter the light reflected by the paper. This reflected light is not white. Most papers reflect least light in the blue, more in the green, and most in the red zones of the spectrum. As a result the colors of the printed image are shifted towards the red [ibid., p. 166].

Inks are lacking in the beamsplitting dichroitic quality that dichroitic filters have. Instead of absorbing only green, magenta inks also absorb some blue and some red. Cyan inks are the worst, because they absorb some green light, do not reflect enough blue, and reflect some of the red light they should absorb [ibid., p. 192]. Yellow inks are almost perfect and do not represent a problem *per se*.

The paper substrate on which inks are deposited is not illuminated from behind but by reflection. If a cyan mark is deposited on a white sheet of paper, the illuminating white light traverses the mark, is reflected by the paper, traverses the cyan mark again, and is observed. If a magenta mark is deposited over or under the cyan mark, the light illuminating the lower mark is no longer white.

In summary, blue cannot be produced by applying equal amounts of cyan and magenta. The solution is to perform a color correction, i.e., to use less magenta than cyan, so that the resulting color is blue. This technique is well understood and has been incorporated in electronic color scanners since their advent on the market in 1955 [ibid., p. 163].

Unfortunately, different printers use quite different inks, so there can be no device-independent color correction. The only solution is to use blue to represent a layer that is not critical, such as vias. Since it is sufficient that it be different from the black used for cuts, a blue is obtained by using an arbitrarily higher proportion of cyan than magenta. This color will never be mixed with other colors because vias always "win" over the representation of the other VLSI layers, so there is no possibility of confusion.

5.2.3. Problematic Colors: Yellow

A few paragraphs ago, it was claimed that yellow inks are almost perfect. In fact, the problem for yellow is not in the inks but in the color itself. For spectral colors, only the narrow range around 0.57 μm to 0.59 μm is categorized as yellow [Murch, 1987]. If this frequency is increased, the color is quickly categorized as yellow-green; if the frequency is decreased, the color is quickly categorized as orange. If plates VIII and IX are compared, it becomes clear that the perception of yellow "degrades" quickly to orange and brown as it is darkened. Yellow is so light that in a chromaticity diagram or in plate IX you cannot clearly see the edges that delimit it.

For this reason, it is not a good idea to use yellow to represent wires, as the wires would become hard to recognize when they run above or below a wire of a different layer. Also, yellow inks are quite vivid, even more saturated than the yellow that can be produced on a color monitor. Combined with the fact that yellow is lighter than the other inks, a yellow feature will always stand out.

When VLSI layout is read, the most important features are the transistors, the active devices. Transistor gates should be recognized quickly and unmistakably. If a metal wire passes over a gate, this wire is easily identified because most transistors are short and wide; therefore, the color of the gate does not have to be strongly modified to show coverage by metal.

The best use of yellow is therefore to represent transistor gates. If 100% yellow is used to this purpose, a "very light vivid yellow," say 30%, can be used to represent wells in CMOS or ion implant in NMOS layout, as suggested by Mead and Conway [1978]. In particular, it is not a good idea to use yellow to represent metal-2 layers, where wide wires are run on top of diffusion, polysilicon, and transistor gates.

5.2.4. Basic Colors

So far, colors have been found to represent cuts (black), vias (blue), and gates and wells (yellow). Colors have to be determined for the remaining layers, namely diffusion, polysilicon, and two or more metal layers. Which color is used to represent each layer is not so important, although it is a good idea to follow some standard such as that proposed by Mead and Conway.

The above list contains at least four layers and cyan, magenta, red, and green are four colors that could be used to represent them. However, if an additional layer must be represented, such as a third metal layer, there are not enough colors. The narrowness of the range of yellow can be exploited to obtain two additional colors to represent layers. Towards green, yellow-green can be added as an additional color. Similarly, towards red, orange or brown can be added.

Plate X shows a chromaticity diagram with a possible palette of basic colors. Note that, to accommodate the small color gamut of printers, green has been replaced by "lightish very strong yellow green" and red by "lightish vivid red orange red".

5.2.5. Traceability and Discriminable Colors

The palette of the basic colors allows the representation of the single layers. In section 5.1, an informal physical model was chosen and the colors of the layer's intersections were calculated using weighted means. When designers read layout, they do not rely on a physical model. This fact can be exploited to reduce the number of colors in the complete palette, thus making the task of finding discriminable colors easier.

One solution could be to select the colors in the palette so that their coordinates in CIELAB-space were all equidistant. However, layout rendered using such a palette is not very readable.

Designers read layout by tracking wires. Therefore, the main criterion for selecting colors for the intersections is to make wires traceable.

It is convenient to divide wires into two classes. The first class consists of the wires on the polysilicon and the two diffusion layers. These wires are usually narrow and are dense in areas of active logic. The second class consists of the metal wires. These wires are usually wide and run on top of cells to distribute V_{dd}, ground, and the clock(s). Class two wires are used for the routing.

The human visual system is more sensitive to changes in lightness than to changes in hue or saturation [Foley and Van Dam, 1982]. For this reason, the colors in the basic palette are not only different in their chromaticity (see plate X), but also in their lightness (as is shown in figure 3).

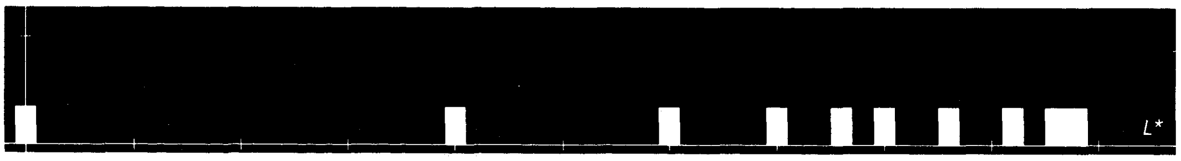


Figure 3. Lightness distribution of the colors in the basic palette shown in plate X. The distance of two tick marks on the abscissa is 10 units in *L.**

When the colors for the intersections of layers are selected, three principles are used:

1. The lightness of class one wires is kept constant.
2. The dominant wavelength of class two wires is varied as little as possible.
3. When it is important to discriminate two different intersection types, the coordinates colors in the CIELAB-space plane are kept as distant as possible.

The first principle allows the designer to trace a class one wire by its intensity alone, making it very easy to follow such a wire when it runs under a class two wire. Remember that the intersection of two class one wires is a gate, which is handled separately as described in sections 5.2.3 and 5.2.6.

The second principle takes advantage of the width of class two wires. Such wires are global in nature and their hue is sufficient to track them. The *dominant wavelength* of a color can be determined graphically using the CIE 1931 (x, y)-chromaticity diagram. White is used as the achromatic reference color. When the point corresponding to the basic color of a class two wire is connected with a straight-line to this reference color, the intersection of this line with the spectrum locus is the dominant wavelength. The colors along this line between the achromatic reference color and the spectrum locus differ only by saturation.

The third principle complements the second. As I explained in section 5.2.1, the (x, y)-chromaticity diagram is not uniform, thus two colors differing slightly in saturation may not be discriminable. As

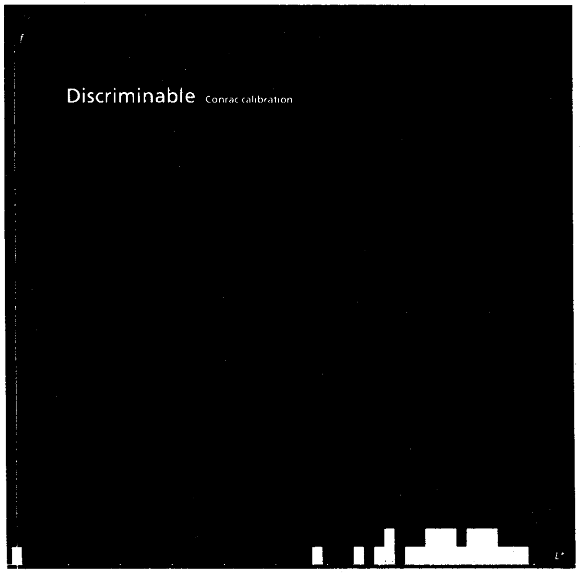


Figure 4. Lightness distribution of the colors in the palette of discriminable colors. The distance of two tick marks on the abscissa is 10 units in L^*. This histogram is printed at the same scale as figure 5.

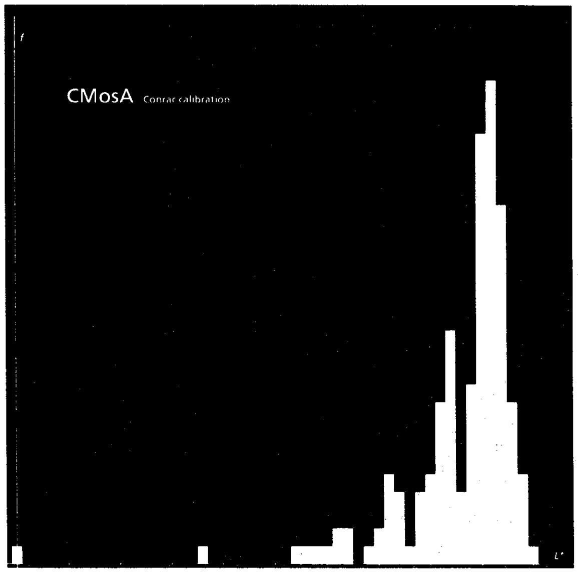


Figure 5. Lightness distribution of the colors used by a VLSI editor and shown in plate VII. The distance of two tick marks on the abscissa is 10 units in L^*.

a first step, the lightness of the colors obtained by the second principle is distributed more uniformly, as shown in figure 4. It is interesting to compare this lightness distribution with the one used by a VLSI editor for display on a monitor and shown in figure 5 (see also the discussion in the introduction of section 5).

Note that the use of discriminable colors, maximizing the traceability of layout, permits the use of a much smaller palette, which allows for more room in the distribution of hues and saturations. The palette of discriminable colors used in the implementation described in section 6 contains only 25 different colors. In particular, wells are often shown only between wires and not always combined with the colors of wires, as in the simple method described in section 5.1. The distribution of the colors in CIELAB-space is depicted in figure 6.

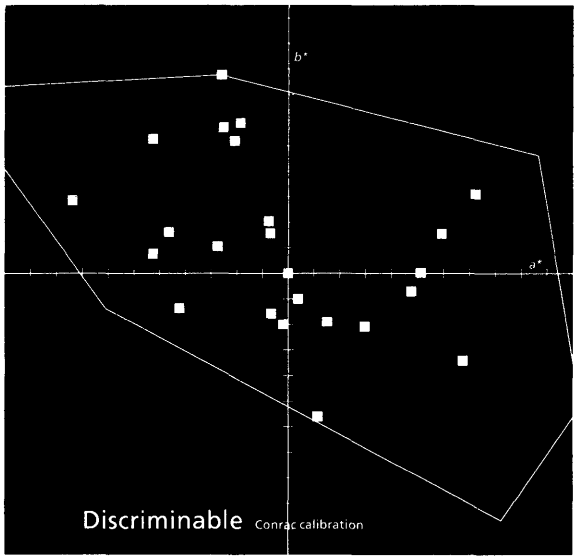


Figure 6. The palette of discriminable colors projected on the *a*b plane of the CIELAB-space. The abscissa is *a** and the ordinate *b**.**

Plate XI revisits the CMOS inverter shown in plate VI. For comparison, plate XII shows the same layout plotted by a conventional state-of-the-art checkplot program. In the original offset print of this report, plate XII probably looks better than plate XI. This came as an accident during the lithography phase. The separations for the color plates have been printed directly on film using a high resolution laser printer [Starkweather, 1985]. For fairness, I masked out the location for plate XII and had a professional lithographer from Kedie/Orent Inc. in Sunnyvale (California) reproduce the separations photographically from a good plot made on a Versatec Electrostatic Color Plotter. While I used a resolution of 106 dpi, Kedie/Orent used 150 dpi; moreover, the original was oversized by a factor of 2, further improving the quality. Trust me, the original Versatec plot is inferior to the original for plate XI, exactly the opposite of the Cromalin proofs I received from the Kedie/Orent.

5.2.6. A Note on Transistor Gates

In section 5.2.3, I advocated the use of vivid yellow for rendering transistor gates. My motivation was to give them prominence, since transistors are the active devices in the circuit. Thus, when a VLSI designer looks at the layout of a cell, he can quickly recognize the cell's function by matching the pattern the gates form.

When the layout is used as a checkplot, the designer already knows the function of a cell and wants to verify its correctness. As elucidated in the previous section, he does so by tracking wires. In this case, the fact of polysilicon over diffusion forming a transistor gate is less important than the ability to easily assess the ramifications of the polysilicon wire.

In the implementation described in section 6, the color palette is loaded dynamically. I supply a choice of two predefined palettes; one rendering the gates in yellow and one rendering them in the same hue as the polysilicon wires. The first palette is being more frequently used.

5.3. Detail Enhancement for Improved Traceability

The human visual system is pipelined in three stages. In the eyes, *reception* is performed, which consists of statistical photon counting. *Extraction* takes place in the cortex and comprises among other functions, edge detection, color identification, and measurement of orientation and motion velocity. The third stage, *inference*, is commonly called the "mind's eye" and accomplishes the perceptual inferences – educated guesses – about the state of the outside world.

In section 5.2, I discussed these three stages in this order. In practice, section 5.2 can be summarized as ascertaining the perceptual inferences drawn by a VLSI designer (traceability, transistor functions), easing the extraction task by facilitating color identification, and selecting the colors that produce the required reception.

Hubel and Wiesel have shown [1968, Hubel et al., 1978] that the most prominent process in the extraction stage is edge detection. In section 5.2, I discussed how the *continuity* of wires could be rendered by appropriately choosing lightness and hue. In this section, I present a simple method that allows quick recognition of regions (that is, *discontinuities*) by facilitating edge detection.

Due to lateral interactions in the neural network of the visual system, Mach bands are perceived near the border of two juxtaposed fields of different luminance [Wyszecki and Stiles, 1982]. On the dark field, a darker band is perceived and on the light field, a lighter band is perceived. Figure 3 demonstrates that a black line separating any two regions that are not cuts will assure that Mach bands are perceived, thus aiding edge detection.

This technique is well known in animated cartoons and comic strips, where regions are outlined in black. In the printing industry, a mask is used to exaggerate the edges between fields of different luminance, a technique known as *detail enhancement* (or *peaking*, in the case of electronic color scanners) [Southworth, 1979].

Some tools to create checkplots attempt to use this stratagem by outlining all geometric figures in the layout, especially when they use cross-hatching to fill the regions. Although this works to some extent if the underlying VLSI editor is based on polygons, it fails if the VLSI editor is based on rectangles, as is shown by plate XIII. The regions of a given color are not rectangular, so outlines clutter the interior of regions.

Since the data is already stored in a corner stitched plane, it is easy to devise a region-finding algorithm. As the tessellation is enumerated and each tile is painted into the master, the neighborhood of the tile is explored. If a neighboring tile's index into the color table is different than the tile's index, a black outline is painted along their boundary. The result is shown in plate XIV.

6. An Implementation

This section describes the platform on which a prototype implementation called *Nectarine* has been realized and how this implementation has been integrated.

6.1. The Cedar Platform

Cedar is a programming environment whose primary purpose is to increase the productivity of programmers whose activities include experimental programming and the development of prototype software systems [Swinehart et al., 1986].

Nectarine processes images produced with the VLSI editor ChipNDale [ibid., p. 458], which can contain both schematics and layout. For the intermediate master, the Interpress [Xerox, 1984] page description language has been used. The use of the Interpress Electronic Printing Standard [Xerox, 1986] (for a summary, refer to Bhushan and Plass [1986]) allows Nectarine to take advantage of all the Cedar imaging tools available for Interpress.

Interpress is a standardized page description language. More precisely, it is an interchange standard which specifies a set of instructions for print software to generate an image of a document page. "Interchange standard" refers to the property Interpress has of being independent of a particular environment or of characteristics of computers or printers. An Interpress master is a program that can be executed on any machine for which an Interpress interpreter—called a *decomposer*—is available. Specifying how to generate images instead of specifying the images themselves—e.g., describing a rectangle as an instruction to print a rectangle of a given dimension at a given position, instead of a bitmap and some printer control commands—leaves an Interpress master independent of such idiosyncrasies as resolution and control commands.

6.2. Nectarine's Program Structure

The program is structured in three modules: the user interface, the color blending implementation, and the main module which calls the corner stitching package to tessellate the layout and subsequently calls the Interpress package to produce masters to submit to the various print servers. Nectarine makes five passes over the data.

In the first pass, the VLSI image is traversed to identify all fonts used and to determine the outline print fonts corresponding to the bitmap screen fonts. This information is employed to construct the preamble of the Interpress master.

In the second pass, all intersections are found and the image plane is decomposed into a tessellation using the corner stitching algorithm. The rectangles are assigned numbers according to the layers intersecting in them. The color table, in which the intersections are classified by their resulting color, is created.

In the third pass, the color table is traversed and a color (from the color table, the entry representing the particular intersections occurring in the rectangle class) is blended for each entry when the simple method described in section 5.1 is employed. Instead, when discriminable colors are utilized, these are read from a palette stored on the disk. It is in this pass that color corrections to improve "printability" are performed.

In the fourth pass, the tessellation is traversed. Each rectangle's color is looked up in the color table that was prepared in the third pass and a corresponding colored rectangle is painted into the Interpress master. In the case of layout, regions are outlined with a black border.

In the fifth and last pass, the annotations are handled. As shown in plate III, painting them into the Interpress master in a separate last pass makes them appear superimposed on the marked paper. The Interpress standard allows one to set a so-called "priority toggle" which enforces the model shown in the first example of plate I. This is described in detail in the Interpress standard [Xerox, 1986, section 4.1] and in the paper by Warnock and Wyatt [1982].

Where appropriate, caches are maintained to avoid recomputations of the same data in successive invocations of the Nectarine command.

6.3. User Interface

In concept, including a particular symbol from a VLSI image into a specific location of a text document is the same operation as copying a portion of text from one document to another. In Tioga, Cedar's text editor [Swinehart et al., 1986, p. 451], text is copied by selecting the destination position and the source text, and applying the copy command.

A paradigm of Cedar user interfaces is the *control panel.* This is a member of a window class containing collections of menu buttons. The user executes a command by clicking a mouse button when the cursor is located on the menu button of the desired command. Figure 7 shows the control panel ChipNDale associates with each VLSI image.

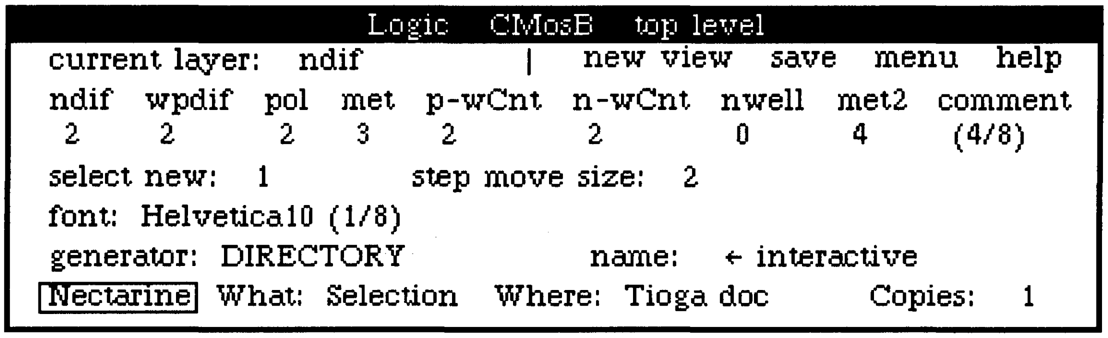

```
                    Logic   CMosB   top level
current layer:  ndif          |  new view   save   menu   help
ndif  wpdif  pol  met  p-wCnt  n-wCnt  nwell  met2  comment
 2     2      2    3    2       2       0      4     (4/8)
select new:  1        step move size:  2
font:  Helvetica10 (1/8)
generator:  DIRECTORY             name:   ← interactive
Nectarine  What:  Selection   Where:  Tioga doc      Copies:   1
```

Figure 7. A ChipNDale control panel. Nectarine uses the last line to provide a triggering menu button and three simple questions with the current answers. The user can flip through the possible answers by clicking with the mouse over the questions.

Nectarine uses the last line in the ChipNDale control panel. The three buttons labelled *What*, *Where*, and *Copies* correspond to the three parameters discussed in section 3.2. Since the set of values that the parameters can assume is known in advance, a new class of menu buttons has been implemented. This class is called *flip button*, because when the user clicks a mouse button over one of them, the answer flips cyclically through the set of possible values. The values of the parameters are retained until they are changed explicitly, because such a change is required infrequently.

Once the parameters are specified, the copy command can be triggered. In figure 7, the highlighted menu button, which is labelled *Nectarine* and outlined with a box, is the menu button that actually triggers action. This button is referred to as the *action* button.

In section 3.3, the need of so-called "advanced" functions hidden to the casual user was postulated. They have been implemented as additional parameters, which have well-behaved defaults. The fields for these parameters are in a second line in the ChipNDale control panel, which becomes visible only when the panel is scrolled up. Casual users do not suspect that the control panel is scrollable, hence the user interface is not complicated by these advanced features.

6.4. Coalescing ChipNDale Images with Tioga Documents

As shown in figure 1, it should be possible to coalesce a ChipNDale image with a Tioga document by repetitively copying portions of the VLSI image at given text positions. At the same time, the image should be scaled to fit the text document.

Tioga documents are tree-structured, with each node approximately corresponding to a paragraph. Tioga allows the definition of special node classes, such as the display of Interpress masters on the screen.

When a node is to be displayed on the screen – or, more generally, formatted – the paint procedure from the node's class is called by Tioga. Using this mechanism, it would have been possible to insert ChipNDale data structures directly into the Tioga document, thus allowing the figures to be edited.

However, this would require that the VLSI editor be loaded each time a document containing such figures was examined. Since many people reading documents with illustrations from ChipNDale are not designers and, therefore, not running DA tools on their workstations, this was considered an undesirable solution.

Instead, the existing Interpress artwork class was improved. The adopted implementation allowed the examination of a document with ChipNDale illustrations on any machine for which an Interpress implementation exists. This method has been favorably accepted by the VLSI designers. Due to Nectarine's powerful user interface, the designers have chosen to keep figures in a separate ChipNDale database and to copy a figure again into a Tioga document whenever the figure is amended.

Coordinates in an Interpress master are fixed values expressed in meters, so that a master is always printed the same size. This presents a difficulty since the field of the page into which a Tioga document is rendered is not static, but determined from a style sheet and local properties at the time of formatting. In order to reflect this flexibility in an illustration copied with the aid of Nectarine, Nectarine places program instructions in the information for the formatter, instead of associating static formatting information (such as a bounding box) as a property of the figure.

These instructions access the formatter's state information to determine the current margins. The known size information of the illustration is employed to compute a scaling factor such that the illustration is fitted flush to the local margins of the document. This scaling factor is in turn used to modify the formatter's state information to achieve the desired effect. In particular, if a document is printed with a different number of columns per page, the illustrations do not have to be regenerated.

This mechanism has required only a minor change in the Interpress artwork class, yet producing active documents in this way is a major timesaver for authors. The reason the implementation of this improvement required only a minor programming effort is that Cedar makes available an interpretive language that can be used to implement formatting procedures in artwork classes. This simple stack-oriented language is called *JaM* [Warnock and Newell, 1980]. JaM uses a postfix notation and is similar to Interpress.

JaM interprets a stream of tokens by placing those that are data on a stack; the tokens that are instructions are executed operating on the data in this stack. Since a Tioga document as seen by a formatter is such a stream, all that needs to be done is to insert appropriate JaM instructions in this stream.

6.5. Local Fidelity in Schematics

In the applications described in this report, the size of each printed rectangle is more important than its absolute position. For instance, in schematics, a bus is represented by a family of parallel straight-line segments. For the designer, it is important that the width of each segment of the family is represented with the same number of pixels. This is achieved by translating the origin to the lower left corner of the tile before painting it, because the Interpress coordinate system always has its origin on a grid point (see "Coordinate precision" [Xerox, 1986, p. 47] for more details). The procedure *PrintTile*, which is called for every tile when the tessellation is enumerated to paint tiles into the master, has the following simple structure for a rectangle of width *w* and height *h*, with its lower left vertex at (x_1, y_1):

```
if the color is different from the previous one then set new color
Interpress.SetXY ($x_1$, $y_1$)
Interpress.Trans () (* resets the origin to the current position *)
Interpress.MaskRectangle (0.0, 0.0, w, h)
```

6.6. Fine Tuning the Color Printer Operation

In section 5, the values of the cyan, magenta, yellow, and black components of a color were considered to be real values in the interval [0, 1]. It is clear how to produce the binary values 1 and 0: ink and no ink. How are the intermediate grey values generated?

It might be of interest to learn first how the RGB signals fed into the color monitor are generated in most of today's workstations, since this is relevant for continuous tone printing. From an electrical point of view, the different voltage values that can be applied to an electron gun above the noise level can be expressed with 8 bits of information. For an RGB image, this would require 24 bits. However, on a color monitor, only 7000 colors can be discriminated when displayed in non-adjacent fields [Murch, 1987].

To avoid waste in memory space and bandwidth, digital-to-analog converter-chips comprise a look-up table, loaded by software, that allows the indexing of a predeclared 24 bit RGB triplet with an 8 bit index. These chips also perform the γ-correction. The luminance of the phosphors is not proportional to the voltage applied to an electron gun. Proportionality is achieved by taking the γ-th root from the tristimulus values, where $\gamma \in [2.5, 3.0]$ [Meyer and Greenberg, 1987]. A better model is to consider the logarithms of luminance and voltage, so that γ becomes the slope of a curve related to the tone reproduction curve. Images displayed with a low γ are perceived to have "pastel colors," while those displayed with a high γ are perceived to have highly saturated colors.

Most printers do not permit continuous tone printing. An additive model is used to mix the colors of the inks with the white of the paper. This additive model combines with the subtractive model of the ink mixing.

One technique, called *halftoning*, consists in using ink spots of variable area. Conceptually, for each of the four color separations (cyan, magenta, yellow, and black) the image is subdivided in a raster of square bins, which is called a *halftone screen*. For each bin, a spot with area proportional to the lightness of the color is deposited on the film or paper.

The subsequent printing of the four separations causes the screens to produce interference patterns, known in the printing industry as *moiré patterns*. The solution is to rotate the four halftone screens by different angles, a technique called *angling* [Strauss, 1967]. The interference patterns depend on the lightness distribution in the image, thus there is no universal set of values for angling. The standard angles are such that moiré patterns are minimized for flesh tones [Southworth, 1979].

The colors for VLSI layout are not related to flesh tones, hence the standard angling will always create moirê patterns. One solution is to find a set of angles for every printer used, a task which is not hard, given that the palette of discriminable colors for VLSI layout presented in section 5.3 contains only 25 colors. A more general solution is to distribute the lightnesses of the colors uniformly in the palette, so that the additive effect is pronounced, and distinct moirê patterns are formed. Since VLSI layout is examined with a relatively large angular subtense, the moirê patterns can actually improve the readability.

Many digital printers cannot produce ink spots of variable area. On such printers, a group of pixels can be used to approximate a dot. This technique is called *halftone approximation* [Foley and Van Dam, 1982]. Accurate algorithms have been devised to obtain the same halftoning quality as in traditional methods [Holladay, 1986]. Unfortunately, halftone approximation substantially reduces the resolution of the printer. For instance, if the printer has a resolution of 1200 pixels per inch and 12 pixels are used to approximate a dot, the resulting resolution is 100 dots per inch.

While such a resolution may be sufficient for realistic images, it is often not acceptable for VLSI layout. Wires can be thinner than a spot of such size, hence the wires become fuzzy and very hard to track. Distributing the lightnesses of the colors in the palette uniformly in each color separation allows the use of smaller groups of pixels per dot, thus increasing the device resolution.

The fuzziness of the wires can be further reduced by using a halftoning algorithm where, for increasing lightness, the pixels are "turned on" from left to right and top to bottom in a regular pattern. If a 3 × 3 pixel matrix is used for each dot, the following threshold matrices have been used in conjunction with the discriminable colors presented in section 5.3:

	.4 .5 .6		.6 .1 .7		.5 .8 .3		.1 .6 .4
C:	.1 .3 .2	M:	.5 .3 .8	Y:	.9 .2 .6	K:	.7 .2 .9
	.7 .8 .9		.4 .2 .9		.1 .7 .4		.5 .8 .3

Such dots or, more appropriately, *cells* have the advantage of degrading gracefully in images with elements of sub-dot size when they are observed at large magnifications. The disadvantage is that putting marks of this size on paper is much more difficult than for the traditional halftone dots. For most inks, the image will look smeary, because not all pixels will receive the same amount of ink. A careful choice of the color palette is mandatory. In particular, these cells are not suitable for realistic images.

This technique still falls in the category of halftoning for full-color printing; it is not to be confused with the "stipple technique" [Trimberger, 1987], which was developed centuries ago to render coats of arms in black-and-white or relief. A stipple pattern represents a given color or VLSI layer, and its tristimulus value cannot be changed. Intersections of stipple patterns are hard to predict and they require a multi-color process to print. While they play an important role for rendering VLSI layout on black-and-white plotters, they are an anachronism in the world of color printing.

For applications in which an even larger resolution is needed, such as illustrating books with images of entire chips, line art can be used. The separations are then printed in multi-color instead of full-color, avoiding halftoning. In conjunction with a high resolution printer [Starkweather, 1985], this allows the printing of VLSI layout at a resolution of $\lambda = 31.75\ \mu$m, in which even the layout of a large chip is readable with the use of a good magnifying glass when it is printed the size of a book page.

Using multi-color printing, it is desirable to use only four separations to control printing costs, because non-opaque quality inks are required. The following colors have yielded good results:

polysilicon: RT6 red

diffusion: GT4 green

metal-1: C28 blue

metal-2: M35/2 magenta

Cuts and vias can be imaged in three of the separations to appear black. At the scale at which line art becomes interesting, the other layers, such as wells, can be omitted without hampering readability for a VLSI designer.

7. Conclusions

Nectarine has changed the way designers work in the Cedar environment. Circuits were previously described using a hardware description language, while today the designer's office walls are covered with block diagrams and schematics. The few project document specimens were done awkwardly using plain English text. A technical writer used to spend days with a calculator and a measuring ruler trying to insert very simple illustrations into Tioga documents. Today, the projects are characterized by comprehensive and easy-to-read documentation with many illustrations, and are printed with professional quality.

Project meetings have become more effective. Before the availability of Nectarine, details of circuits were painted *ad hoc* on a whiteboard from memory – frequently with errors that had far-reaching effects. Nowadays, dossiers are circulated and discussed, while diagrams of the system and the subsystems are pinned on the walls.

Nectarine is a good example of serendipity. None of the described benefits had been anticipated in the project, which started with the goal of implementing a quick hack to submit checkplots automatically to print servers whenever a chip was assembled. Instead of immediately writing the required code, actual user's needs were investigated, indicating that the missing tool in the system was not a hack to submit checkplots but a tool that knew how to image IC schematics and layout.

The use of the Interpress standard has allowed us to take advantage of a large portion of existing imaging code. Also, the images can be transferred to a variety of machines, which would not be the case if the print files were bitmaps or if a VLSI editor was needed to examine and manipulate figures in a document editor.

The clean factorization of the code, which resulted in the five pass implementation organized in three modules, has made it easy to enhance Nectarine. New applications, such as printing only one layer (overglass) to obtain pad-bonding specifications from a chip layout, or printing at a given scale to produce films for printed circuit boards, could be implemented in just a few hours.

To handle massive inputs, Nectarine processes the ChipNDale images in horizontal strips called *bands*. The bands do not overlap, forming a geometrical decomposition of the image. This means that Nectarine's five passes could be executed in parallel on independent data exploiting a MIMD (multiple instruction multiple data) workstation with coarse granularity. The data pertinent to a band would be pipelined through a sequence of processors, each executing static code. On bus-oriented architectures, this should avoid the problem of excessive bus traffic due to the cache inconsistencies observed in task-queue implementations. This speculation has yet to be proven experimentally.

Acknowledgments

Remember your users, for they will certainly remember you. (R. Atkinson)

I thank the Manager of the Design and Architecture Area, Bryan Preas, and acknowledge contributions by Lissy Bland, Neil Gunther, Jeff Hoel, Christian Jacobi, Jack Kent, Giuliana Lavendel, Subhana Menis, Hal Murray, Ken Pier, Michael Plass, Maureen Stone, and Doug Wyatt. The designers in the Dragon project were tolerant in accepting the initial long running times; their penetrating comments provided constructive feedback.

References

[Beretta and Meier, 1986]

Giordano Beretta and Andreas Meier: *Scan Converting Polygons Based on Plane-Sweep*, Technical Report 68, Institut für Informatik, Eidgenössische Technische Hochschule Zürich, Switzerland, August 1986.

[Bhushan and Plass, 1986]

Abhay Bhushan and Michael Plass: "The Interpress Page and Document Description Language," *Computer*, **18**, 6, June 1986.

[Bruno, 1983]

Michael H. Bruno, editor: *Pocket Pal, A Graphic Arts Production Handbook, Thirteenth Edition*, International Paper Company, New York, NY, 1983.

[Card et al., 1983]

Stuard K. Card, Thomas P. Moran, and Allen Newell: *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ, 1983.

[Foley and Van Dam, 1982]

James D. Foley and Andries Van Dam: *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, MA, 1982.

[Hagmann, 1986]

Robert B. Hagmann: "Sharing Processing Power in a Workstation Environment," in *Sixth International Conference on Distributed Computing Systems*, 260 - 267, IEEE Computer Society Press, Washington, DC, May 1986.

[Holladay, 1986]

Thomas M. Holladay: "An Optimum Algorithm for Halftone Generation for Displays and Hard Copies," *Proceedings of the Society for Information Display*, **21**, 2, 185-192, Palisades Institute for Research, Inc., New York, NY, May 1986.

[Hubel and Wiesel, 1968]

David H. Hubel and Torsten N. Wiesel: "Receptive Fields and Functional Architecture of Monkey Striate Cortex," *Journal of Physiology*, **195**, 215-243, 1968.

[Hubel et al., 1978]

David H. Hubel, Torsten N. Wiesel, and Michael P. Stryker: "Anatomical Demonstration of Orientation Columns in Macaque Monkey," *Journal of Comparative Neurology*, **177**, 361-380, 1978.

[Merrifield, 1987]

Robin M. Merrifield: "Visual Parameters for Color CRTs," in *Color and the Computer*, H. J. Durrett ed., Academic Press, Inc., Boston, MA, 1987.

[Mead and Conway, 1978]

Carver Mead and Lynn Conway: *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Reading, MA, 1978.

[Meyer and Greenberg, 1987]

Gary W. Meyer and Donald P. Greenberg: "Perceptual Color Spaces for Computer Graphics," in *Color and the Computer*, H. J. Durrett ed., Academic Press, Inc., Boston, MA, 1987.

[Moran, 1980]

Thomas P. Moran: "A Framework for Studying Human-Computer Interaction," in *Methodology of Interaction*, R. A. Guedj et al. eds., North-Holland Publishing Company, Amsterdam, The Netherlands, 1980.

[Murch, 1987]

Gerald Murch: "Color Displays and Color Science," in *Color and the Computer*, H. J. Durrett ed., Academic Press, Inc., Boston, MA, 1987.

[Newman, 1980]

William M. Newman: "Some Notes on User Interface Design," in *Methodology of Interaction*, R. A. Guedj et al. eds., North-Holland Publishing Company, Amsterdam, The Netherlands, 1980.

[Newman and Sproull, 1979]

William M. Newman and Robert F. Sproull: *Principles of Interactive Computer Graphics, Second Edition*, McGraw-Hill Book Company, New York, NY, 1979.

[Nievergelt, 1982]

Jürg Nievergelt: "Errors in Dialog Design and How To Avoid Them," in *Document Preparation Systems*, J. Nievergelt, G. Coray, J.-D. Nicoud and A. C. Shaw eds., North-Holland Publishing Company, Amsterdam, The Netherlands, 1982.

[Nievergelt and Preparata, 1982]

Jürg Nievergelt and Franco P. Preparata: "Plane-Sweep Algorithms for Intersecting Geometric Figures," *Communications of the ACM*, **25**, 10, 739 - 747, 1982.

[Nobes, 1985]

Nigel Nobes: "Computer Aided Design Systems: An Integrated Approach to Technical Publishing," in *Computer Graphics '85: Proceedings of the International Conference held in London*, 45 - 54, Online Publications, Pinner, Mddx., England, October 1985.

[Ousterhout, 1984]

John K. Ousterhout: "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools," *IEEE Transactions on Computer Aided Design*, **CAD-3**, 1, January 1984.

[Schwarz et al., 1987]

Michael W. Schwarz, William B. Cowan, and John C. Beatty: "An Experimental Comparison of RGB, YIQ, LAB, HSV and Opponent Color Models," *ACM Transactions on Graphics*, **6**, 2, April 1987.

[Shand, 1985]

Mark A. Shand: *Algorithms for Corner Stitched Data Structures*, Technical Report 268, Basser Department of Computer Science, the University of Sydney, N.S.W. Australia, August 1985.

[Silverstein, 1987]

Louis D. Silverstein: "Human Factors for Color Display Systems: Concepts, Methods, and Research," in *Color and the Computer*, H. J. Durrett ed., Academic Press, Inc., Boston, MA, 1987.

[Southworth, 1979]

Miles Southworth: *Color Separation Techniques, Second Edition*, Graphic Arts Publishing, Livonia, NY, 1979.

[Starkweather, 1985]

Gary K. Starkweather: "A High Resolution Laser Printer," *Journal of Imaging Technology*, **11**, 6, 300-305, January 1985.

[Strauss, 1967]

Victor Strauss: *The Printing Industry, An Introduction to its Many Branches, Processes and Products*, Printing Industries of America, Inc., Washington, DC, 1967.

[Swinehart et al., 1986]

Daniel C. Swinehart, Polle T. Zellweger, Richard J. Beach, and Robert B. Hagmann: "A Structural View of the Cedar Programming Environment," *ACM Transactions on Programming Languages and Systems*, **8**, 4, October 1986.

[Trimberger, 1987]
Stephen M. Trimberger: *An Introduction to CAD for VLSI*, Kluwer Academic Publishers, Boston, MA, 1987.

[Warnock and Newell, 1980]
John Warnock and Martin Newell: *JaM Manual.* Unpublished internal report, Xerox Palo Alto Research Center, Palo Alto, CA, 1980.

[Warnock and Wyatt, 1982]
John Warnock and Douglas K. Wyatt: "A Device Independent Graphics Imaging Model For Use with Raster Devices," *Computer Graphics*, **16**, 3, July 1982 (the same as the SIGGRAPH 82 proceedings).

[Wyszecki and Stiles, 1982]
Günter Wyszecki and W. S. Stiles: *Color Science: Concepts and Methods, Quantitative Data and Formulae, Second Edition*, John Wiley & Sons, New York, NY, 1982.

[Xerox, 1984]
Xerox Corporation: *Introduction to Interpress*, XSIG 038404, Stamford, CT, April 1984.

[Xerox, 1986]
Xerox Corporation: *Interpress Electronic Printing Standard*, Version 3.0, XNSS 048601, Stamford, CT, January 1986.

Index	RGB			xy		L*a*b*			
30	0.7882	0.5255	0.9529	0.3203	0.3105	86.88	11.93	-5.90	met2 pdif
31	0.9569	0.9137	0.1216	0.3287	0.3750	86.16	-12.02	17.33	met2 pol pdif
32	0.9569	0.9137	0.1216	0.3287	0.3750	86.16	-12.02	17.33	met2 pol nwel pdif
33	0.1647	0.7451	0.8196	0.2861	0.3258	85.54	-11.68	-5.92	met nwel
34	0.1647	0.7451	0.8196	0.2861	0.3258	85.54	-11.68	-5.92	met
35	0.4431	0.6039	0.8941	0.3031	0.3156	85.21	1.20	-6.76	met2 met nwelCont
36	0.3686	0.8549	0.4863	0.2950	0.3497	85.15	-17.28	3.41	met ndif
37	0.3686	0.8549	0.4863	0.2950	0.3497	85.15	-17.28	3.41	met nwel pdif
38	0.3686	0.8549	0.4863	0.2950	0.3497	85.15	-17.28	3.41	met pdif
39	0.5451	0.9686	0.1569	0.3038	0.3782	84.70	-23.93	13.78	nwel pdif
40	0.5451	0.9686	0.1569	0.3038	0.3782	84.70	-23.93	13.78	green
41	0.5451	0.9686	0.1569	0.3038	0.3782	84.70	-23.93	13.78	ndif
42	0.4824	0.7725	0.5373	0.3031	0.3425	84.66	-10.51	2.35	met2 met ndif
43	0.4824	0.7725	0.5373	0.3031	0.3425	84.66	-10.51	2.35	met2 met pdif
44	0.8314	0.8902	0.1216	0.3234	0.3760	84.46	-14.44	16.41	met pol nwel pdif
45	0.8314	0.8902	0.1216	0.3234	0.3760	84.46	-14.44	16.41	met pol pdif
46	0.1490	0.9569	0.3490	0.2795	0.3652	83.94	-30.25	5.79	nwelCont nwel
47	0.1490	0.9569	0.3490	0.2795	0.3652	83.94	-30.25	5.79	pwelCont
48	0.5765	0.5137	0.8314	0.3129	0.3134	82.89	6.73	-5.83	met2 met
49	0.5529	0.5686	0.7176	0.3113	0.3219	82.42	2.19	-3.18	met2 nwelCont nwel
50	0.5529	0.5686	0.7176	0.3113	0.3219	82.42	2.19	-3.18	met2 nwelCont
51	0.9137	0.2667	0.9765	0.3356	0.2938	81.17	26.14	-9.02	met2 nwel
52	0.1765	0.6706	0.4431	0.2869	0.3451	77.27	-17.81	0.58	met nwelCont nwel
53	0.1765	0.6706	0.4431	0.2869	0.3451	77.27	-17.81	0.58	met pwelCont
54	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 met
55	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 met2 ndif
56	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 met nwel pdif
57	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 met nwel
58	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 nwel pdif
59	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 met2 nwel pdif

Colors for some layer combinations. RGB values: displayed on monitor; L*a*b* values: printed on Versatec plotter.

Index	RGB			xy		L*a*b*			
0	0.0000	0.0000	0.0000	0.3143	0.3300	0.00	0.00	0.00	cut met2 nwel pdif
1	1.0000	0.1333	0.1020	0.4198	0.3358	58.84	30.62	15.82	red
2	1.0000	0.2392	0.2667	0.3779	0.3286	67.61	23.40	8.75	met pol
3	0.8588	0.2706	0.4157	0.3550	0.3191	70.34	19.79	2.65	met2 pol nwel
4	0.7608	0.2706	0.4784	0.3457	0.3143	70.51	18.33	-0.18	met2 met pol
5	0.0431	0.2549	0.9255	0.2880	0.2831	70.94	8.31	-18.00	cut2 nwel
6	0.1765	0.6706	0.4431	0.2869	0.3451	77.27	-17.81	0.58	met nwelCont
7	0.9137	0.2667	0.9765	0.3356	0.2938	81.17	26.14	-9.02	met2
8	0.5529	0.5686	0.7176	0.3113	0.3219	82.42	2.19	-3.18	met2 pwelCont
9	0.5765	0.5137	0.8314	0.3129	0.3134	82.89	6.73	-5.83	met2 met nwel
10	0.8275	0.8431	0.1529	0.3247	0.3719	83.61	-12.27	15.24	met2 met pol nwel pdif
11	0.1490	0.9569	0.3490	0.2795	0.3652	83.94	-30.25	5.79	nwelCont
12	0.8314	0.8902	0.1216	0.3234	0.3760	84.46	-14.44	16.41	met pol ndif
13	0.4824	0.7725	0.5373	0.3031	0.3425	84.66	-10.51	2.35	met2 met nwel pdif
14	0.5451	0.9686	0.1569	0.3038	0.3782	84.70	-23.93	13.78	pdif
15	0.3686	0.8549	0.4863	0.2950	0.3497	85.15	-17.28	3.41	green blue
16	0.4431	0.6039	0.8941	0.3031	0.3156	85.21	1.20	-6.76	met2 met pwelCont
17	0.1647	0.7451	0.8196	0.2861	0.3258	85.54	-11.68	-5.92	blue
18	0.9569	0.9137	0.1216	0.3287	0.3750	86.16	-12.02	17.33	met2 pol ndif
19	0.8039	0.7843	0.4706	0.3196	0.3462	86.67	-4.58	6.40	met2 met pol ndif
20	0.7882	0.5255	0.9529	0.3203	0.3105	86.88	11.93	-5.90	met2 ndif
21	1.0000	1.0000	0.0000	0.3297	0.3872	86.98	-16.27	21.57	yellow
22	0.4824	0.6824	0.9020	0.3029	0.3197	87.72	-0.76	-5.49	met2 met nwelCont nwel
23	1.0000	1.0000	0.7020	0.3174	0.3416	96.43	-3.95	4.89	nwel
24	1.0000	1.0000	0.0000	0.3297	0.3872	86.98	-16.27	21.57	green red
25	1.0000	1.0000	0.0000	0.3297	0.3872	86.98	-16.27	21.57	gate
26	1.0000	1.0000	0.0000	0.3297	0.3872	86.98	-16.27	21.57	pol ndif
27	1.0000	1.0000	0.0000	0.3297	0.3872	86.98	-16.27	21.57	pol pdif
28	1.0000	1.0000	0.0000	0.3297	0.3872	86.98	-16.27	21.57	pol nwel pdif
29	0.7882	0.5255	0.9529	0.3203	0.3105	86.88	11.93	-5.90	met2 nwel pdif

Colors for some layer combinations. RGB values: displayed on monitor; L*a*b* values: printed on Versatec plotter.

Colophon

This report was produced by the author using exclusively an experimental Interpress 3.0 implementation in Cedar. The text was composed in Tioga. The chromaticity diagrams were created with Meta-Palette. Except for plates V and XVI, the remaining artwork was created in ChipNDale and coalesced with the text using Nectarine. The camera-ready copies for the text and the separations on film for the color plates were produced on the Erie laser printer. The separations for plate V were produced by Kedie/Orent Inc. in Sunnyvale (California) on a Crosfield scanner from a micro-photograph and those for plate XII from a Versatec plot. Composition of the color pages was performed with a custom built program that electronically pastes up Interpress masters, while the separations for the form containing the color pages have been stripped by Kedie/Orent, Inc.

Muller Printing Co. in Santa Clara, California, printed the color plates with a five-color 40" Heidelberg press on 70 lb. Productolith Gloss Book paper using Thomas inks in the SOP colors for cyan, magenta, yellow, and black. Fong Brothers Printing in Brisbane, California, printed the text with an A.B. Dick 1600 System on 60 lb. White Vellum paper and bound the reports.

Renêe Butruce of Xerox, Nancy Bryant of Fong Brothers Printing, Charles Perry of Muller Printing, Don Sheridon and Fran Stout of Kedie/Orent were instrumental in making the production possible. The Manager of the Electronic Document Laboratory, Rick Beach, encouraged the experiment of producing a technical report in color. Tim Diebert and Dan Murphy of Xerox gave the author some useful advice on the printing industry.

XEROX

A New Approach to Imaging IC Layout and Schematics   Giordano Bruno Beretta

EDL-88-3