Cell Codec 14

The Cell image compression technology, which was developed by Sun, has been optimized for the rapid decompression and display of images on simple hardware. Therefore, Cell compression is able to achieve reasonable display quality on indexed-color frame buffers. The initial focus of the Cell technology is on Sun-to-Sun communications, where the benefits of fast decoding outweigh the benefits of standards. Some possible areas of application include media distributions on CD-ROM and multimedia mail applications.

Note – To get an idea of the quality of an image that has been compressed and decompressed using the Cell compressor, see Color Plate 4.

This chapter is divided into four sections. First, it explains how the Cell compressor works and the type of applications it was designed for. Second, it explains briefly how to create a Cell CIS. Third, it discusses CIS attributes that apply specifically to the Cell codec (as opposed to the general CIS attributes covered in the section "General CIS Attributes" on page 257). Fourth, the chapter discusses some accelerated playback routines (called *molecules*) that you may be able to take advantage of when you're playing back movies.

14

How the Cell Codec Works

The Cell encoding process transforms individual video frames into a bytestream that is displayable with the Cell decompressor. Normally, the encoder works with RGB images; however, it can also handle XIL images whose color-space attribute is set to `ycc601` or `ycc709`. The decompressor always produces RGB images.

In the first step of the encoding process, video images are analyzed to produce an appropriate colormap to represent the frames to be encoded (unless the programmer has already specified one). This step allows the specification of the colormap size in order to leave colors unused. This strategy enhances cooperation with the window manager and other applications. Cell also provides for Adaptive Colormap Selection, in which a new colormap is generated when the current colormap becomes unsuitable. This colormap can be used in subsequent frames.

Given the images and the colormap, the second step is to encode the individual frames into a Cell bytestream. The basic coding scheme used in Cell encoding is based on an image coding method called Block Truncation Coding. A 4-by-4 region of pixels (a *cell*) from an image is represented by 2 colors and a 16-bit mask. See Figure 14-1.

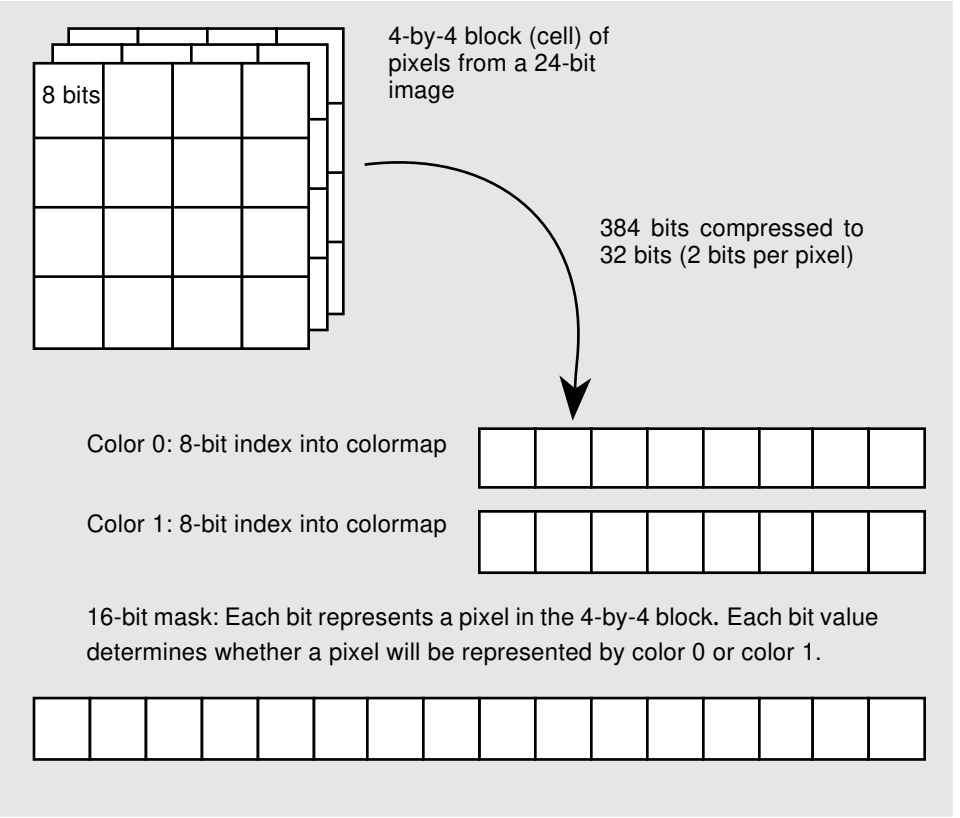

Figure 14-1 Cell Compression

The mask indicates which color to place at each of the pixel positions. The mask and colors may be chosen to maintain certain statistics of the cell, or they may be chosen to reduce contouring in a manner similar to ordered dithering.

The primary advantage of this coding method lies in the similarity of the decoding process to the operation of character fonting. The character display process for a color frame buffer takes as input a foreground color, a background color, and a mask that specifies which color to use at each pixel. Because this function is so important to the window system, it is often implemented as a display primitive in graphics accelerators. The Cell

compression technique uses this existing capability to provide full-motion video decoding with no special hardware or modifications to the window system.

There are actually two different encoding methods used to generate the encoded bytestream. One encoding method, known as Block Truncation Coding (BTC), chooses a 4-by-4 binary pixel mask and two colors (a foreground and a background color), while attempting to maintain the mean and variance of the luminance within the block. A second method, called the Dither technique, determines the pair of colors from the colormap that produce the least error when dithered over the 4-by-4 region. The BTC method is fastest and usually produces good results; the Dither method is slower, but is less likely to produce noticeable contours in regions of slow color variation.

Finally, encoded frames are combined into a frame sequence in the interframe compression step, where the compression ratio can be increased by anywhere from a factor of two to a factor of five. The colors and mask in each cell are compared to those used in the previous frames of the movie. If the colors and mask match to within a certain tolerance, a special skip code is generated. Runs of skip codes are combined to further reduce the bytestream. If either a single color changes or the mask changes, special escape codes are sent to update the changed data. Changes in the colormap are also detected by the interframe encoder, which causes special codes to be inserted into the stream to update the colormap.

Note – Details about the makeup of a Cell bytestream can be found in Appendix D, "Cell and CellB Bytestream Definitions."

Choosing a Colormap

The compressor chooses which colormap to use in encoding the current image in one of three ways. If Adaptive Colormap Selection (ACS) is enabled and a new colormap has not been associated with the compressor since the last call to `xil_compress()`, the compressor uses a colormap adapted to the current frame. When ACS is disabled, the compressor will use the colormap associated with the `COMPRESSOR_COLORMAP` attribute, if that attribute has been set. If ACS is disabled and the `COMPRESSOR_COLORMAP` attribute has not been set, the compressor calls `xil_choose_colormap()` to generate an optimal

colormap for the image. When this optimal colormap is created, it is associated with the `COMPRESSOR_COLORMAP` attribute and will be used in encoding subsequent frames.

Cell Compression Ratios

The basic Cell compression technique achieves a compression ratio of 2 bits per pixel. Additional encodings further reduce the data required per pixel. First, the compressor can use code words to specify a run of constant intensity cells. This type of coding is very effective in synthetic imagery, often leading to rates of less than 1 bit per pixel. The decompressor can also optimize the decompression of these constant colored cells by using rectangle-fill hardware. A second coding technique involves the introduction of frame-to-frame coherence (interframe coding), by instructing the decoder to skip over cells that are identical to those in the previous frame. These skip codes are implemented by simply updating the current writing position to skip over the number of cells specified.

The combination of the basic Cell encoding with run codes and skip codes can lead to compression ratios of about half a bit per pixel.

Image Types

The Cell compressor is designed to work with 3-band RGB and YC_bC_r images. The width and height of these images must be divisible by four. The Cell decompressor always produces RGB images.

Creating a Cell CIS

Before you can use the Cell codec to compress images or decompress a Cell bytestream, you must create a Cell CIS. You do this by passing the compressor name `Cell` to the function `xil_cis_create()`. See the code fragment below.

```
XilCis cis;
XilSystemState state;

cis = xil_cis_create(state, "CellB");
```

14

Cell Codec Attributes

As discussed in the section "General CIS Attributes" on page 257, there is a class of attributes that can be set for any CIS. There is also a set of attributes that are valid only for CISs attached to a Cell codec. You set compressor-specific attributes using the function `xil_cis_set_attribute()`, and you read compressor-specific attributes using the function `xil_cis_get_attribute()`.

Note – Some attributes can only be read, others can only be set, and others can be both read and set. The example, or examples, that conclude the discussion of each attribute indicate how the attribute can be used.

The Cell attributes can be broadly grouped into those that affect compression and those that affect decompression. The attributes are discussed under these headings.

Compression Attributes

Setting any of the following attributes affects how the Cell compressor compresses images.

BITS_PER_SECOND

The `BITS_PER_SECOND` attribute controls the size of the bytestream the Cell compressor produces. You specify the maximum number of bits the compressor can use to encode one second's worth of video, and the compressor guarantees that it will meet this bit-rate requirement on a frame-group basis (where a frame group is a key frame and all the ensuing frames up to the next key frame). That is, if every sixth frame is a key frame and your video was captured at 30 frames per second, the compressor will encode each frame group in a maximum of `BITS_PER_SECOND` divided by 5.

Note – You use the attribute `COMPRESSOR_FRAME_RATE` to indicate the rate at which your images were captured.

Setting BITS_PER_SECOND to 0, the default value, disables bit-rate control. In addition, if you set BITS_PER_SECOND to a rate lower than the compressor can achieve, an error is generated, and bit-rate control is disabled.

The code below shows the bit-rate being set to 1152000 bits per second. This is the rate necessary to encode 30 320-by-240 frames at half a bit per pixel.

```
XilCis cis;
int bit_rate;

bit_rate = 1152000;
xil_cis_set_attribute(cis, "BITS_PER_SECOND",
    (void *)bit_rate);
```

This code reads the value of the BITS_PER_SECOND attribute.

```
XilCis cis;
int bit_rate;

xil_cis_get_attribute(cis, "BITS_PER_SECOND",
    (void **)&bit_rate);
```

COLORMAP_ADAPTION

The value of the COLORMAP_ADAPTION attribute determines whether the Cell compressor's Adaptive Colormap Selection (ACS) feature is enabled. When enabled, ACS generates a colormap for the next image to be compressed by looking at the colors in the current image. Thus, each frame in a movie may have its own colormap.

The possible values for this attribute are TRUE and FALSE, which are values of type Xil_boolean. A setting of TRUE, the default value, enables ACS, and FALSE disables it.

The code below shows the COLORMAP_ADAPTION attribute being set to FALSE.

```
XilCis cis;

xil_cis_set_attribute(cis, "COLORMAP_ADAPTION", (void *)FALSE);
```

This code reads the value of the attribute.

```
Xil_boolean acs_enabled;

xil_cis_get_attribute(cis, "COLORMAP_ADAPTION",
    (void **)&acs_enabled);
```

COMPRESSOR_COLORMAP

The COMPRESSOR_COLORMAP attribute specifies the colormap the Cell compressor should use as it encodes images. You set this attribute by passing to xil_cis_set_attribute() an XIL lookup table (a data structure of type XilLookup) that contains 8-bit indexes on the input side and 24-bit RGB values on the output side. The default value of this attribute is NULL.

The code below shows the attribute being set.

```
XilCis cis;
XilLookup colormap;

xil_cis_set_attribute(cis, "COMPRESSOR_COLORMAP",
    (void *)colormap);
```

COMPRESSOR_FRAME_RATE

Use the COMPRESSOR_FRAME_RATE attribute to let the compressor know the rate at which the images to be compressed were captured. Express this rate in microseconds per frame (a microsecond is one millionth of a second). The default value is 33333, which indicates that the frames were captured at 30 frames per second.

The frame rate you supply, or the default frame rate, is encoded in the Cell bytestream. When you play your movie back, you can read the value of the DECOMPRESSOR_FRAME_RATE attribute to determine the rate at which frames should be decompressed.

The code fragment below shows the frame rate being set to 66666 microseconds per frame (15 frames per second).

```
XilCis cis;
Xil_unsigned32 microseconds;

microseconds = 66666;
xil_cis_set_attribute(cis, "COMPRESSOR_FRAME_RATE",
    (void *)microseconds);
```

COMPRESSOR_MAX_CMAP_SIZE

The COMPRESSOR_MAX_CMAP_SIZE attribute is an integer that defines the maximum number of entries in the colormap, or colormaps, encoded in the Cell bytestream. If Adaptive Colormap Selection is enabled, this attribute limits the size of the colormaps produced by the compressor. If ACS is disabled, the attribute limits the size of the colormap you can pass to the compressor using the COMPRESSOR_COLORMAP attribute. If you pass a colormap with more than the maximum number of entries, the colormap will be truncated.

When you first create a Cell CIS, the COMPRESSOR_MAX_CMAP_SIZE attribute is set to -1, which indicates that the attribute is settable at this point. To set the attribute to a value other than 256, you must set it prior to your application's first call to xil_compress(). If you have not set the attribute by that point, it defaults to 256. Also, note that the colormap size can only be set once during the life of a CIS.

Note – When decompressing a Cell bytestream, you can determine the maximum colormap size by reading the CIS attribute DECOMPRESSOR_MAX_CMAP_SIZE. Knowing this maximum will enable you to allocate the appropriate number of X colorcells to hold the colormap.

The following code shows the COMPRESSOR_MAX_CMAP_SIZE attribute being set to 240.

```
XilCis cis;
int cmap_size = 240;

xil_cis_set_attribute(cis, "COMPRESSOR_MAX_CMAP_SIZE",
    (void *)cmap_size);
```

This code reads the value of the attribute.

```
XilCis cis;
int cmap_size;

xil_cis_get_attribute(cis, "COMPRESSOR_MAX_CMAP_SIZE",
    (void **)&cmap_size);
```

COMPRESSOR_USER_DATA

This attribute writes *user data* to a Cell bytestream. This user data can be anything you choose as long as it does not exceed 8 Kbytes. It can be an XGL rendering or an executable. The attribute's main purpose, however, is to enable you to store audio data in the bytestream.

Before setting this attribute, you must assign values to the members of a structure of type XilCellUserData. The definition of this structure is shown below.

```
typedef struct {
    Xil_unsigned8 *data;
    Xil_unsigned32 length;
} XilCellUserData;
```

The member data is a pointer to the user data, and the member length specifies the length of the data in bytes.

Once you've set up this structure, you can set the attribute using code similar to this fragment.

```
XilCis cis;
XilCellUserData user_data;

xil_cis_set_attribute(cis, "COMPRESSOR_USER_DATA",
    (void *)&user_data);
```

After you set the attribute, the next time your application calls `xil_compress()`, the Cell compressor:

1. Writes the user data to the bytestream just before the compressed image data.
2. Clears the setting of the attribute.

Thus, each time you set `COMPRESSOR_USER_DATA`, the compressor writes data to the bytestream only once.

To read user data from a Cell bytestream, you must read the attribute `DECOMPRESSOR_USER_DATA`.

ENCODING_TYPE

The `ENCODING_TYPE` attribute indicates whether a Cell compressor will use Block Truncation Coding or the Dither method. Block Truncation Coding chooses a 4-by-4 binary pixel mask and two colors (a foreground and a background color), while attempting to maintain the mean and variance of the luminance within the block. The Dither technique determines the pair of colors from the colormap that produce the least error when dithered over the 4-by-4 region. The BTC method is fastest, and usually produces good results; the Dither method is slower, but is less likely to produce noticeable contours in regions of slow color variation.

The value of the attribute can be either of the enumeration constants shown in the definition below.

```
typedef enum {
    BTC, DITHER
} XilCellEncodingType;
```

The default value is BTC.

The call below shows the ENCODING_TYPE attribute being set to DITHER.

```
XilCis cis;

xil_cis_set_attribute(cis, "ENCODING_TYPE", (void *)DITHER);
```

This code shows the attribute being read.

```
XilCis cis;
XilCellEncodingType encode_type;

xil_cis_get_attribute(cis, "ENCODING_TYPE",
    (void **)&encode_type);
```

KEYFRAME_INTERVAL

A key frame in a Cell bytestream is one that contains a bytestream information header and a colormap and uses no interframe escape codes. The KEYFRAME_INTERVAL attribute is an integer (type int) that specifies how frequently the compressor should encode key frames in the bytestream. That is, if the value of the attribute is 6, every sixth frame will be a key frame. The default value of the attribute is 6.

If you set the KEYFRAME_INTERVAL attribute to 0, no key frames are encoded in the resulting Cell bytestream. In this case, bit-rate control is disabled. (For further information about bit-rate control, see the section "BITS_PER_SECOND" on page 276.)

The code below shows KEYFRAME_INTERVAL being set to 10.

```
XilCis cis;
int key_frame = 10;

xil_cis_set_attribute(cis, "KEYFRAME_INTERVAL",
    (void *)key_frame);
```

This code reads the value of the KEYFRAME_INTERVAL attribute.

```
XilCis cis;
int key_frame;

xil_cis_get_attribute(cis, "KEYFRAME_INTERVAL",
    (void **)&key_frame);
```

TEMPORAL_FILTERING

This attribute turns on or off a form of temporal filtering. When this filtering is turned on, the compressor does not encode a new value for a pixel in frame *n* if the value of that pixel is within a certain tolerance of the same pixel in frame *n* - 1. Having the filter on generally reduces noise in an image sequence and also significantly reduces the size of the bytestream the compressor produces. The primary reason to turn the filter off would be to eliminate ghosting artifacts.

By default, the attribute is set to TRUE.

The code below shows TEMPORAL_FILTERING being set to FALSE.

```
XilCis cis;

xil_cis_set_attribute(cis, "TEMPORAL_FILTERING", (void *)FALSE);
```

This code reads the value of the TEMPORAL_FILTERING attribute.

```
XilCis cis;
Xil_boolean status;

xil_cis_get_attribute(cis, "TEMPORAL_FILTERING",
    (void **)&status);
```

Decompression Attributes

Setting any of the following attributes affects how the Cell compressor decompresses images.

DECOMPRESSOR_COLORMAP

When playing a Cell movie by decompressing frames and then using `xil_nearest_color()` to convert the resulting RGB images to 8-bit images, you must set this attribute if you want your playback code to be accelerated. The attribute specifies the XIL lookup table that `xil_nearest_color()` should use when doing its 24-bit to 8-bit conversion. Of course, your application also needs to write the values stored in this lookup table to the X colormap your application is using.

As the Cell decompressor decompresses frames, it may change the values in the lookup table. (By default, the lookup table is read-only, but you can make it writable by setting the attribute `RDWR_INDICES`, which is discussed in the section "RDWR_INDICES" on page 287.) Therefore, your application may need to call the function `xil_lookup_get_version()` to check the version number of the lookup table after each call to `xil_decompress()`. A change in version number means that the values in the lookup table have changed. If the values have changed, your application must ensure that corresponding changes are made in its X colormap before displaying the most recently decompressed frame.

Note – The section "Playing Cell Movies" on page 241 discusses an example program that sets this attribute and checks the version number of the lookup table.

You can also read the value of the `DECOMPRESSOR_COLORMAP` attribute. If it has been set, you will get back a handle to an XIL lookup table (the decompressor's colormap). If it has not been set, `xil_cis_get_attribute()` will return `NULL`.

The code fragment below shows DECOMPRESSOR_COLORMAP being set.

```
XilCis cis;
XilLookup colormap;

xil_cis_set_attribute(cis, "DECOMPRESSOR_COLORMAP",
    (void *)colormap);
```

This code reads the value of the attribute.

```
XilCis cis;
XilLookup colormap;

xil_cis_get_attribute(cis, "DECOMPRESSOR_COLORMAP",
    (void **)&colormap);
```

DECOMPRESSOR_FRAME_RATE

If the COMPRESSOR_FRAME_RATE attribute was set when the Cell compressor encoded your movie, you can use the DECOMPRESSOR_FRAME_RATE attribute to determine the rate, in microseconds per frame, at which the frames in the movie were captured. The attribute may contain different values at different points in the Cell bytestream.

If the COMPRESSOR_FRAME_RATE attribute was not set when the Cell compressor encoded your movie, the DECOMPRESSOR_FRAME_RATE attribute will contain the default value 33333 (30 frames per second).

The code fragment below shows the DECOMPRESSOR_FRAME_RATE attribute being read.

```
XilCis cis;
Xil_unsigned32 frame_rate;

xil_cis_get_attribute(cis, "DECOMPRESSOR_FRAME_RATE",
    (void **)&frame_rate);
```

DECOMPRESSOR_MAX_CMAP_SIZE

This is a read-only attribute that indicates the size of the colormap you should use when playing back your Cell movie. It's important to use the smallest colormap possible because your application must create an X colormap that is the same size as the decompressor's XIL colormap. If the X colormap is too large, colormap flashing may occur when it is installed.

Information about a movie's maximum colormap size is encoded in the Cell bytestream that represents the movie. Therefore, it an error to read this attribute of an empty CIS.

The example code below shows the `DECOMPRESSOR_MAX_CMAP_SIZE` attribute being read.

```
XilCis cis;
int cmapsize;

xil_cis_get_attribute(cis, "DECOMPRESSOR_MAX_CMAP_SIZE",
    (void **)&cmapsize);
```

DECOMPRESSOR_USER_DATA

This attribute holds a pointer to any user data that was encoded with the most recently decompressed image. To get this pointer, you might use code similar to this.

```
XilCis cis;
XilCellUserData user_data;

xil_cis_get_attribute(cis, "DECOMPRESSOR_USER_DATA",
    (void **)&user_data);
```

If user data was encoded with the last image that was decompressed, `user_data.data` will be a pointer to that data, and `user_data.length` will be the length of the data in bytes. If no user data was encoded with that image, `user_data.data` will be `NULL`, and `user_data.length` will be 0.

Any data pointer you get by using this attribute is only valid until the next call to `xil_decompress()`.

RDWR_INDICES

If your Cell movie-playback application uses the function `xil_nearest_color()` to convert 24-bit RGB images to 8-bit pseudocolor images, `xil_nearest_color()` uses the decompressor's colormap (XIL lookup table) to perform this conversion. For your application to display images using the best colors possible, the Cell decompressor must be able to alter the color values in this lookup table as the colors in the frames it is decompressing change. By default, however, the colormap is read only.

The `RDWR_INDICES` attribute enables you to make the colormap writable for a subset of its entries. Before you actually set the attribute, you must set up a structure of type `XilIndexList`. The definition of this structure is shown below.

```
typedef struct {
    Xil_unsigned32 *pixels;
    Xil_unsigned16 ncolors;
} XilIndexList;
```

You set the `ncolors` member of the structure to indicate how many of the elements in the colormap are being made writable. The array `pixels` lists the pixel values in the colormap whose RGB values the decompressor may change.

Note – If you allow the decompressor to alter the contents of its colormap, you must check the version number of the colormap after each call to `xil_decompress()`. You check this version number using the function `xil_lookup_get_version()`. If the version number has changed—that is, the contents of the colormap have changed—you must make corresponding changes in your application's X colormap before displaying the most recently decompressed frame.

Calls to `RDWR_INDICES` are not cumulative. Only the RGB values associated with the pixel values specified in your most recent setting of the attribute are writable.

The code fragment below shows the `RDWR_INDICES` attribute being set.

```
XilCis cis;
XilIndexList indexes;

xil_cis_set_attribute(cis, "RDWR_INDICES", (void *)&indexes);
```

To see how the structure `indexes` might be filled out, see the function `xilcis_color.c` in the directory `$XILHOME/examples/movie_player_example`.

Cell Molecules

The XIL library includes a series of *molecules* that accelerate the playback of Cell movies. These molecules are optimized routines that perform the jobs of two or more functions from the XIL API. You do not call such an optimized routine directly; rather, the library calls a molecule when your program calls a predefined sequence of XIL functions, sometimes with specific arguments. (These sequences of functions are discussed later in this section.)

For example, if your program calls `xil_decompress()` to decode an image stored in a Cell CIS and then calls `xil_nearest_color()` to convert the decoded image from a 24- to an 8-bit image, the library may not call these two functions. Instead, it may call a molecule that performs the decompression and the conversion in an optimized way.

Note – This replacement of two or more atomic functions by a molecule is made possible by the XIL library's deferred-execution scheme, which is discussed in Chapter 21, "Acceleration in XIL Programs." This chapter also contains a more detailed definition of molecules than the one presented here and explains how to determine whether a molecule you want to call is actually being executed.

The library's Cell-decompression molecules enable you to accelerate the playback of a Cell bytestream on

- A one-bit destination image
- A display image associated with a GX frame buffer (available only on local GX frame buffer screens)
- An 8-bit destination image other than a GX display image

The procedures you follow to call these molecules are documented later in this chapter, in the section "Calling Cell Molecules" on page 290. Before moving on to this subject, however, the chapter lists some general rules you must follow to execute any decompression molecule.

Rules for Calling Decompression Molecules

This section goes over the global rules for calling decompression molecules.

- As mentioned earlier, you must call a predefined sequence of XIL functions, sometimes with specific arguments. This sequence will be replaced by the molecule.

Note - The calls to the functions in the sequence do not necessarily have to be consecutive statements in your program. For instance, notice how the example shown in "Other 8-Bit Destination Images," which begins on page 294, calls one function in the sequence, `xil_decompress()`, and then immediately afterward calls an XIL function that is not part of the sequence and checks that function's return value. Depending on this return value, the example may then call a routine that loads an X colormap before it gets around to calling the second function in the sequence. The key here is that between calls to the functions in the sequence, the program cannot call any XIL functions whose execution the library will defer. For more information about the XIL library's deferred-execution scheme, read the section "What Is Deferred Execution?" on page 395.

- The images being decompressed and the molecule's destination image must have the same width and height, except when the molecule performs a scaling operation. In that case, the destination image must have the same dimensions as the scaled version of the source image.
- The destination image must not have a region of interest. This image will not have a region of interest unless you have explicitly set its region-of-interest attribute.
- The destination image must have an origin of 0.0, 0.0. This is the destination image's default origin.
- If a molecule uses an intermediate image, it too must have an origin of 0.0, 0.0 and no region of interest.

If any of these conditions is not met, the XIL functions in your playback code will be executed individually, and will not be replaced by a molecule. This will have a significant impact on the speed at which your movie is played.

Note – There are certain decompression molecules to which some of these global rules do not apply. These exceptions will be noted as they arise.

Calling Cell Molecules

Cell decompression molecules have been implemented for applications that play back movies using 1- and 8-bit destination images (usually display images). These molecules and the series of XIL functions you must call for these molecules to be executed are discussed below.

One-Bit Destination Image

One molecule has been defined for playing back Cell movies using a 1-bit destination image. This molecule is appropriate for playing movies on monochrome displays.

The molecule performs several tasks:

- Decompresses an image
- Converts the RGB image to a 1-band image by extracting the luminance of the RGB image
- Rescales the values in the 8-bit grayscale image so that they fall in the range 0 to 255
- *Optionally* zooms the 8-bit image by a factor of 2 in both the *x* and *y* dimensions
- Performs an ordered dither on the 8-bit image to produce a 1-bit image

For this molecule to be called, your application must include code similar to that shown below.

```
XilColorspace rgblinear, ylinear;
float scale[1], offset[1];
XilLookup colorcube;
int mult[1] = {-1};
unsigned short dims[1] = 2;
XilDitherMask dmask;

rgblinear = xil_colorspace_get_by_name(state, "rgblinear");
ylinear = xil_colorspace_get_by_name(state, "ylinear");
xil_set_colorspace(imageRGB_24, rgblinear);
xil_set_colorspace(imageY_8, ylinear);
scale[0] = 255.0 / (235.0 - 16.0);
offset[0] = -16.0 * scale[0];

/* Basically, the colorcube to be used for the ordered dither
   must be defined as shown here. The molecule will still execute
   if mult[0] is 1 instead of -1; however, this change will cause
   images to be displayed in reverse video. */
colorcube = xil_colorcube_create(state, XIL_BIT, XIL_BYTE, 1, 0,
    mult, dims);

/* The dither mask to be used in the ordered dither must be an
   8-by-8 mask. You can change the values in the mask if you
   want to. */
dmask = xil_dithermask_get_by_name(state, "dm881");

/* MOLECULE STARTS HERE */
xil_decompress(cis, imageRGB_24);
xil_color_convert(imageRGB_24, imageY_8);
xil_rescale(imageY_8, imageY_8, scale, offset);
if (ZOOM) {
    xil_scale(imageY_8, zoom_imageY_8, "nearest", 2.0, 2.0);
    xil_ordered_dither(zoom_imageY_8, zoom_image_1, colorcube,
    dmask);
}
else
    xil_ordered_dither(imageY_8, image_1, colorcube, dmask);
/* MOLECULE ENDS HERE */
```

Note – Arguments shown in boldface must be typed as shown for the molecule to execute correctly.

SPARC *Eight-Bit Display Image (GX)*

Another molecule has been defined specifically for playing back Cell movies on *local* GX frame-buffer screens. This molecule performs these tasks:

- Decompresses an image
- Converts the RGB image to an 8-bit image by finding the closest match for the RGB values in the source image in a lookup table
- *Optionally* zooms the 8-bit image by a factor of 2 in both the *x* and *y* dimensions

Note – Several of the molecule rules presented in the section "Rules for Calling Decompression Molecules" on page 289 do not apply to the version of this molecule that does not perform the zooming operation. If no zooming is requested, the destination image for the molecule does not need to be the same size as the images stored in the Cell CIS. In addition, the destination image can have a region of interest and an origin other than 0.0, 0.0.

For this molecule to be called, your application must use the code shown below.

```
XilLookup cmap;
XilIndexList indexlist;

/* You must set this attribute, or the molecule will not
   execute. cmap is the lookup table that will be passed
   to xil_nearest_color(). */
xil_cis_set_attribute(cis, "DECOMPRESSOR_COLORMAP",
    (void *)cmap);

/* To get the best results, you should set this attribute
   so that the decompressor can change any of the entries in
   cmap. */
xil_cis_set_attribute(cis, "RDWR_INDICES", (void *)&indexlist);

/* MOLECULE STARTS HERE */
xil_decompress(cis, imageRGB_24);

/* This checking of the lookup table's version number is
   necessary, but is not part of the molecule */
if (lu_version != xil_lookup_get_version(cmap)) {
    /* Include code to write the appropriate values from cmap to
       the application's X colormap */
    ...
    lu_version = xil_lookup_get_version(cmap);
}

if (ZOOM) {
    xil_nearest_color(imageRGB_24, image_8, cmap);
    xil_scale(image_8, zoom_image_GX, "nearest", 2.0, 2.0);
}
else
    xil_nearest_color(imageRGB_24, image_GX, cmap);
/* MOLECULE ENDS HERE */
```

Other 8-Bit Destination Images

If you want to play back your Cell movie using an 8-bit destination image other than a local GX display image, you can use one of three molecules. The first two molecules use a nearest-color strategy to convert images from 24 bits to 8 bits, and the third molecule uses an ordered dither. All three molecules require that you set the CIS attribute `DECOMPRESSOR_COLORMAP` before you execute them.

Nearest-Color Approach

Functionally the first molecule in this class is identical to the basic molecule (no zoom) designed for playing movies on a local GX frame buffer. (See the section "Eight-Bit Display Image (GX)" on page 292.) However, it is not as fast as the GX version, so displaying a movie on an indexed-color frame buffer other than the GX will not be quite as fast as on a GX.

For this molecule to execute, your application must use the code shown in the section "Eight-Bit Display Image (GX)." In addition:

- The destination image must be stored on a word boundary. It will be stored on such a boundary unless it is a child image created with an *x* offset from its parent that is not a multiple of four.
- The pixel stride in the destination image must be one. This will always be true unless the destination image is a single-band child of a multiband parent.

The second molecule in this class rapidly decompresses Cell images and displays them in a small window. It performs the following tasks:

- Decompresses an image
- Converts the RGB image to an 8-bit image by finding the closest match for the RGB values in the source image in a lookup table
- Scales the image down using *x* and *y* scale factors of 1 / (4 * *n*), where *n* is a positive integer

The destination image must be at least 4-by-4 pixels in size.

For this molecule to be called, your application must use code similar to that shown below.

```
/* MOLECULE BEGINS HERE */
xil_decompress(cis, imageRGB_24);
if (lu_version != xil_lookup_get_version(cmap)) {
    /* Include code to write the appropriate values from cmap to
       the application's X colormap */
    ...
    lu_version = xil_lookup_get_version(cmap);
}
xil_nearest_color(imageRGB_24, image_8, cmap);
xil_scale(image_8, small_image_8, "nearest", .25, .25);
/* MOLECULE ENDS HERE */
```

Ordered-Dither Approach

This molecule performs the following tasks:

- Decompresses an image
- *Optionally* zooms the RGB image by a factor of 2 in the *x* and *y* dimensions
- Dithers the 3-band image to a 1-band image using a colorcube

For the molecule to execute, the destination image must be word aligned and must have a pixel stride of one.

To call this molecule, your application must use the code similar to that shown below.

```
/* Create a colorcube and dither mask. All legal XIL
   colorcubes are supported, including those that have
   decreasing ramps in one or more bands. The dither mask
   must be a 4-by-4 mask, but there are no restrictions
   regarding its contents. */
...

/* MOLECULE STARTS HERE */
xil_decompress(cis, imageRGB_24);
if (ZOOM) {
    xil_scale(imageRGB_24, zoom_imageRGB_24, "nearest", 2.0,
        2.0);
   xil_ordered_dither(zoom_imageRGB_24, zoom_image_8, colorcube,
        dithermask);
}
else
    xil_ordered_dither(imageRGB_24, image_8, colorcube,
        dithermask);
/* MOLECULE ENDS HERE */
```