MPEG-1 Codec 19

This chapter discusses the MPEG-1 video codec specified by the Moving Picture Experts Group, an ISO working group. This group has produced a standard that is similar to the H.261 standard developed by the CCITT (see Chapter 18, "H.261 Codec"), but places less emphasis on low bit rates. By accepting a higher bit rate—for example, 1.5 Mbits per second—an MPEG-1 codec is able to recreate very high-quality pictures and to produce a bitstream that is easily editable.

The rate of 1.5 Mbits/s makes the MPEG-1 codec especially viable in applications that read compressed data from CD-ROMs because even older CD-ROM readers can read data at this speed. Thus, putting an MPEG-1 bitstream on a CD-ROM is an effective way to distribute movies, business presentations, and training videos.

The remainder of this chapter is divided into four sections. The first section provides an overview of how an MPEG-1 codec works. The second explains briefly how to create an MPEG-1 CIS. The third discusses CIS attributes that apply specifically to a CIS associated with an XIL MPEG-1 compressor or decompressor (as opposed to the general CIS attributes covered in the section "General CIS Attributes" on page 257). And the fourth section introduces the subject of accelerating the playback of MPEG-1 bitstreams. For further information on this subject, see Chapter 21, "Acceleration in XIL Programs."

19

Note – This chapter discusses both MPEG-1 compression and decompression. However, the current release of the XIL library includes only an MPEG-1 decompressor. The compressor interface is defined for third parties who want to implement XIL MPEG-1 compressors.

How an MPEG-1 Codec Works

Since the work done by the Motion Picture Experts Group grew out of the work done by the CCITT in developing the H.261 video codec, this section explains how the MPEG-1 codec works by comparing and contrasting it with the H.261 codec. (If you're unfamiliar with how an H.261 codec works, see the section "How an H.261 Codec Works" on page 338.) The present section also includes a subsection that describes the organizational structure that MPEG-1 imposes on a video sequence.

Similarities Between MPEG-1 and H.261

The key similarities between MPEG-1 and H.261 are listed below:

- Both compressors work with YC_bC_r pictures in which the information in the chroma channels has been subsampled so that there is one C_b and one C_r value for each 2-by-2 block of luma values.
- Like the H.261 compressor, the MPEG-1 compressor can compress a macroblock (a 16-by-16 block of pixels in a picture) by encoding the actual Y, C_b, and C_r values in the macroblock (intraframe encoding) or by encoding the differences between values in the current block and values in the corresponding macroblock in the previous picture (forward prediction). In addition, when using the forward-prediction encoding method, the MPEG-1 compressor, like the H.261 compressor, can employ motion compensation.
- Both compressors encode 8-by-8 blocks of pixel values or difference values using the same method. The compressors first perform a Discrete Cosine Transform (DCT) on the 8-by-8 block of values. This operation transforms the values in the block from the spatial to the transform domain. Second, the compressors quantize the coefficients produced by the DCT. Finally, the compressors use entropy coding to encode the quantized coefficients.

Differences Between MPEG-1 and H.261

Although the Moving Picture Experts Group drew heavily on the work of the CCITT, there are also very significant differences between the MPEG-1 and H.261 compressors. Most of these differences provide for random access to the MPEG-1 bitstream and make the bitstream easily editable.

I Pictures and P Pictures

In H.261, the ideas of intraframe encoding and predictive encoding are applied for the most part at the macroblock level. MPEG-1, on the other hand, includes the notion of intraframe-encoded pictures and predicted pictures.

In an intraframe-encoded picture, or I picture, all macroblocks are intraframe encoded. This, of course, means that the decoder needs no information from a preceding picture to decode an I picture. For this reason, seeks to I frames can be performed very quickly.

In a predicted picture, or P picture, each macroblock can be intraframe encoded or encoded using the forward-prediction method. This type of picture is very similar to an H.261 picture.

Typically, an MPEG-1 bitstream will contain more P pictures than I pictures because the P picture can be encoded using fewer bits. Encoding difference values generally requires fewer bits than encoding pixel values. Also, the encoder does not have to encode macroblocks that are very similar to their counterparts in the preceding I or P picture.

B Pictures

From what we've said so far, you might picture an MPEG-1 bitstream as containing periodic I pictures followed by a number of P pictures. See Figure 19-1.

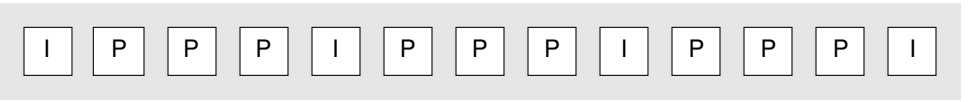

Figure 19-1 An MPEG-1 Bitstream Containing I and P Pictures

This does constitute a legal MPEG-1 bitstream. However, the bitstream may also contain one or more bidirectionally predicted pictures, or B pictures, between any pair of I or P pictures. In a B picture, each macroblock may be

- Intraframe encoded
- Forward predicted from the nearest preceding I or P picture
- Backward predicted from the nearest succeeding I or P picture
- Bidirectionally predicted from the nearest preceding I or P picture and the nearest succeeding I or P picture

We've already discussed intraframe encoding and forward prediction.

Backward prediction is strictly analogous to forward prediction. A backward-predicted macroblock is encoded with respect to the values in the corresponding macroblock in a picture that follows its own picture in the video sequence. This option may seem unintuitive at first thought because it results in a situation where pictures are not transmitted in the order in which they will be displayed. See Figure 19-2.

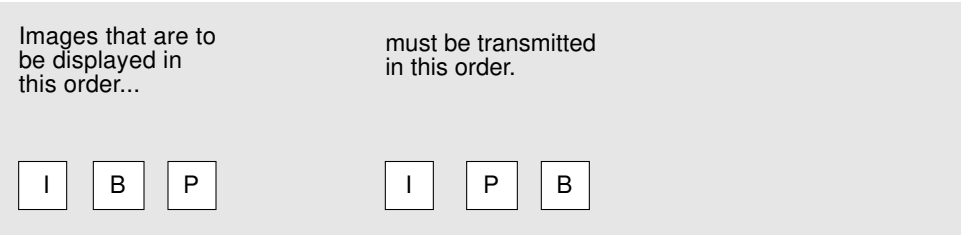

Figure 19-2 MPEG-1 Display Order Versus Decoding Order

However, backward prediction is an option because it can lead to a more compact bitstream in instances where the macroblocks in a B picture match their counterparts in the closest succeeding I or P picture more closely than they match their counterparts in the closest preceding I or P picture.

A bidirectionally predicted macroblock is coded with respect to the corresponding macroblock in both the closest preceding and succeeding I/P pictures. See Figure 19-3.

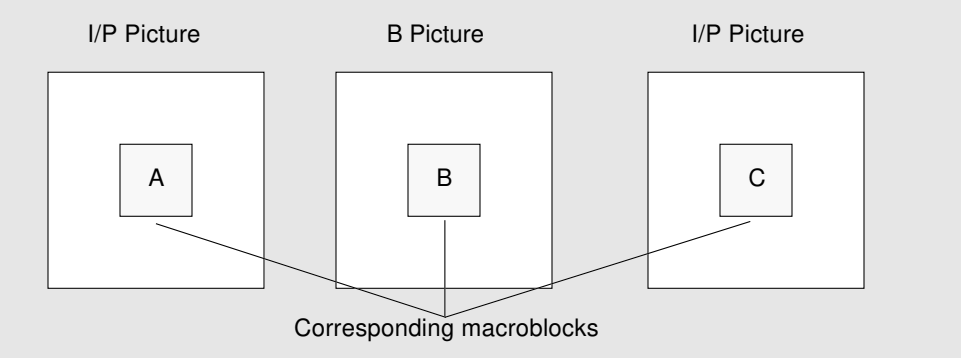

Figure 19-3 Bidirectional Prediction in MPEG-1

In this case, the encoder averages the values in macroblocks A and C and then encodes the difference between values in macroblock B and the averaged macroblock. Bidirectional prediction is effective because the difference mentioned above is often so small that it does not need to be encoded.

For MPEG-1, the behavior of the `xil_cis_get_bits_ptr()` function differs from its usual behavior. For a bitstream with out-of-order frames (that is, a bitstream with B frames), the actual number of frames in the data returned by `xil_cis_get_bits_ptr()` may not equal the value of its `nframes` parameter. The value of `nframes` is the number of frames the CIS read frame has been advanced by the `xil_cis_get_bits_ptr()` call. A seek back by `nframes` by calling `xil_cis_seek()` will restore the read position to the original read frame (before the `xil_cis_get_bits_ptr()` call). This is useful for an application that uses a preview mode, where the compress and write-to-file is followed by a decompress.

Since the number of frames reported may not represent the actual number of frames, if the chunk is subsequently used for an `xil_cis_put_bits()` or `xil_cis_put_bits_ptr()` call, the `nframes` parameter must be set to -1, which indicates an unknown number of frames.

Groups of Pictures

Another concept used in MPEG-1 that does not apply in H.261 is that of a group of pictures. This is a series of consecutive pictures from a video sequence. Generally, the group of pictures provides a unit of the bitstream that can be removed without destroying the integrity of the bitstream and a unit

that can be decoded independently of the rest of the bitstream. As you'll see in a moment, however, the group of pictures does not always have these characteristics.

By definition, a group of pictures (considered in display order) must begin with an I picture or with one or more B pictures followed by an I picture. It must end with an I or a P picture. A closed group of pictures can be decoded without any information from the preceding group of pictures. Thus, a closed group is one that begins with an I picture or one that begins with one or more B pictures whose macroblocks have been encoded using only intraframe encoding and backward prediction. An open group of pictures begins with one or more B pictures, at least one of which contains macroblocks encoded using forward or bidirectional prediction. This type of group can be decoded only if the preceding group of pictures is available. If that group is not available—for example, if an MPEG-1 bitstream editing program has removed the group—a broken-link bit must be set in the header for the open group of pictures.

How MPEG-1 Organizes a Video Sequence

We've already mentioned some of the organizational units used in MPEG-1, such as the group of pictures and the macroblock. This section provides a complete overview of these units. This information should be useful to you as you read about MPEG-1 CIS attributes later in this chapter because many of these attributes are associated with a particular unit.

The largest unit MPEG-1 defines is the video sequence. You might think of this unit as an entire movie or presentation. Each MPEG-1 bitstream includes a sequence header, which defines several attributes for the entire sequence. These include the pixel aspect ratio for the sequence, the picture rate for the sequence in pictures per second, and the bit rate of the data channel over which the compressed sequence will be moved. An additional sequence header is written to the bitstream each time one of the encoder's quantization tables is changed.

Each sequence is divided into a series of groups of pictures. See the section "Groups of Pictures" on page 363 for a definition of this unit. If you're compressing video using an XIL-compliant MPEG-1 encoder, you can control the makeup of each group of pictures using the attribute `COMPRESSOR_PATTERN`.

Each group of pictures consists of pictures, which are individual frames of video. The major attributes associated with each picture are a picture type and a temporal reference. The major picture types are intraframe-encoded pictures, forward-predicted pictures, and bidirectionally predicted pictures. The temporal reference is an integer identifying a picture's place within a group of pictures.

The largest subdivision of a picture is called a slice, which consists of a series of consecutive macroblocks. Slices within a picture may vary in size, but each macroblock in a picture must be part of a slice. This unit is designed primarily to help a decoder recover from a bitstream error. If a decoder detects an error, one way to recover is to skip to the next slice header.

Slices, as mentioned in the preceding paragraph, are built from macroblocks. These are 16-by-16 block of pixels. The macroblock is the level at which motion compensation is performed. Also, the type of encoding used—intraframe, forward prediction, backward prediction, or bidirectional prediction—can change from macroblock to macroblock.

Finally, each macroblock is divided into six 8-by-8 blocks. Four of these blocks contain luma values, one contains C_b values, and one contains C_r values. This is the level at which the DCT is performed.

Creating an MPEG-1 CIS

Before you can use the MPEG-1 decompressor to decompress an MPEG-1 bitstream, you must create an MPEG-1 CIS (and write an MPEG-1 bitstream to the CIS). You create this CIS by passing the decompressor name `Mpeg1` to the function `xil_cis_create()`. See the code fragment below.

```
XilCis cis;
XilSystemState state;

cis = xil_cis_create(state, "Mpeg1");
```

MPEG-1 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 an MPEG-1 compressor or

decompressor. You set codec-specific attributes using the function `xil_cis_set_attribute()`, and you read codec-specific attributes using the function `xil_cis_get_attribute()`.

The MPEG-1 attributes can be broadly grouped into those that affect compression and those that affect decompression. The attributes are discussed under these headings below.

Compression Attributes

Setting any of the following attributes affects how an MPEG-1 compressor compresses images. It is also possible to read the value of each of these attributes.

COMPRESSOR_BITS_PER_SECOND

You use this attribute to tell the encoder how many bits it can use to encode one second's worth of pictures. This value is written to the sequence header for a picture sequence and is assumed to remain constant for the duration of the sequence. You cannot change the value of this attribute after the encoder compresses the first picture in the sequence.

The value you assign to the attribute must fall in the range 1 to 104,856,800. Before inserting this value in the sequence header, the encoder rounds it up to the nearest multiple of 400. The default value of the attribute is 1,152,000.

The code below shows how you might set the value of `COMPRESSOR_BITS_PER_SECOND` to 1,856,000.

```
XilCis cis;
int bits_per_second = 1856000;

xil_cis_set_attribute(cis, "COMPRESSOR_BITS_PER_SECOND",
    (void *)bits_per_second);
```

To read the value of the attribute, you might use code similar to this.

```
XilCis cis;
int bits_per_second;

xil_cis_get_attribute(cis, "COMPRESSOR_BITS_PER_SECOND",
    (void **)&bits_per_second);
```

COMPRESSOR_INSERT_VIDEO_SEQUENCE_END

Normally, the value of this attribute is `FALSE`. If you set it to `TRUE`, any time a subsequent call is made to `xil_cis_flush()`, the encoder writes a video-sequence-end (*eos*) code to the last frame in the CIS. This action is added to the normal actions taken by the flush routine. When set to `FALSE`, this attribute doesn't affect the normal actions of the flush routine.

The code below shows the attribute being set to `TRUE`.

```
XilCis cis;

xil_cis_set_attribute(cis,
    "COMPRESSOR_INSERT_VIDEO_SEQUENCE_END", (void *)TRUE);
```

Note – The library prevents multiple *eos* codes from being written to the same frame. Thus, when the write-frame's number doesn't change and `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END` is `TRUE`, multiple calls to `xil_cis_flush()` result in only one *eos* code in the frame; this code is written after the first call to `xil_cis_flush()`.

You can read the value of this attribute using code similar to the following:

```
XilCis cis;
Xil_boolean sequence_end_status;

xil_cis_get_attribute(cis,
    "COMPRESSOR_INSERT_VIDEO_SEQUENCE_END",
    (void **)&sequence_end_status);
```

Table 19-1 shows two command sequences that result in the same CIS pattern; these sequences assume the call to `xil_cis_set_attribute()` sets the `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END` attribute to `TRUE`.

Note – On this page and the following pages, the term *CIS pattern* refers to the order that frames appear in the bitstream. Don't confuse this with the `COMPRESSOR_PATTERN` attribute (see page 376), which lets you specify the makeup of a group of pictures. For example, the `COMPRESSOR_PATTERN` IBBPBBP will result in the bitstream CIS pattern IPBBPBB.

Table 19-1 Two Command Sequences: CIS Pattern = IPBB

Command Sequence 1	**Command Sequence 2**
xil_compress	xil_cis_set_attribute
xil_compress	xil_compress
xil_compress	xil_compress
xil_compress	xil_compress
xil_cis_flush /* normal */	xil_compress
xil_cis_set_attribute	xil_cis_flush /* normal + eos */
xil_cis_flush /* normal + eos */	

For the two sequences shown in Table 19-1, the resulting CIS sequence is:

```
vsh.I0.P3.B1.B2.eos
```

where *vsh* represents the video sequence header information, and *eos* represents the end-of-sequence code. Between the *vsh* and the *eos* code are the frame types and their display id's; among the frame types, I = interframe encoding, P = predicted pictures, and B = bidirectionally predicted pictures.

From the XIL point of view, the *vsh* is bundled with the first frame, *I0*, and the *eos* is bundled with the last frame, *B2*.

There may be multiple *vsh* components associated with one *eos*, since the *vsh* changes as certain CIS attributes change (within the XIL limitations that there are no width/height changes).

In addition, there may be multiple sequences within a bitstream; thus,

```
vsh-----eos.vsh----eos.vsh----eos
```

constitutes three sequences.

If frames are compressed into the CIS after the call to `xil_cis_flush()`, it's the compressor's responsibility to provide the *vsh* per sequence. Before the application changes an attribute that would result in a new sequence header, it must first output an *eos* for the current sequence by calling `xil_cis_flush()` with `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END = TRUE`. For example, before you can change the attribute `COMPRESSOR_BITS_PER_SECOND` (see page 366), you must first set `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END` to `TRUE` and call `xil_cis_flush()` to write the *eos* code to the bitstream.

An MPEG-1 sequence isn't valid without the *eos* code; therefore, the last frame in the sequence must be followed by the *eos* code. Since it cannot be predicted when an application will end a sequence, the MPEG-1 codec *reserves* the last frame or subgroup of frames in the CIS so that you can write an *eos* to that frame or subgroup. The reserved frame or subgroup must be released before it can be retrieved with `xil_cis_get_bits_ptr()` or `xil_decompress()`.

Note – The reserved frame or subgroup affects the logic you need for calling `xil_cis_get_bits_ptr()` and `xil_decompress()`. For example, typically you call `xil_decompress()` from within a loop that executes only when `xil_cis_has_frame()` returns `TRUE`. When decompressing an MPEG-1 CIS, you must first release the reserved frames before you can expect a loop control condition that depends on the return value of `xil_cis_has_frame()` to evaluate to `TRUE`.

The following paragraphs tell you how to release the reserved frames. Beginning on page 371, the discussion moves to some example function-call sequences and shows how these sequences would affect both the return value of `xil_cis_has_frame()` and `xil_cis_get_bits_ptr()`.

A frame or subgroup is released when:

- It is followed by an *eos*, thus providing a valid sequence. This is done by setting the `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END` attribute to `TRUE` and then calling `xil_cis_flush()`, as described above.
- It is followed by another frame or subgroup. For an all I bitstream or a mixed I-P bitstream, this just means that another frame is added to the CIS. Since there are no out-of-order frames, a subgroup for this type of CIS is just a frame.

The following sequences release a frame:

```
I0.I1          I1 releases I0
I0.P1          P1 releases I0
```

For any bitstream with B frames, releasing a frame or subgroup is more complicated. B frames have a future-predictive frame that appears in the bitstream *previous* to the B frame, and multiple B frames may share the same predictive frame. So you can get the following (in these examples, any of the I's can be replaced by a P):

```
I2.B1
I3.B1.B2
I4.B1.B2.B3
I6.B1.B2.B3.B4.B5
I9.B1.B2.B3.B4.B5.B6.B7.B8
```

A sequence is considered a complete *subgroup* when its predictive frame has the display id N, and the predictive frame is followed by one or more B frames, the last of whose display id is N-1. Thus, all of the above bitstream fragments are complete subgroups.

The following are examples of bitstreams with incomplete subgroups:

```
I3.B1          missing B2
I6.B1.B2.B3    missing B4,B5
```

When there are incomplete subgroups, the compressor must buffer the compressed frames until the buffer contains a complete subgroup, at which point the compressor adds the completed subgroup to the bitstream. In this manner, the compressor can handle interruptions to the compression-frame sequence as necessary. For example, if, halfway through the IPBBB group, the video player's Stop/Output button is pressed, the compressor must be able to take the resulting I0.P4.B1 bitstream and rework it into a legal output form.

Once the subgroup is complete, it cannot be released until it is followed by an *eos*, or it is followed by another valid subgroup. The following sequences show a subgroup being released:

```
I0.I2.B1.eos              I2.B1 released by eos
I0.I2.B1.I4.B3            I2.B1 released by I4.B3
I0.I3.B1.B2.P5.B4         I3.B1.B2 released by P5.B4
I0.I3.B1.B2.P5.B4.eos     P5.B4 released by eos
```

The tables below show how subgroups are affected by a particular sequence of function calls. In each of these tables:

- Column heading **WF** is an abbreviation for Write Frame
- Column heading **has_frame** is an abbreviation for the `xil_cis_has_frame()` function and the column text shows what the return value for that function would be
- Column heading **get_bits_ptr** is an abbreviation for the `xil_cis_get_bits_ptr()` function and the column text shows the frames that would be returned by a call to that function
- Column heading **RF** is an abbreviation for Read Frame and the column text shows what the read frame would be after the call to `xil_cis_get_bits_ptr()`
- A call to `xil_cis_set_attribute()` in the sequence sets the `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END` attribute to `TRUE`

In Table 19-2, the CIS pattern is all I frames and the result is that subgroup I2 is released by an appended *eos*. Notice that the first `xil_cis_flush()` call has no affect on the subgroup since the default setting for the `COMPRESSOR_INSERT_VIDEO_SEQUENCE_END` attribute is `FALSE`.

Table 19-2 Releasing a Frame: CIS Pattern = All I Frames

Function	WF	has_frame	get_bits_ptr	RF	CIS
xil_compress	1	FALSE	NULL	0	I0
xil_compress	2	TRUE	I0	1	I0.I1
xil_compress	3	TRUE	I1	2	I0.I1.I2
xil_cis_flush	3	FALSE	NULL	2	I0.I1.I2
xil_cis_set_attribute					
xil_cis_flush	3	TRUE	I2.eos	3	I0.I1.I2.eos

In Table 19-3, the CIS pattern is IPB and the result is that subgroup P2.B1 is released by the appended *eos*.

Table 19-3 Releasing a Frame: CIS Pattern = IPB

Function	WF	has_frame	get_bits_ptr	RF	CIS
xil_compress	1	FALSE	NULL	0	I0
xil_compress	2	FALSE	NULL	0	I0
xil_compress	3	TRUE	I0	1	I0.P2.B1
xil_cis_set_attribute					
xil_cis_flush	3	TRUE	P2.B1.eos	3	I0.P2.B1.eos

Finally, in Table 19-4, the CIS pattern is IPBBPBB and the result is that subgroup P3.B1.B2 is released by the next subgroup, P6.B4.B5.

Table 19-4 Releasing a Frame: CIS Pattern = IPBBPBB

Function	WF	has_frame	get_bits_ptr	RF	CIS
xil_compress	1	FALSE	NULL	0	I0
xil_compress	2	FALSE	NULL	0	I0
xil_compress	3	FALSE	NULL	0	I0
xil_compress	4	TRUE	I0	1	I0.P3.B1.B2
xil_compress	5	FALSE	NULL	1	I0.P3.B1.B2
xil_compress	6	FALSE	NULL	1	I0.P3.B1.B2
xil_compress	7	TRUE	P3.B1.B2	4	I0.P3.B1.B2.P6.B4.B5

COMPRESSOR_INTRA_QUANTIZATION_TABLE

This attribute enables you to specify an 8-by-8 matrix of values to be used in quantizing the DCT coefficients for blocks in intraframe-encoded macroblocks. If you set this attribute, the table will be written to your bitstream's sequence header. If you don't set it, the encoder will use a default implementation-specific table.

If you supply your own table, it must meet these requirements:

- The first value in the table must be an 8, the fixed quantization level of the DC coefficient for the block.
- The remaining values must fall in the range 1 to 255.
- The values must be listed in the zigzag order shown in Figure 19-4.

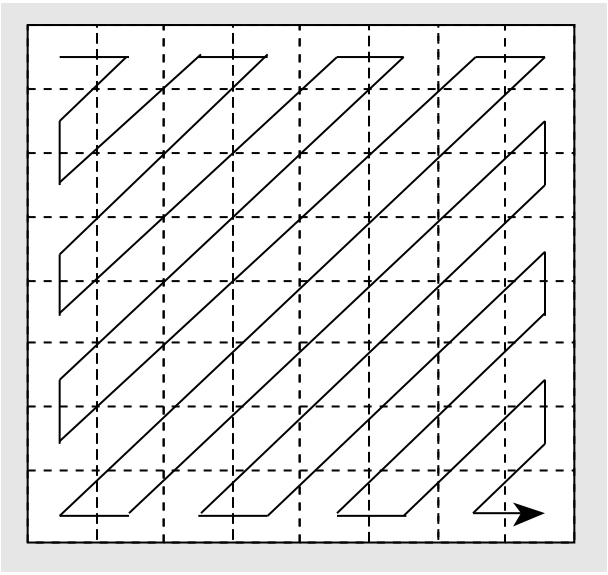

Figure 19-4 Zigzag Ordering of Quantization Table Values

The code below shows how to specify a table. This particular table is the one that is used as an example in the MPEG-1 standard and may well be the encoder's default table.

```
XilCis cis;
Xil_unsigned8 intra_quant_table[64] = {
    8, 16, 19, 22, 26, 27, 29, 34,
    16, 16, 22, 24, 27, 29, 34, 37,
    19, 22, 26, 27, 29, 34, 34, 38,
    22, 22, 26, 27, 29, 34, 37, 40,
    22, 26, 27, 29, 32, 35, 40, 48,
    26, 27, 29, 32, 35, 40, 48, 58,
    26, 27, 29, 34, 38, 46, 56, 69,
    27, 29, 35, 38, 46, 56, 69, 83 };

xil_cis_set_attribute(cis,
    "COMPRESSOR_INTRA_QUANTIZATION_TABLE",
    (void *)intra_quant_table);
```

You can read the value of the attribute using code similar to this.

```
XilCis cis;
Xil_unsigned8 *intra_quant_table;

xil_cis_get_attribute(cis,
    "COMPRESSOR_INTRA_QUANTIZATION_TABLE",
    (void **)&intra_quant_table);
```

If, after this call, `intra_quant_table` is a null pointer, the default table is in use; otherwise, `intra_quant_table` is a pointer to a user-supplied quantization matrix. In the latter case, your application must free the matrix pointed to by `intra_quant_table` when it is no longer needed.

COMPRESSOR_NON_INTRA_QUANTIZATION_TABLE

This attribute enables you to specify an 8-by-8 matrix of values to be used in quantizing the DCT coefficients for blocks in nonintraframe-encoded macroblocks. If you set this attribute, the table will be written to your bitstream's sequence header. If you don't set it, the encoder will use an implementation-specific default table.

If you supply an alternate table, it must meet these requirements:

- The values in the table must fall in the range 1 to 255.
- The values must be listed in the zigzag order shown in Figure 19-4 on page 373.

The code below is an example of how you might supply a custom table. This table, which contains all 16's, is used as an example in the MPEG-1 specification.

```
XilCis cis;
Xil_unsigned8 non_intra_quant_table[64];
int i;


for (i = 0; i < 64; i++)
    non_intra_quant_table[i] = 16;

xil_cis_set_attribute(cis,
    "COMPRESSOR_NON_INTRA_QUANTIZATION_TABLE",
    (void *)non_intra_quant_table);
```

You can read the value of the attribute using code similar to this.

```
XilCis cis;
Xil_unsigned8 *non_intra_quant_table;

xil_cis_get_attribute(cis,
    "COMPRESSOR_NON_INTRA_QUANTIZATION_TABLE",
    (void **)&non_intra_quant_table);
```

COMPRESSOR_PATTERN

This attribute enables you to specify the makeup of each group of pictures. You dictate the number of pictures in each group, the types of the pictures in the group, and the display order of those pictures. Your specification must conform to the following rules:

- A group of pictures may begin with one or more B pictures or an I picture.
- If the group starts with one or more B pictures, those pictures must be followed by an I picture.
- The first I picture in the group may be followed by any number of I and P pictures, and any combination of an I and a P picture may have intervening B pictures.
- The group of pictures must end with an I or a P picture.

Note – A group of pictures can also contain all D pictures. (A D picture contains only low-frequency information and is meant to be used for fast visible searches.) However, the XIL library's MPEG-1 decompressor cannot decode bitstreams containing D pictures.

You specify the makeup of the group by assigning values to a structure of type `XilMpeg1Pattern`:

```
typedef struct {
    char *pattern;
    Xil_unsigned32 repeat_count;
} XilMpeg1Pattern;
```

The string `pattern` determines the types of the pictures (in display order) that will appear at the beginning of the group of pictures. This string must contain a combination of the characters shown in Table 19-5.

Table 19-5 Characters Representing Picture Types

Character	Picture Type
I	Intraframe picture
P	Forward-predicted picture
B	Bidirectionally predicted picture
D	DC-coefficient picture

For example, suppose you want each group of pictures to begin with the thirteen pictures shown in Figure 19-5.

Figure 19-5 Sample Group of Pictures

You would set `pattern` to `"IBBPBBPBBPBBP"`.

The parameter `repeat_count` is a number greater than 0 that determines the total number of pictures in the group of pictures. For example, suppose that you set `repeat_count` to 3 as shown in the following example.

```
XilCis cis;
XilMpeg1Pattern picture_pattern = {"IBBPBBPBBPBBP", 3};

xil_cis_set_attribute(cis, "COMPRESSOR_PATTERN",
    (void *)&picture_pattern);
```

This call requests that the pattern shown in Figure 19-5 be repeated 3 times, so that there will be a total of 39 pictures in the group.

When you set this attribute, the encoder terminates the group of pictures it was working on immediately. Thus, the next call to `xil_compress()` causes the encoder to compress the first picture in a new group of the type you just specified.

If you do not set this attribute, a default implementation-specific pattern will be used. To select the default pattern after having selected a custom pattern, you pass a null pointer to `xil_cis_set_attribute()`.

You can also read the value of `COMPRESSOR_PATTERN`. To do this, you might use code similar to the following.

```
XilCis cis;
XilMpeg1Pattern *picture_pattern;

xil_cis_get_attribute(cis, "COMPRESSOR_PATTERN",
    (void **)&picture_pattern);
```

If, after this call, `picture_pattern` is a null pointer, the encoder is using the default picture pattern. Otherwise, `picture_pattern` points to a structure containing a pattern string and a repeat count. In the latter case, when you have finished using the structure, your application should free the storage that the XIL library allocated to hold the pattern string and then free the storage the library allocated to hold the structure.

COMPRESSOR_PEL_ASPECT_RATIO

The value of this attribute, along with the width and height of a picture in pixels, indicates the shape a picture is meant to have when it is displayed. Basically, the ratio of the width to the height of the displayed picture should be:

width in pixels / (height in pixels * aspect ratio)

If you supply an aspect ratio, this value is written to the sequence header for your bitstream and can be read later by an XIL-compliant MPEG-1 decompressor. The default value is implementation specific.

The valid values for COMPRESSOR_PEL_ASPECT_RATIO are contained in the enumeration shown below.

```
typedef enum {
    NullDefault,
    Ratio_1_0,      /* 1.0    */
    Ratio_0_6735,   /* 0.6735 */
    Ratio_0_7031,   /* 0.7031 */
    Ratio_0_7615,   /* 0.7615 */
    Ratio_0_8055,   /* 0.8055 */
    Ratio_0_8437,   /* 0.8437 */
    Ratio_0_8935,   /* 0.8935 */
    Ratio_0_9157,   /* 0.9157 */
    Ratio_0_9815,   /* 0.9815 */
    Ratio_1_0255,   /* 1.0255 */
    Ratio_1_0695,   /* 1.0695 */
    Ratio_1_0950,   /* 1.0950 */
    Ratio_1_1575,   /* 1.1575 */
    Ratio_1_2015    /* 1.2015 */
} XilMpeg1PelAspectRatio;
```

The most important values in the enumeration are Ratio_1_0, Ratio_0_9157, and Ratio_1_0950. The value Ratio_1_0 indicates that the pictures in the sequence can be shown without distortion on a computer monitor, which has square pixels. Ratio_0_9157 indicates that pictures can be displayed without distortion on CCIR Rec. 601 625-line televisions. And Ratio_1_0950 indicates that pictures can be displayed without distortion on CCIR Rec. 601 525-line televisions.

You can use the value NullDefault to reset the aspect ratio to its default value.

The code below shows COMPRESSOR_PEL_ASPECT_RATIO being set to Ratio_1_0.

```
XilCis cis;
XilMpeg1PelAspectRatio aspect_ratio = Ratio_1_0;

xil_cis_set_attribute(cis, "COMPRESSOR_PEL_ASPECT_RATIO",
    (void *)aspect_ratio);
```

You can read the current value of this attribute using code similar to that shown below.

```
XilCis cis;
XilMpeg1PelAspectRatio aspect_ratio;

xil_cis_get_attribute(cis, "COMPRESSOR_PEL_ASPECT_RATIO",
    (void **)&aspect_ratio);
```

If `aspect_ratio` equals `NullDefault` after this call, the aspect ratio for the sequence is the default aspect ratio defined by the implementors of the encoder.

COMPRESSOR_PICTURE_RATE

You use this attribute to specify the picture rate of the sequence being encoded in pictures per second. This value and the value of `COMPRESSOR_BITS_PER_SECOND` enable the encoder to determine how many bits it can use to encode each picture.

The valid values for `COMPRESSOR_PICTURE_RATE` are the enumeration constants shown below. These constants correspond to the rates of commonly available sources of analog and digital video.

```
typedef enum {
    NullDefault,
    Rate_23_976,   /* 23.976 */
    Rate_24,       /* 24.0   */
    Rate_25,       /* 25.0   */
    Rate_29_97,    /* 29.97  */
    Rate_30,       /* 30.0   */
    Rate_50,       /* 50.0   */
    Rate_59_94,    /* 59.94  */
    Rate_60        /* 60.0   */
} XilMpeg1PictureRate;
```

The default value of this attribute is implementation specific. The code below shows the value being set to `Rate_29_97`.

```
XilCis cis;
XilMpeg1PictureRate picture_rate = Rate_29_97;

xil_cis_set_attribute(cis, "COMPRESSOR_PICTURE_RATE",
    (void *)picture rate);
```

You can read the current value of this attribute using code similar to that shown below.

```
XilCis cis;
XilMpeg1PictureRate picture_rate;

xil_cis_get_attribute(cis, "COMPRESSOR_PICTURE_RATE",
    (void **)&picture_rate);
```

If `picture_rate` equals `NullDefault` after this call, the rate for the sequence is the default rate defined by the implementors of the encoder.

COMPRESSOR_SLICES_PER_PICTURE

The value of this attribute provides a hint to the encoder concerning the number of slices into which it should divide the next picture it compresses; however, an XIL-compliant encoder may ignore this suggested value.

The valid values for this attribute range from 1 to the number of macroblocks in the picture. (The default value of the attribute is implementation specific.) Because slices are designed to aid in error recovery from bitstream errors, the value you choose should be dictated by your assessment of the likelihood that data will be corrupted. If bitstream errors will be rare, one slice per picture may be sufficient. On the other hand, if errors will be common, one slice per row of macroblocks may be needed. Don't request more slices than you need because there is a certain amount of overhead—for instance, a 40-bit slice header—associated with each slice.

The code below shows COMPRESSOR_SLICES_PER_PICTURE being set to 13.

```
XilCis cis;
int slices = 13;

xil_cis_set_attribute(cis, "COMPRESSOR_SLICES_PER_PICTURE",
    (void *)slices);
```

You can also read the value of this attribute. To do this, you might use the code shown below.

```
XilCis cis;
int slices;

xil_cis_get_attribute(cis, "COMPRESSOR_SLICES_PER_PICTURE",
    (void **)&slices);
```

If slices contains 0 after this call, the value of the attribute is set to an implementation-specific default.

COMPRESSOR_TIME_CODE

This attribute enables you to associate a time stamp with the first picture in the next group of pictures that is encoded. This time stamp contains the same information as a SMPTE time stamp.

Before actually setting the attribute, you must declare and assign values to the members of a structure of type XilMpeg1TimeCode.

```
typedef struct {
    Xil_boolean drop_frame_flag;
    Xil_unsigned32 hours;
    Xil_unsigned32 minutes;
    Xil_unsigned32 seconds;
    Xil_unsigned32 pictures;
} XilMpeg1TimeCode;
```

An application would obtain the values for most of the members of this structure—`hours`, `minutes`, `seconds`, and `pictures`—by reading a SMPTE time code in the video being encoded. The member `hours` can have a value in the range 0 to 23, and `minutes` and `seconds` can have values in the range 0 to 59. The member `pictures`, where a picture is a subdivision of a second, can also have a value in the range 0 to 59.

The other member in the structure, `drop_frame_flag`, should set to 0 unless the picture rate for the bitstream is 29.97. In this case, the flag can be set to 0 or 1. A setting of 0 indicates that the bitstream should be played back as if the picture rate were 30 pictures per second. A setting of 1 indicates that a play-back application should not count certain pictures so that movie time will not get ahead of wall clock time. Specifically, the application should not count pictures 0 and 1 at the beginning of each minute except minutes 0, 10, 20, 30, 40, and 50.

The code below sets `COMPRESSOR_TIME_CODE` so that the time stamp 19:07:34:13 is associated with the next group of pictures encoded. The drop-frame flag is set to 0.

```
XilCis cis;
XilMpeg1TimeCode time_code = {0, 19, 7, 34, 13};

xil_cis_set_attribute(cis, "COMPRESSOR_TIME_CODE",
    (void *)&time_code);
```

To read the value of `COMPRESSOR_TIME_CODE`, you would use code similar to the following.

```
XilCis cis;
XilMpeg1TimeCode *time_code;

xil_cis_get_attribute(cis, "COMPRESSOR_TIME_CODE",
    (void **)&time_code);
```

After this call, `time_code` will point to a structure of type `XilMpeg1TimeCode` that contains the current time-code information, or it will be a null pointer, indicating that `COMPRESSOR_TIME_CODE` has not been set. If the library successfully creates a time-code structure, your application is responsible for freeing the structure's storage.

Decompression Attributes

Only one of the attributes discussed in this section—DECOMPRESSOR_QUALITY—affects how the decoder works. The remaining attributes simply return information from the bitstream being decoded.

DECOMPRESSOR_QUALITY

The value of this attribute provides a hint to the decompressor concerning how it should handle the trade-off between the quality of reconstructed pictures and the speed of decoding those pictures. An XIL-compliant decoder need not act on this hint.

The valid values for this attribute are integers in the range 1 to 100. A value of 100 is a request that the decoder produce the highest quality pictures possible, and a value of 1 is a request that the decompressor decode pictures as fast as possible. By default, DECOMPRESSOR_QUALITY is set to 100.

The code below shows DECOMPRESSOR_QUALITY being set to 90. This setting usually provides satisfactory quality and may result in a significant increase in speed (compared to a setting of 100).

```
XilCis cis;
int quality = 90;

xil_cis_set_attribute(cis, "DECOMPRESSOR_QUALITY",
    (void *)quality);
```

To read the value of this attribute, use code similar to that shown below.

```
XilCis cis;
int quality;

xil_cis_get_attribute(cis, "DECOMPRESSOR_QUALITY",
    (void **)&quality);
```

DECOMPRESSOR_BROKEN_LINK

This is a read-only attribute that indicates whether B pictures at the beginning of the current group of pictures (before any I pictures) can be decoded correctly. Generally, of course, these pictures can be decoded correctly. However, it's possible that during an editing session, the preceding group of pictures—or maybe even just the last I or P picture in that group—was removed. If that type of editing takes place and a B picture at the beginning of the next group of pictures contains macroblocks that were forward predicted or bidirectionally predicted, the decoder will not be able to decode the complete picture. In this case, a broken-link flag should have been set in the group-of-pictures header at the time of the editing.

To read the value of this attribute, you might use code similar to this.

```
XilCis cis;
Xil_boolean broken_link;

xil_cis_get_attribute(cis, "DECOMPRESSOR_BROKEN_LINK",
    (void **)&broken_link);
```

If the broken-link flag for the group of pictures currently being decoded is set, the value of `broken_link` will be `TRUE`. In this case, any B pictures at the beginning of the group cannot be decoded correctly. If the flag is not set, the value will be `FALSE`.

DECOMPRESSOR_CLOSED_GOP

This is a read-only attribute that indicates whether the group of pictures currently being decoded is a closed group or an open group. A closed group is one that begins with an I picture or with one or more B pictures that can be decoded without reference to the last picture in the preceding group. This type of group can be decoded independently. An open group, on the other hand, begins with one or more B pictures, at least one of which contains macroblocks that are forward or bidirectionally predicted. Such a B picture cannot be decoded if the preceding group of pictures is not available (for example, if it has been edited out of the bitstream).

To read the value of this attribute, you might use code similar to this.

```
XilCis cis;
Xil_boolean closed_gop;

xil_cis_get_attribute(cis, "DECOMPRESSOR_CLOSED_GOP",
    (void **)&closed_gop);
```

If the closed-group-of-pictures flag for the group is set, the value of `closed_gop` will be `TRUE`. In this case, any B pictures at the beginning of the group can be decoded independently of the preceding group. If the flag is not set, the value will be `FALSE`.

DECOMPRESSOR_FRAME_TYPE

This is a read-only attribute whose value describes the type of the last picture decoded. This type can be any one of the types included in the following enumeration:

```
typedef enum {
    I,      /* intraframe picture */
    P,      /* forward predicted picture */
    B,      /* bidirectionally predicted picture */
    D       /* DC picture */
} XilMpeg1FrameType;
```

For a description of I, P, and B pictures, see the sections "I Pictures and P Pictures" on page 361 and "B Pictures" on page 361.

The code below illustrates how you might read the value of `DECOMPRESSOR_FRAME_TYPE`.

```
XilCis cis;
XilMpeg1FrameType frame_type;

xil_cis_get_attribute(cis, "DECOMPRESSOR_FRAME_TYPE",
    (void **)&frame_type);
```

DECOMPRESSOR_PEL_ASPECT_RATIO_VALUE

This is a read-only attribute whose value is the pixel aspect ratio stored in the sequence header for the MPEG-1 sequence you're decoding. This value, along with the width and height of the pictures in the sequence, defines the shape of a decoded picture.

The value stored in this attribute will be one of the following constants.

- 0.6735
- 0.7031
- 0.7615
- 0.8055
- 0.8437
- 0.8935
- 0.9157
- 0.9815
- 1.0
- 1.0255
- 1.0695
- 1.0950
- 1.1575
- 1.2015

The key values in this list are 1.0, 0.9157, and 1.0950. A value of 1.0 indicates that the pixels are square and that the pictures in the sequence are meant to be displayed on computer monitors. The value 0.9157 means that decoded pictures can be displayed without distortion on 625-line 50 Hz televisions, and the value 1.0950 indicates that decoded pictures can be displayed without distortion on 525-line 60 Hz televisions.

To read the value of this attribute, use code similar to that shown below.

```
XilCis cis;
float aspect_ratio;

xil_cis_get_attribute(cis,
    "DECOMPRESSOR_PEL_ASPECT_RATIO_VALUE",
    (void **)&aspect_ratio);
```

DECOMPRESSOR_PICTURE_RATE_VALUE

The value of this read-only attribute is the picture-rate value from the sequence header of the MPEG-1 sequence you're decoding. This picture-rate value is the number of pictures that should be decoded and displayed each second.

The value will be one of the constants listed below.

- 23.976
- 24.0
- 25.0
- 29.97
- 30.0
- 50.0
- 59.94
- 60.0

You can use code similar to that shown below to read the value of this attribute.

```
XilCis cis;
float picture_rate;

xil_cis_get_attribute(cis, "DECOMPRESSOR_PICTURE_RATE_VALUE",
    (void **)&picture_rate);
```

DECOMPRESSOR_TEMPORAL_REFERENCE

Because the pictures in a group of pictures may need to be decoded in one order and displayed in another order, each picture contains a temporal-reference value. This value indicates a picture's place in the display order for the group. Picture 0 is meant to be displayed first, picture 1 to be displayed second, and so on. The maximum value the temporal-reference field can accommodate is 1023; therefore, if a group of pictures contains more than 1024 pictures, the counter is reset to 0 for picture 1025 (or more generally, for picture (1024 * *n*) + 1).

The attribute DECOMPRESSOR_TEMPORAL_REFERENCE contains the temporal-reference value for the most recently decompressed picture. To read the value of this attribute, you might use code similar to this.

```
XilCis cis;
int temporal_reference;

xil_cis_get_attribute(cis, "DECOMPRESSOR_TEMPORAL_REFERENCE",
    (void **)&temporal_reference);
```

Note – In a legal MPEG-1 bitstream, the temporal reference values for the pictures in a group must be consecutive values (except in the case where the counter resets). However, it is not uncommon for bitstreams that have been edited to violate this requirement. That is, if the pictures with temporal-reference values between 12 and 24 have been cut from a group, you may find that the picture with a temporal of 11 is followed by one with a temporal reference of 25.

DECOMPRESSOR_TIME_CODE

One field in a group-of-pictures header is a SMPTE time code. This time code is associated with the first picture (in display order) in the group. When you read the attribute DECOMPRESSOR_TIME_CODE, the decoder places information about this time code for the current group of pictures in a structure of type XilMpeg1TimeCode.

```
typedef struct {
    Xil_boolean drop_frame_flag;
    Xil_unsigned32 hours;
    Xil_unsigned32 minutes;
    Xil_unsigned32 seconds;
    Xil_unsigned32 pictures;
} XilMpeg1TimeCode;
```

The member hours will have a value in the range 0 to 23; minutes, seconds, and pictures will have values in the range 0 to 59.

You can read the value of this attribute using code similar to this.

```
XilCis cis;
XilMpeg1TimeCode *time_code;

xil_cis_get_attribute(cis, "DECOMPRESSOR_TIME_CODE",
    (void **)&time_code);
```

Your application is responsible for freeing the storage the library allocates to hold the time-code structure.

MPEG-1 Molecules

The XIL library includes a series of *molecules* that accelerate the playback of MPEG-1 bitstreams. 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.

For example, if your program calls `xil_decompress()` to decode an image stored in an MPEG-1 CIS and then calls `xil_ordered_dither()` to dither 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 dithering in an optimized way.

For information about the MPEG-1 molecules that are available and information about how to call those molecules, see the section "XIL Molecules" on page 398.