JPEG Baseline Sequential Codec 16

The JPEG baseline sequential coder-decoder is one of the digital-image codecs that has been specified by the Joint Photographic Experts Group, a joint ISO and CCITT technical committee. The JPEG lossless compressor discussed in Chapter 17, "JPEG Lossless Codec," is another of these codecs. Taken together, the JPEG codecs are meant to provide a standard means of compressing still continuous-tone (grayscale and color) images.

The JPEG standard was originally developed for use in areas such as desktop publishing, graphic arts, medical imaging, and document imaging, where the archiving of still images is important. However, the introduction of high-performance hardware capable of coding and decoding JPEG images in real-time has enabled the development of full-motion video applications based on JPEG.

This chapter is divided into four sections. The first section explains how the compressor works and the type of applications it was designed for. The second explains briefly how to create a JPEG baseline sequential CIS. The third discusses CIS attributes that apply specifically to the baseline sequential codec (as opposed to the general CIS attributes covered in the section "General CIS Attributes" on page 257). The fourth introduces the subject of accelerating the playback of JPEG bitstreams. For further information on this subject, see Chapter 21, "Acceleration in XIL Programs."

16

How the JPEG Baseline Sequential Codec Works

The JPEG baseline sequential compressor is one of the DCT-based compressors, which also include the MPEG-1 and H.261 compressors. Figure 16-1 shows the basic steps the compressor uses to compress an image.

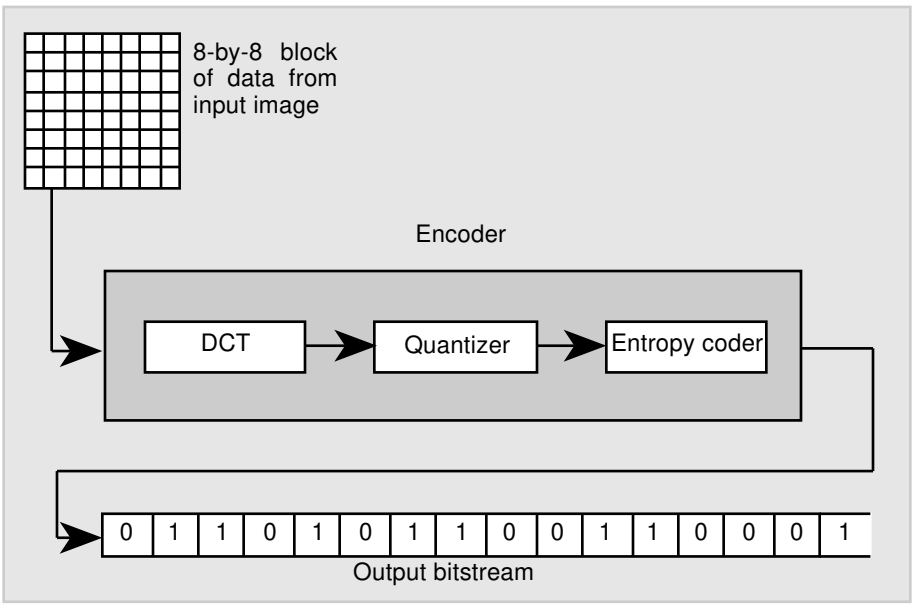

Figure 16-1 JPEG Baseline Sequential Compressor

As the figure indicates, the input to the encoder is an 8-by-8 block of samples from the image being compressed. The compressor encodes each block of data in an image by:

- Performing a Discrete Cosine Transform (DCT) on the 8-by-8 block of data
- Quantizing the results of the DCT
- Entropy coding the results of the quantization step

Each of these steps is considered in more detail in the sections below.

Discrete Cosine Transform

The Discrete Cosine Transform is a mathematical operation that takes a block of image samples as its input and converts the information in that input from the spatial domain to the frequency domain. For example, in JPEG, the input to the DCT is an 8-by-8 matrix whose values represent brightness levels at particular x, y coordinates, and the output is an 8-by-8 matrix whose values represent relative amounts of the 64 spatial frequencies that make up the input data's spectrum. In the output matrix, information about the lowest frequencies is stored in the upper-left corner, and information about the highest frequencies is stored in the lower-right corner. See Figure 16-2.

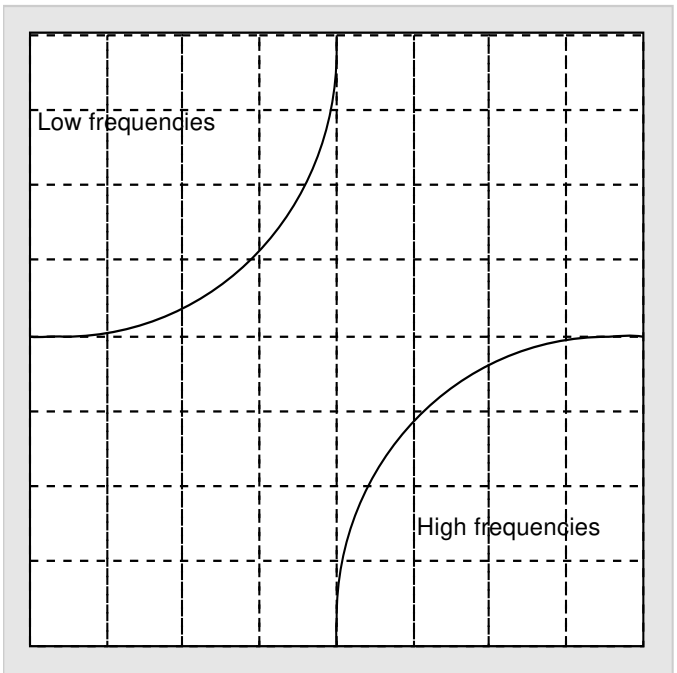

Figure 16-2 Output of the Discrete Cosine Transform

This transformation provides a strong basis for compression because in a typical block of input, low spatial frequencies far outweigh high spatial frequencies. As a result, most of the values in the output matrix, outside of those in the upper-left corner, will have values close to 0 and will end up not being encoded.

Quantization

Quantization is the simplest step in the encoder's algorithm. It simply involves dividing each value in the matrix output by the DCT by the corresponding value in a quantization table.

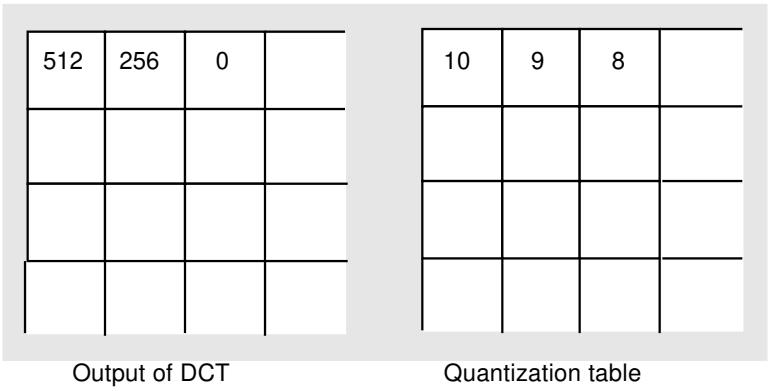

Figure 16-3 Quantization in the JPEG Encoder

Figure 16-3 shows part of an 8-by-8 block of values output by the Discrete Cosine Transform and part of a table to be used in quantizing this data. In this example, the quantizer will divide 512 by 10 and then round 51.2 off to 51. Likewise, it will divide 256 by 9 and 0 by 8. During quantization, any value in the matrix on the left that is divided by a number greater than itself times 2 will go to 0 and will not be encoded in the JPEG bitstream. The quantizer is the part of the JPEG baseline sequential encoder that causes the encoder to be a lossy one.

Entropy Coding

Once a block of data has been quantized, it enters an entropy coder, which creates the actual JPEG bitstream. The entropy coder first looks at the values in the quantized block of data in the zigzag sequence shown in Figure 16-4. Using this sequence ensures that the encoder will encounter all nonzero values in the block as early as possible.

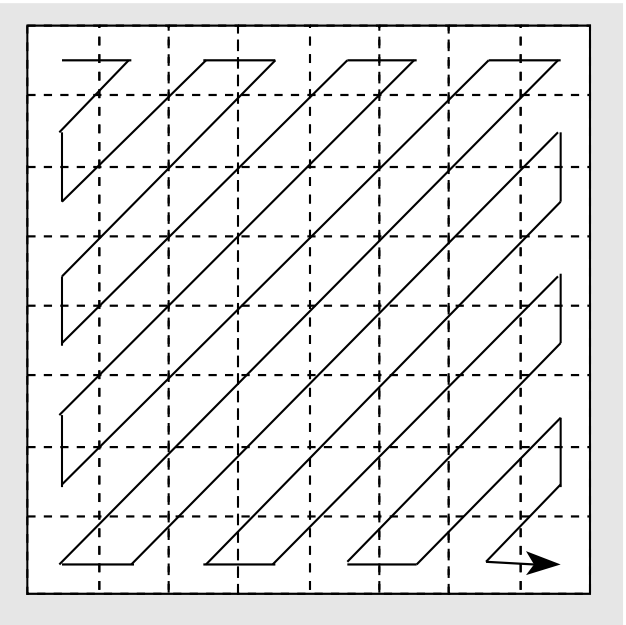

Figure 16-4 Zigzag Sequencing in JPEG Encoder

As the entropy encoder moves through the values in this order, it records three pieces of information each time it encounters a nonzero value: the number of 0's it passed over before finding the nonzero value, the number of bits it will take to encode the nonzero value, and the value itself. The first two pieces of information are considered a pair. For example, if the encoder skips over 7 zeros and then finds a 3, the pair will be 7/2 (2 is number of bits required to encode the value 3). The encoder then consults a Huffman table to find the bit sequence that represents the pair 7/2 and writes this bit sequence to the JPEG bitstream. It then encodes the value 3 using a variable-length code. The

encoder continues this process until all the remaining values in the block are 0's, at which point, it writes a special end-of-block bit sequence to the bitstream.

Creating a JPEG Baseline Sequential CIS

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

```
XilCis cis;
XilSystemState state;

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

JPEG Baseline Sequential 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 JPEG baseline sequential codec. You set codec-specific attributes using the function `xil_cis_set_attribute()`, and you read them using the function `xil_cis_get_attribute()`.

The JPEG baseline sequential 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 the JPEG baseline sequential compressor compresses images.

BAND_HUFFMAN_TABLE

This attribute enables you to associate a particular Huffman table with a particular band of your image. As shown in Table 16-2 on page 318, the JPEG compressor supports four tables (0 to 3) for both types of DCT coefficients (DC and AC). Table 16-1 shows the default relationship between image bands and Huffman tables.

Table 16-1 Image Bands and Huffman Tables

Table	Type	Band
0	DC	Used to encode the DC coefficient in band 0
1	DC	Used to encode the DC coefficients in all bands except band 0
2	DC	Not used
3	DC	Not used
0	AC	Used to encode the AC coefficients in band 0
1	AC	Used to encode the AC coefficients in all bands except band 0
2	AC	Not used
3	AC	Not used

Note – Tables 2 and 3, for both DC and AC coefficients, can be used for JPEG extended-baseline-sequential mode encoding. Bear in mind, though, that many JPEG decompressors will not be able to decompress the extended-mode bitstream.

Before calling `xil_cis_set_attribute()` to change one of these relationships, you must declare and assign values to the members of a structure of type `XilJpegBandHTable`.

```
typedef struct {
    int band;
    int table;
    XilJpegHTableType type;
} XilJpegBandHTable;
```

The member band can have a value in the range 0 to 255 and represents the image band to be associated with a different Huffman table. (The first band in an image is band 0.) The member table can be 0 to 3, and the member type can be set to DC or AC (which are enumeration constants of type XilJpegHTableType). The setting of type determines whether the table you specify will be used to encode DC or AC coefficients in the band you selected.

The code fragment below associates table 0 (type DC) with band 1 of the image to be encoded.

```
XilCis cis;
XilJpegBandHTable table_for_band = {1, 0, DC};

xil_cis_set_attribute(cis, "BAND_HUFFMAN_TABLE",
    (void *)&table_for_band);
```

BAND_QUANTIZER

The BAND_QUANTIZER attribute enables you to specify the quantization table that will be used in encoding a particular band of an image. Before setting this attribute, you must declare and assign values to the members of a structure of type XilJpegBandQTable.

```
typedef struct {
    int band;
    int table;
} xilJpegBandQTable;
```

The integer band can have a value in the range 0 to 255. In a YC_bC_r image, the Y band is band 0, the C_b band is band 1, and the C_r band is band 2.

The integer table can have a value in the range 0 to 3. By default, table 0 contains the values shown in Table K.1 of Annex K of the ISO JPEG specification, table 1 contains the values shown in Table K.2 of Annex K, and tables 2 and 3 are not loaded. You can load tables 2 and 3 or change the values in tables 0 and 1 using the attribute QUANTIZATION_TABLE. See the section "QUANTIZATION_TABLE" on page 322.

By default, band 0 is associated with table 0, and all other bands are associated with table 1. The example code below shows band 2 being associated with table 2.

```
XilCis cis;
XilJpegBandQTable band_to_table;

band_to_table.band = 2;
band_to_table.table = 2;
xil_cis_set_attribute(cis, "BAND_QUANTIZER",
    (void *)&band_to_table);
```

BYTES_PER_FRAME

BYTES_PER_FRAME is a read-only attribute that tells you the number of bytes in the last compressed image written to the CIS. The value of this attribute is 0 if no data has been written to the CIS.

BYTES_PER_FRAME is useful for helping you select an appropriate setting for the COMPRESSION_QUALITY attribute. (See the section "COMPRESSION_QUALITY" on page 314.) If the number of bytes being used to store a compressed image is too high, you can lower it by lowering the value of COMPRESSION_QUALITY, and vice versa.

The code fragment below reads the value of the BYTES_PER_FRAME attribute.

```
XilCis cis;
int bytes_per_frames;

xil_cis_get_attribute(cis, "BYTES_PER_FRAME",
    (void **)&bytes_per_frame);
```

COMPRESSED_DATA_FORMAT

COMPRESSED_DATA_FORMAT is a set-only attribute that defines the format of the JPEG compressor's output. It can be set to either INTERCHANGE or ABBREVIATED_FORMAT.

Setting the attribute to INTERCHANGE causes the compressor to produce output in JPEG interchange format. In this format, the quantization tables and Huffman tables required by the decompressor are included in each compressed frame.

The default value, ABBREVIATED_FORMAT, causes the compressor to produce output in JPEG abbreviated format. In this format, quantization tables and Huffman tables are not included in a compressed frame if the tables needed to decompress that frame have been defined in a previous frame in the sequence. If a table changes during the course of the sequence, the new table definition is included in the first compressed frame that uses the new table values.

Note – The compressor does not enable you to produce abbreviated-table output in which frames contain only table specifications. However, the decompressor will accept bitstreams in this format.

The code fragment below shows COMPRESSED_DATA_FORMAT being set to INTERCHANGE.

```
XilCis cis;

xil_cis_set_attribute(cis, "COMPRESSED_DATA_FORMAT",
    (void *)INTERCHANGE);
```

COMPRESSION_QUALITY

The COMPRESSION_QUALITY attribute tells the compressor how it should handle the trade-off between image quality and compression ratio. The attribute can be set to any value between 1 and 100. Setting the attribute to 100 is a request that the compressor produce very high quality images, even though this will mean a lower compression ratio. A setting of 1 tells the compressor to increase its compression ratio, even though the result will be lower image quality. By default, this attribute is set to 50.

Studies by the JPEG committee indicate that for color images of average complexity, the following relationships exist between level of compression and image quality:

- 0.25-0.5 bits/pixel: moderate to good quality, sufficient for some applications;
- 0.5-0.75 bits/pixel: good to very good quality, sufficient for many applications;
- 0.75-1.5 bits/pixel: excellent quality, sufficient for most applications;
- 1.5-2.0 bits/pixel: usually indistinguishable from the original, sufficient for the most demanding applications.[1]

Also, see Color Plate 4. It shows an image compressed at a high quality setting (95) and one compressed at a low quality setting (5).

The setting of the COMPRESSION_QUALITY attribute controls image quality and the compression ratio by determining a scaling factor the compressor will use in creating scaled versions of its quantization tables. These scaled tables are used during compression. A COMPRESSION_QUALITY value of 50 results in a scaling factor of 1; that is, the scaled tables are identical to the original tables. A COMPRESSION_QUALITY value greater than 50 results in a scaling factor that is less than 1, and a value less than 50 results in a scaling factor greater than 1.

Note – The trade-off between image quality and compression ratio can also be affected by the values stored in the compressor's quantization tables. See the section "QUANTIZATION_TABLE" on page 322.

The code fragment below shows the COMPRESSION_QUALITY attribute being set to 100.

```
XilCis cis
int quality = 100;

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

1. G. K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, April 1991, p. 35.

ENCODE_INTERLEAVED

ENCODE_INTERLEAVED is a set-only attribute that determines whether the compressor produces an interleaved bitstream when compressing multiband images. If the attribute is set to TRUE, the default value, the compressor produces an interleaved bitstream, and if the attribute is set to FALSE, the compressor produces a noninterleaved bitstream.

This attribute is ignored if the image being compressed has more than four bands because the bitstream for such images is never interleaved. Also, note that the attribute ENCODE_411_INTERLEAVED takes precedence over this attribute if the image being compressed has three bands.

In the example below, ENCODE_INTERLEAVED is being set to FALSE.

```
XilCis cis;

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

ENCODE_411_INTERLEAVED

ENCODE_411_INTERLEAVED is a set-only attribute that affects how the compressor handles 3-band images and is intended specifically for use with YC_bC_r images (not RGB images). If the input to the compressor is an image of a different type, this attribute should be set to FALSE (its default value).

Note – If the input images are not 3-band images and this attribute is set to TRUE, the compressor will operate as if the attribute were set to FALSE.

If the input is a YC_bC_r image and ENCODE_411_INTERLEAVED is set to TRUE, the compressor:

- Subsamples the data in the color bands so that only one color value will be encoded for each four color values in the original (4:1:1)
- Interleaves the data for the three image bands on a macroblock basis, where a macroblock is defined as a 16-by-16 block of pixels

---

Note – This is the only method the XIL library provides for producing JPEG bitstreams with subsampled bands.

If you are making a movie from YC_bC_r images, `ENCODE_411_INTERLEAVED` should be set to `TRUE` because the XIL library's accelerated routines for playing back JPEG movies require input in the format described above. The `ENCODE_411_INTERLEAVED` attribute tells the JPEG compressor to subsample the u and v data instead of using the evenly sampled bands that the XIL software hands it. The output of a decompression (and later export) is still an evenly sampled 3-banded image with the u and v duplicated.

If the attribute is set to `FALSE`, the setting of the `ENCODE_INTERLEAVED` attribute determines the format of the JPEG bitstream.

In the example below, `ENCODE_411_INTERLEAVED` is being set to `TRUE`.

```
XilCis cis;

xil_cis_set_attribute(cis, "ENCODE_411_INTERLEAVED",
    (void *)TRUE);
```

HUFFMAN_TABLE

This attribute enables you to supply the compressor with a Huffman table to use in encoding quantized DC or AC coefficients. Before calling `xil_cis_set_attribute()` to set this attribute, you must store information about the Huffman table in a structure of type `XilJpegHTable`.

```
typedef struct {
    int table;
    XilJpegHTableType type;
    XilJpegHTableValue *value;
} XilJpegHTable;
```

The member `table` must be set to 0 to 3, and the member `type` can be set to `DC` or `AC`. In a Huffman table of type `DC`, the array `value` contains 16 elements. In a table of type `AC`, this array contains 256 elements.

The default contents of the eight Huffman tables are shown in Table 16-2.

Table 16-2 Default Huffman Tables

Table	Type	Default Contents
0	DC	Contains the values specified in Table K.3 of the ISO JPEG specification. These values are useful for encoding the DC coefficients of the luminance band of 8-bit YC_bC_r images.
1	DC	Contains the values specified in Table K.4 of the ISO JPEG specification. These values are useful for encoding the DC coefficients of the chrominance bands of 8-bit YC_bC_r images.
2	DC	Empty.
3	DC	Empty.
0	AC	Contains the values specified in Table K.5 of the ISO JPEG specification. These values are useful for encoding the AC coefficients of the luminance band of 8-bit YC_bC_r images.
1	AC	Contains the values specified in Table K.6 of the ISO JPEG specification. These values are useful for encoding the AC coefficients of the chrominance bands of 8-bit YC_bC_r images.
2	AC	Empty.
3	AC	Empty.

Note – Tables 2 and 3, for both DC and AC coefficients, can be used for JPEG extended-baseline-sequential mode encoding. Bear in mind, though, that many JPEG decompressors will not be able to decompress the extended-mode bitstream.

The final member of the `XilJpegHTable` structure, `value`, is an array of structures of type `XilJpegHTableValue`.

```
typedef struct {
    int bits;
    int pattern;
} XilJpegHTableValue;
```

Each structure of this type defines a pair of values to be written to the Huffman table. The member `bits` is the length of a Huffman code in bits, and `pattern` contains the code itself (the `bits` least significant bits of `pattern` are the code).

The examples below indicate how the structures referred to above are used in setting up a Huffman table. This first example shows table 0 (type `DC`) being loaded with values suitable for encoding DC coefficients.

```
XilCis cis;
XilJpegHTable huffman_table;

XilJpegHTableValue huffman_values[16] = {
    {2, 0x0000}, {3, 0x0002}, {3, 0x0003}, {3, 0x0004},
    {3, 0x0005}, {3, 0x0006}, {4, 0x000e}, {5, 0x001e},
    {6, 0x003e}, {7, 0x007e}, {8, 0x00fe}, {9, 0x01fe},
    {0, 0x0000}, {0, 0x0000}, {0, 0x0000}, {0, 0x0000}
};

huffman_table.table = 0;
huffman_table.type = DC;
huffman_table.value = huffman_values;
xil_cis_set_attribute(cis, "HUFFMAN_TABLE",
    (void *)&huffman_table);
```

Note that there are only twelve meaningful value pairs being loaded into the table. The last four are there to fill out `huffman_table.value`, which is an array of sixteen structures.

This next example shows table 0 (type `AC`) being loaded with values suitable for encoding AC coefficients.

```
XilCis cis;
XilJpegHTable huffman_table;

XilJpegHTableValue huffman_values[256] = {
    {4, 0x000a}, {2, 0x0000}, {2, 0x0001}, {3, 0x0004},
    {4, 0x000b}, {5, 0x001a}, {7, 0x0078}, {8, 0x00f8},
    {10, 0x03f6}, {16, 0xff82}, {16, 0xff83}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {4, 0x000c}, {5, 0x001b}, {7, 0x0079},
    {9, 0x01f6}, {11, 0x07f6}, {16, 0xff84}, {16, 0xff85},
```

```
    {16, 0xff86}, {16, 0xff87}, {16, 0xff88}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {5, 0x001c}, {8, 0x00f9}, {10, 0x03f7},
    {12, 0x0ff4}, {16, 0xff89}, {16, 0xff8a}, {16, 0xff8b},
    {16, 0xff8c}, {16, 0xff8d}, {16, 0xff8e}, {0, 0x0000},
    {0, 0x0000},{0, 0x000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {6, 0x003a}, {9, 0x01f7}, {12, 0x0ff5},
    {16, 0xff8f}, {16, 0xff90}, {16, 0xff91}, {16, 0xff92},
    {16, 0xff93}, {16, 0xff94}, {16, 0xff95}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {6, 0x003b}, {10, 0x03f8}, {16, 0xff96},
    {16, 0xff97}, {16, 0xff98}, {16, 0xff99}, {16, 0xff9a},
    {16, 0xff9b}, {16, 0xff9c}, {16, 0xff9d}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},


    {0, 0x0000}, {7, 0x007a}, {11, 0x07f7}, {16, 0xff9e},
    {16, 0xff9f}, {16, 0xffa0}, {16, 0xffa1}, {16, 0xffa2},
    {16, 0xffa3}, {16, 0xffa4}, {16, 0xffa5}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {7, 0x007b}, {12, 0x0ff6}, {16, 0xffa6},
    {16, 0xffa7}, {16, 0xffa8}, {16, 0xffa9}, {16, 0xffaa},
    {16, 0xffab}, {16, 0xffac}, {16, 0xffad}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {8, 0x00fa}, {12, 0x0ff7}, {16, 0xffae},
    {16, 0xffaf}, {16, 0xffb0}, {16, 0xffb1}, {16, 0xffb2},
    {16, 0xffb3}, {16, 0xffb4}, {16, 0xffb5}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {9, 0x01f8}, {15, 0x7fc0}, {16, 0xffb6},
    {16, 0xffb7}, {16, 0xffb8}, {16, 0xffb9}, {16, 0xffba},
    {16, 0xffbb}, {16, 0xffbc}, {16, 0xffbd}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {9, 0x01f9}, {16, 0xffbe}, {16, 0xffbf},
    {16, 0xffc0}, {16, 0xffc1}, {16, 0xffc2}, {16, 0xffc3},
    {16, 0xffc4}, {16, 0xffc5}, {16, 0xffc6}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {9, 0x01fa}, {16, 0xffc7}, {16, 0xffc8},
```

```
    {16, 0xffc9}, {16, 0xffca}, {16, 0xffcb}, {16, 0xffcc},
    {16, 0xffcd}, {16, 0xffce}, {16, 0xffcf}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {10, 0x03f9}, {16, 0xffd0}, {16, 0xffd1},
    {16, 0xffd2}, {16, 0xffd3}, {16, 0xffd4}, {16, 0xffd5},
    {16, 0xffd6}, {16, 0xffd7}, {16, 0xffd8}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {10, 0x03fa}, {16, 0xffd9}, {16, 0xffda},
    {16, 0xffdb}, {16, 0xffdc}, {16, 0xffdd}, {16, 0xffde},
    {16, 0xffdf}, {16, 0xffe0}, {16, 0xffe1}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {11, 0x07f8}, {16, 0xffe2}, {16, 0xffe3},
    {16, 0xffe4}, {16, 0xffe5}, {16, 0xffe6}, {16, 0xffe7},
    {16, 0xffe8}, {16, 0xffe9}, {16, 0xffea}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {0, 0x0000}, {16, 0xffeb}, {16, 0xffec}, {16, 0xffed},
    {16, 0xffee}, {16, 0xffef}, {16, 0xfff0}, {16, 0xfff1},
    {16, 0xfff2}, {16, 0xfff3}, {16, 0xfff4}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000},

    {11, 0x07f9}, {16, 0xfff5}, {16, 0xfff6}, {16, 0xfff7},
    {16, 0xfff8}, {16, 0xfff9}, {16, 0xfffa}, {16, 0xfffb},
    {16, 0xfffc}, {16, 0xfffd}, {16, 0xfffe}, {0, 0x0000},
    {0, 0x0000},{0, 0x0000},{0, 0x0000},{0, 0x0000}
};

huffman_table.table = 0;
huffman_table.type = AC;
huffman_table.value = huffman_values;
xil_cis_set_attribute(cis, "HUFFMAN_TABLE",
    (void *)&huffman_table);
```

Note that `huffman_values` is an array of 256 structures of type `XilJpegHTableValue`, and that the structures are organized as 16 groups of 16 structures. Because there are not 16 bit lengths (`bits`) for each of the 16 run lengths, each group of 16 structures contains 5 meaningless structures at the end.

OPTIMIZE_HUFFMAN_TABLES

You can use this attribute to tell the compressor to generate optimal Huffman tables instead of using the default tables included in the ISO specification or tables you loaded earlier. When the compressor creates optimal Huffman tables, its Huffman tables can vary from image to image, and you should see higher rates of compression.

If `OPTIMIZE_HUFFMAN_TABLES` is set to `FALSE`, its default value, the compressor uses fixed Huffman tables for each image in a sequence. If the attribute is set to `TRUE`, the compressor uses optimal Huffman tables.

Note – If an application sets `OPTIMIZE_HUFFMAN_TABLES` to `TRUE` and then later sets it to `FALSE`, the compressor loads a default set of tables. It does not restore the tables it was using when the attribute was first set to `TRUE`.

The following code fragment shows `OPTIMIZE_HUFFMAN_TABLES` being set to `TRUE`.

```
XilCis cis;

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

QUANTIZATION_TABLE

The `QUANTIZATION_TABLE` attribute enables you to load values into one of four quantization tables that the compressor may use to quantize DCT coefficients. Prior to actually setting the attribute you must declare and assign values to the members of an `XilJpegQTable` structure.

```
typedef struct {
    int table;
    int (*value)[8];
} XilJpegQTable;
```

The integer `table` must have a value between 0 and 3. This value indicates which of the four quantization tables you want to load. By default, table 0 contains the values shown in Table K.1 of Annex K of the ISO JPEG

specification. These values are designed to be used with the luminance band of 8-bit YCC images. Likewise, table 1 contains the values shown in Table K.2 of Annex K. The values in this table are designed to be used with the chrominance bands of 8-bit YCC images. By default, tables 2 and 3 are not loaded.

You use the pointer `value` (a pointer to an array of 8 integers) to point to the 64 quantization values that you want to load into the table.

The code fragment below illustrates 64 quantization values being loaded into table 2.

```
Xilcis cis;
XilJpegQTable quantization;
int quantization_array[8][8] = {
    {16, 11, 10, 16, 24, 40, 51, 61},
    {12, 12, 14, 19, 26, 58, 60, 55),
    {14, 13, 16, 24, 40, 57, 69, 56},
    {14, 17, 22, 29, 51, 87, 80, 62},
    {18, 22, 37, 56, 68, 109, 103, 77},
    {24, 35, 55, 64, 81, 104, 113, 92},
    {49, 64, 78, 87, 103, 121, 120, 101},
    {72, 92, 95, 98, 112, 100, 103, 99}
};

quantization.table = 2;
quantization.value = quantization_array;
xil_cis_set_attribute(cis, "QUANTIZATION_TABLE",
    (void *)&quantization);
```

TEMPORAL_FILTERING

This attribute controls whether the JPEG compressor filters 3-band images before it encodes them. If the attribute is set to `FALSE`, its default value, the compressor does not filter the images, and if it is set to `TRUE`, the compressor does filter them. This filtering can help in cases where the images being encoded contain a lot of noise; however, the filtering can also have the undesirable side effect of producing ghosting artifacts.

The example below shows the attribute being set to TRUE.

```
XilCis cis;

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

And this fragment demonstrates how to read the attribute's value.

```
XilCis cis;
Xil_boolean filtering_enabled;

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

Decompression Attributes

Setting the following attribute affects how the JPEG baseline sequential codec decompresses images.

DECOMPRESSION_QUALITY

You use the DECOMPRESSION_QUALITY attribute to provide a hint to the compressor concerning how it should handle the trade-off between the quality of decompressed images and speed of decompression. The decompressor is free to ignore this hint.

The attribute's value must be an integer in the range of 1 to 100. A setting of 100 is a request for a high level of image quality, and a setting of 1 is a request from a high playback speed. By default, the attribute is set to 100.

Note – The JPEG compressor increases playback speed by decreasing the number of quantized coefficients it uses in reconstructing an image. It drops high-frequency coefficients first.

The example code below shows DECOMPRESSION_QUALITY being set to 75.

```
XilCis cis;
int quality = 75;

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

IGNORE_HISTORY

The setting of this attribute affects what happens when you seek backward or forward in a JPEG CIS whose random-access flag is not set. If your CIS's random-access flag is set, the value of IGNORE_HISTORY is irrelevant: you can always seek backward or forward and get correct results. To determine whether the random-access flag is set, call the function xil_cis_get_random_access().

Note – When you create a JPEG CIS, its random-access flag is always set. This flag will remain set if the quantization and Huffman tables needed to decode the entire JPEG bitstream are encoded with the first image or if those tables are encoded with each image in the sequence. Otherwise, the decoder will clear the flag as soon as it discovers that the bitstream cannot be accessed randomly.

If your CIS's random-access bit is not set and IGNORE_HISTORY is FALSE (its default value), the following rules apply:

- Backward seeds are illegal.
- A forward seek will be successful; that is, the image you decode after the seek will be reconstructed properly. However, a forward seek with IGNORE_HISTORY set to FALSE may be slower than one with the attribute set to TRUE.

If your CIS's random-access bit is not set and IGNORE_HISTORY is TRUE, these rules apply:

- Backward seeks are legal. However, it is the responsibility of your application to seek to an image that can be decoded correctly. That is, the image you seek to must either define its own tables or depend on the tables that were most recently loaded into the decoder.

- You can also seek forward. However, your application should not seek forward past images that contain table definitions if those definitions will be needed to decode the image you're seeking to. The decoder does not ensure that these table definitions are loaded.

The code below shows `IGNORE_HISTORY` being set to `TRUE`.

```
XilCis cis;

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

JPEG Molecules

The XIL library includes a series of *molecules* that accelerate the playback of JPEG baseline sequential 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 a JPEG CIS and then calls `xil_ordered_dither()` 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 dithering in an optimized way.

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