Preparing Images for Display 6

To introduce you to programming with the XIL library, Chapter 2, "Basic XIL Program," presented an example program that read in an 8-bit grayscale image and displayed it in an X window created from one of four X visuals. This chapter looks at a revision of that example called `display`. The new example can display images of several data types and images of more than one band and was designed to illustrate how to handle some of the many display cases you may encounter.

The source files for the program can be found in the directory `$XILHOME/examples/display`. Table 6-1 list these source files and indicates what the code in each file does.

Table 6-1 Source Files for `display`

Source File	What the Code Does
`display.c`	Contains `main()`, which prepares different types of source images to be copied to various types of display images
`fileio.c`	Reads an image from a file and loads the image data into an XIL image
`window.c`	Creates the program's X colormap and manages that colormap

6

The first part of this chapter explains briefly how to build and run the `display` example. The bulk of the chapter then discusses the most important tasks performed in the example. These include:

- Converting a single-band image to a multiband image
- Converting an `XIL_SHORT` image to an `XIL_BYTE` image
- Converting an RGB image to a pseudocolor image and a colormap
- Converting a 24-bit image to a 1-bit image
- Converting an 8-bit image to a 1-bit image
- Displaying a 1-bit image on a monochrome display

The chapter concludes with a list of all the cases handled in the program.

Running the Sample Display Program

To run this example, follow these instructions:

1. **Change your working directory to** `$XILHOME/examples/display`**.**
2. **Build the program using the makefile in that directory.**
3. **Execute the program from the directory in which you built it, using the command line**

```
% display toys.header
```

The program will display an RGB image of a group of toys in an X window. To stop the program, move your pointer into the program's window, and click any mouse button.

Converting a Single-Band Image to a Multiband Image

The `display` program approaches this problem in two ways: by passing the source image through a lookup table and by replicating the source image in the multiple bands of the destination image.

Passing the Source Image Through a Lookup Table

If there are only a few different values in the source image—for example, when the source image is a 1-bit image—passing the source image through a lookup table is an effective method of performing this conversion. For instance, if all the values in the source are 0 or 1, you can use a lookup table with only two

entries: one that specifies the values to appear in each band of the destination for pixels that have a value of 0 in the source, and another that specifies the values to appear in each band of the destination for pixels that have a value of 1.

For example, the program uses this method to prepare a bit image for display in a 24-bit window. To make the source image compatible with the display image, the program converts the source image to a 3-band, 8-bit image. To do this, the example:

1. Creates a temporary image, `retained_image`, that has the same width and height as the source image, but is 3 bands deep and will contain `XIL_BYTE` data. (This temporary image will later be copied to the display.)
2. Creates the lookup table (a data structure of type `XilLookup`) shown in Figure 6-1.

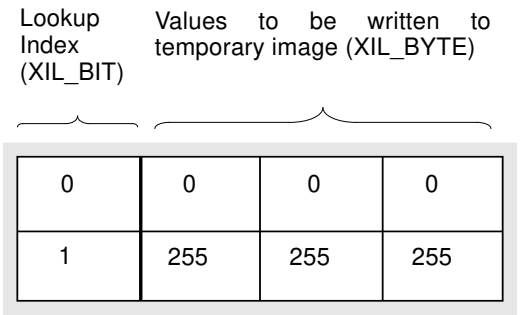

Figure 6-1 Lookup Table

3. For each pixel in the source image, looks up the proper value in the left column of the table and writes the three values to its right to the first, second, and third bands of the corresponding pixel in `retained_image`.

The code that implements these steps is shown below.

```
#define BITSIZE 0x2

XilImage src_image, retained_image;
XilLookup lookup;
Xil_unsigned8 lookupdata[] = {0, 0, 0, 255, 255, 255};

retained_image = xil_create(state, width, height, 3, XIL_BYTE);
lookup = xil_lookup_create(state, XIL_BIT, XIL_BYTE, 3,
    BITSIZE, 0, lookupdata);
xil_lookup(src_image, retained_image, lookup);
```

Replicating the Source Image in the Bands of the Destination

The second method the example uses to convert a single-band source image to a multiband destination image is to replicate the source image in each band of the destination. To perform this task using the XIL library, you must first create a child image representing each band of the destination and then copy the source to each child image. Because any changes made to an XIL child image affect the parent, these copies result in the source being copied to each band of the destination.

The display program converts a 1-band `XIL_BYTE` source image to a 3-band `XIL_BYTE` image using the code shown below.

```
XilImage image, retained_image, band0, band1, band2;

retained_image = xil_create(state, width, height, 3, XIL_BYTE);
band0 = xil_create_child(retained_image, 0, 0, width, height, 0,
     1);
band1 = xil_create_child(retained_image, 0, 0, width, height, 1,
     1);
band2 = xil_create_child(retained_image, 0, 0, width, height, 2,
     1);
xil_copy(image, band0);
xil_copy(image, band1);
xil_copy(image, band2);
```

Converting an XIL_SHORT Image to an XIL_BYTE Image

Converting an XIL_SHORT image to an XIL_BYTE image is a two-step process. First, you must rescale the values in the source image so that they fall in the range 0 to 255. Then, you must cast the values in the source image to values of type XIL_BYTE.

In the rescaling step, you could simply map the lowest possible value in the source (-32768) to the lowest possible value in the destination (0) and the highest possible value in the source (32767) to the highest possible value in the destination (255). However, you'll generally achieve better results by first finding the extrema in the source and then mapping the lowest value actually in the source to 0 and the highest value to 255.

The code that display uses to perform this task (in one instance) is shown below.

```
#define CMAPSIZE 256

float low[1], high[1];
float mult[1], offset[1];

xil_extrema(image, high, low);
mult[0] = (NCOLORS - 1) / (high[0] - low[0]);
offset[0] = -((low[0] * (NCOLORS - 1)) /
    (high[0] - low[0])) + (CMAPSIZE - NCOLORS);
xil_rescale(image, image, mult, offset);
xil_cast(image, retained_image);
```

The call to xil_extrema() stores the source image's highest value in high[0] and its lowest value in low[0]. Then, the call to xil_rescale() maps the lowest value in the image to 0 and the highest to 255. Once the values in image have been brought into the proper range, the example casts them to values of type XIL_BYTE. When casting a 16-bit value to an 8-bit value, xil_cast() preserves the 8 least significant bits of each input value, which is what the program requires.

6

Converting an RGB Image to an Indexed-Color Image and a Colormap

The `display` program prepares a true-color image to be shown on an indexed-color display by performing an ordered dither on the source image. The general procedure the example employs is to create a colorcube and a dither mask and then to call the function `xil_ordered_dither()`. After this operation, the program has a single-band version of the source image and a colorcube that defines the RGB values to be associated with each value in the indexed-color image. Before displaying the indexed-color image, the example creates a virtual X colormap and writes the values in the XIL colorcube to the X colormap.

Note – For a more detailed explanation of ordered dithering, see the section "Dithering an Image" on page 105.

The code the example uses to perform the ordered dither is shown below.

```
XilLookup cmap;
XilDitherMask mask;

cmap = xil_lookup_get_by_name(state, "cc496");
mask = xil_dithermask_get_by_name(state, "dm443");
set_colormap(xdisplay, xwindow, cmap);
retained_image = xil_create(state, width, height, 1, datatype);
xil_ordered_dither(image, retained_image, cmap, mask);
```

The colorcube `cc496` and the dither mask `dm443` are objects that the library creates when it is initialized. The colorcube is appropriate for dithering an RGB image to 216 colors (the indexes will range from 38 to 253), and `dm443` is a 4-by-4 dither mask.

Note – These objects could also have been created with the functions `xil_colorcube_create()` and `xil_dithermask_create()`.

Before the indexed-color image can be displayed, the color values in the colorcube `cmap` must be stored in colorcells 38 to 253 in the application's X colormap. This task is handled by the routine `set_colormap()`. Note that the colormap data that is being loaded into the X colormap is taken from the lookup table that was used as the colorcube for the dithering operation.

The XIL library also includes two other types of dithering operations that you can use in preparing a true-color image to be shown on an indexed-color display. The functions that perform these dithering operations are called `xil_nearest_color()` and `xil_error_diffusion()`. For further information about these functions, see the section “xil_nearest_color()” on page 108 and the section “xil_error_diffusion()” on page 114.

Converting a 24-Bit Image to a 1-Bit Image

The example uses a two-step process to convert an RGB image to a 1-bit image. The first step is to convert the 24-bit source image to an 8-bit grayscale image by extracting luminance information from the source image. Using the XIL library, you perform this task by converting the source image from the RGB color space to the Y color space. The destination image for this color-space conversion must be a single-band image.

The code `display` used to perform this color-space conversion is shown below.

```
image1 = xil_create(state, width, height, 1, XIL_BYTE);
xil_set_colorspace(image, xil_colorspace_get_by_name(state,
    “rgblinear”));
xil_set_colorspace(image1, xil_colorspace_get_by_name(state,
    “ylinear”));
xil_color_convert(image, image1);
```

The second step in the process is to dither the 8-bit image to a 1-bit image using either the function `xil_ordered_dither()` or `xil_error_diffusion()`. If execution speed is your primary concern, use the ordered-dither routine. To obtain the best quality image, use the error-diffusion function. For more information about these functions, see the sections “xil_ordered_dither()” on page 118 and “xil_error_diffusion()” on page 114.

The `display` example performs this task using `xil_error_diffusion()`.

```
#define BITSIZE 0x2

XilKernel distribution;
static int multipliers[1] = {-1};
static unsigned int dimensions[1] = {BITSIZE};

retained_image = xil_create(state, width, height, 1, XIL_BIT);
distribution = xil_kernel_get_by_name(state, "floyd-steinberg");
colormap = xil_colorcube_create(state, XIL_BIT, XIL_BYTE, 1, 0,
    multipliers, dimensions);
xil_error_diffusion(image1, retained_image, colormap,
    distribution);
```

Converting an 8-Bit Image to a 1-Bit Image

This task is the same as the second step described in the preceding section. You should dither the 8-bit image to a 1-bit image using either the function `xil_ordered_dither()` or `xil_error_diffusion()`. As noted above, the first of these functions is the fastest, but the second generally produces better looking results.

Displaying a 1-Bit Image on a Monochrome Display

The `display` program is set up to deal with `XIL_BIT` images in which 0's represent black pixels and 1's represent white pixels (the normal case). In the X colormap for monochrome displays supported by the XIL library, however, 0 represents white and a 1 represents black; therefore, the `display` program takes the one's complement of the source image (using `xil_not()`) before displaying it.

Types of Images Displayed

This section simply lists the cases handled by the `display` example so that you can quickly determine whether a case you're interested in is covered. In Table 6-2, the types of images the program can process are shown in the left

column, and the supported X visuals are listed in the top row. A check mark indicates that the program can display a particular type of image in an X window created using a particular X visual.

Table 6-2 Cases Handled by the `display` Program

	TrueColor 24 Bits	PseudoColor **8 Bits**	**GrayScale 8 Bits**	**StaticGray 8 Bits**	**StaticGray 1 Bit**
XIL_BIT 1 Band	✔		✔	✔	✔
XIL_BYTE 1 Band	✔			✔	✔
XIL_BYTE 3 Bands	✔	✔	✔		✔
XIL_SHORT 1 Band	✔		✔	✔	✔

Note – For an explanation of X visuals and their uses, see the *Xlib Programming Manual*.

6