Magic Maintainer's Manual #3: Display Styles, Colormaps, and Glyphs

Robert N. Mayo
John Ousterhout

Computer Science Division
Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720

This tutorial corresponds to Magic version 7.

Tutorials to read first:

All of them.

Commands introduced in this tutorial:

(None)

Macros introduced in this tutorial:

(None)

1 Introduction

This document gives overall information about the files that tell Magic how to display information on the screen. There are three types of files that contain display information: display styles files, color-map files, and glyph files.

2 Display Styles

Display styles files describe how to draw rectangular areas and text. A single file contains a large number of display styles. Each display style contains two kinds of information: a) how to modify pixels (which bits of the pixel should be changed and what their new value(s) should be); and b) which pixels to modify. Part b) consists of things like "fill the entire area," or "modify only those pixels in the area that are given by a particular stipple pattern," or "draw a dashed-line around the area's outline." In the case of text, "which pixels to modify" is determined by the font for the text, which is not part of the display style, so the display style information for this is ignored. See the manual page **dstyle (5)** for details on the format of display styles files.

Display styles are designed to take into account both the characteristics of certain technologies and the characteristics of certain displays. For example, a bipolar process may require information to be displayed very differently than a MOS process, and a black-and-white display will be used much differently than a color display. Thus there can be many different display styles files, each corresponding to a particular class of technologies and a class of displays. The names of styles files reflect these classes: each display styles file has a name of the form ***x.y*.dstyle5**, where *x* is the technology class (given by the **styletype** line in the **styles** section of the technology file), and *y* is the class of display. Each display driver knows its display class; the driver initialization routine sets an internal Magic variable with the display class to use. Right now we have two display styles files: **mos.7bit.dstyle5** and **mos.bw.dstyle5**. Both files contain enough different styles to handle a variety of MOS processes, including both nMOS and CMOS (hence the **mos** field). **Mos.7bit.dstyle5** is designed for color displays with at least seven bits of color per pixel, while **mos.bw.dstyle5** is for black-and-white displays (stipple patterns are used instead of colors).

3 Color Maps

The display styles file tells how to modify pixels, but this doesn't completely specify the color that will be displayed on the screen (unless the screen is black-and-white). For color displays, the pixel values are used to index into a *color map*, which contains the red, green, and blue intensity values to use for each pixel value. The values for color maps are stored in color-map files and can be edited using the color-map-editing window in Magic. See **cmap (5)** for details on the format of color-map files.

Each display styles file uses a separate color map. Unfortunately, some monitors have slightly different phosphors than others; this will result in different colors if the same intensity values are used for them. To compensate for monitor differences, Magic supports multiple color maps for each display style, depending on the monitor being used. The monitor type can be specified with the **-m** command line switch to Magic, with **std** as the default. Color-map files have names of the form *x.y.z* **.cmap1**, where *x* and *y* have the same meaning as for display styles and *z* is the monitor type. Over the last few years monitor phosphors appear to have standardized quite a bit, so almost all monitors now work well with the **std** monitor type. The color map **mos.7bit.std.cmap1** is the standard one used at Berkeley.

4 Transparent and Opaque Layers

One of the key decisions in defining a set of display styles for a color display is how to use the bits of a pixel (this section doesn't apply to black-and-white displays). One option is to use a separate bit of each pixel (called a *bit plane*) for each mask layer. The advantage of this is that each possible combination of layer overlaps results in a different pixel value, and hence a different color (if you wish). Thus, for example, if metal and poly are represented with different bit planes, poly-without-metal, metal-without-poly, poly-and-metal, and neither-poly-nor-metal will each cause a different value to be stored in the pixel. A different color can be used to display each of these combinations. Typically, the colors are chosen to present an illusion of transparency: the poly-and-metal color is chosen to make it appear as if metal were a transparent colored foil placed on top of poly. You can see this effect if you paint polysilicon, metal1, and metal2 on top of each other in our standard technologies.

The problem with transparent layers is that they require many bits per pixel. Most color displays don't have enough planes to use a different one for each mask layer. Another option is to use a group of planes together. For example, three bits of a pixel can be used to store seven mask layers plus background, with each mask layer corresponding to one of the combinations of the three bits. The problem with this scheme is that there is no way to represent overlaps: where there is an overlap, one of the layers must be displayed at the expense of the others. We call this scheme an *opaque* one since when it is used it appears as if each layer is an opaque foil, with the foils lying on top of each other in some priority order. This makes it harder to see what's going on when there are several mask layers in an area.

The display styles files we've designed for Magic use a combination of these techniques to get as much transparency as possible. For example, our **mos.7bit.dstyle5** file uses three bits of the pixel in an opaque scheme to represent polysilicon, diffusion, and various combinations of them such as transistors. Two additional bits are used, one each, for the two metal layers, so they are transparent with respect to each other and the poly-diff combinations. Thus, although only one poly-diff combination can appear at each point, it's possible to see the overlaps between each of these combinations and each combination of metal1 and metal2. Furthermore, all of these styles are overridden if the sixth bit of the pixel is set. In this case the low order five bits no longer correspond to mask layers; they are used for opaque layers for things like labels and cell bounding boxes, and override any mask information. Thus, for example, when metal1 is displayed it only affects one bit plane, but when labels are displayed, the entire low-order six bits of the pixel are modified. It's important that the opaque layers like labels are drawn after the transparent things that they blot out; this is guaranteed by giving them higher style numbers in the display styles files.

Finally, the seventh bit of the pixel is used for highlights like the box and the selection. All 64 entries in the color map corresponding to pixel values with this bit set contain the same value, namely pure white. This makes the highlights appear opaque with respect to everything else. However, since they have their own bit plane which is completely independent of anything else, they can be drawn and erased without having to redraw any of the mask information underneath. This is why the box can be moved relatively quickly. On the other hand, if Magic erases a label it must redraw all the mask information in the area because the label shared pixel bits with the mask information.

Thus, the scheme we've been using for Magic is a hierarchical combination of transparent and opaque layers. This scheme is defined almost entirely by the styles file, so you can try other schemes if you wish. However, you're likely to have problems if you try anything too radically different; we haven't tried any schemes but the one currently being used so there are probably some code dependencies on it.

For more information on transparent and opaque layers, see the paper "The User Interface and Implementation of an IC Layout Editor," which appeared in *IEEE Transactions on CAD* in July 1984.

5 Glyphs

Glyphs are small rectangular bit patterns that are used in two places in Magic. The primary use for glyphs is for programmable cursors, such as the shapes that show you which corner of the box you're moving and the various tools described in Tutorial #3. Each programmable cursor is stored as a glyph describing the pattern to be displayed in the cursor. The second use of glyphs is by the window package: the little arrow icons appearing at the ends of scroll bars are stored as glyphs, as is the zoom box in the lower-left corner of the window. We may eventually use glyphs in a menu interface (but don't hold your breath).

Glyphs are stored in ASCII glyph files, each of which can hold one or more glyph patterns. Each glyph is represented as a pattern of characters representing the pixels in the glyph. Each character selects a display style from the current display styles file; the display style indicates the color to use for that pixel. See the manual page **glyphs (5)** for details on the syntax of glyphs files.

The window glyphs are stored in files of the form **windows***XX***.glyphs**. The *XX* indicates how wide the glyphs are, and is set by the graphics driver for a particular display. We started out with a **windows7.glyphs** and a **windows11.glyphs**. Since then, display resolution has increased greatly so we have also created a **windows14.glyphs** and a **windows22.glyphs**. The positions of the various glyphs in these files is important, and is defined in the **window** module of Magic.

Programmable cursors are stored in files named *x***.glyphs**, where *x* is determined by the device driver for the display. Displays capable of supporting full-color cursors use **color.glyphs**; displays that can only support monochrome cursors used **bw.glyphs**. The order of the various glyphs in these files is important. It is defined by the files **styles.h** in the **misc** module of Magic.