Mars Orbiter Camera

Software Interface Specification
Narrow Angle and Wide Angle
Standard Data Products

M. Caplinger
Malin Space Science Systems, Inc.

Approved by:

____
M. Malin, Principal Investigator

September 1999 (revised for MGS)
(formatted April 7, 2000)

1. Introduction

1.1. Purpose

This document describes the format of the Mars Orbiter Camera (MOC, previously known as the Mars Observer Camera) Narrow Angle (NA) and Wide Angle (WA) Standard Data Products.

1.2. Scope

The format and content specifications in this SIS apply to all phases of the project for which this product is available.

1.3. Applicable Documents

Mars Global Surveyor Science Data Management Plan (JPL 542-310)

Mars Global Surveyor Project Archive Generation, Validation and Transfer Plan (JPL 542-312)

Mars Global Surveyor Project Data Management Plan (JPL 542-403)

Mars Observer Camera Software User's Guide (Part 1: Flight Software)

Mars Observer Camera Instrument Template (I-Kernel PDS document)

Mars Observer Project Archive Policy and Data Transfer Plan, 642-447.

Planetary Science Data Dictionary Document, PDS Version 2.0, May 1991.

Margaret Cribbs, "Comments on the MOC label", IOM# 361-92-MAC016, 2 April 1992.

1.4. Functional Description

1.4.1. Data Content Summary

Each MOC Standard Data Product is a single image in the compressed format as delivered from the instrument. The data have been depacketized and reformatted with standard labels, but are otherwise "raw"; that is, as received from the instrument. In that sense these products are most closely analogous to the Experiment Data Record (EDR) products of previous missions.

The only formatting differences between Narrow and Wide Angle products are the maximum possible width of the image (2048 for the Narrow Angle and 3456 for the Wide Angle) and the presence or values of some of the keywords; the products are otherwise formatted identically.

1.4.2. Source and Transfer Method

MOC products are produced by the *makepds* program from the format internally used at the MOC Mission Operations Facility (MOF). This program reads a raw MOC Science Data Protocol (MSDP) file (see the MOC Software User's Guide), extracts some information from its headers, formats and attaches the PDS labels, and appends the compressed fragment data.

It is expected that there will be two ways to receive MOC products: by electronic file transfer from the Planetary Data System, and on some archival medium such as CD-ROM.

1.4.3. Recipients and Utilization

These products will be available to MOC team members, the Mars Global Surveyor science community, the planetary science community, and other interested parties. Descriptions of data rights and proprietary periods are beyond the scope of this document, and are discussed in the Mars Global Surveyor Project Archive Policy and Data Transfer Plan, and in unique Operational Interface Agreements between the MOC Science Team and other parties.

These products will be used for engineering support, direct science analysis, or the construction of other science products.

1.4.4. Pertinent Relationships with Other Interfaces

See the MOC Software User's Guide for descriptions of other interfaces.

1.5. Assumptions and Constraints

Note that this file contains compressed image data. Decompression will result in a raw image that is not corrected for instrument signature, effects of spacecraft motion, or the effects of imaging geometry. Although there is enough information in the header to do some processing, for more sophisticated processing ancillary files will be required. These ancillary files are not described in this document. Examples of ancillary files are calibration files, viewing geometry files (e.g., SPICE kernels), image index tables, etc.

2. Environment

2.1. Hardware Characteristics and Limitations

2.1.1. Special Equipment and Device Interfaces

Interfaces to access either CD-ROM volumes or electronic file transfer are described elsewhere; for example, see TBD.

2.1.2. Special Setup Requirements

None.

2.2. Interface Medium Characteristics

2.3. Failure Protection, Detection, and Recovery

Raw instrument telemetry will be archived by JPL on CD-ROM. These archives and depacketized compressed image data will be archived at the MOC MOF.

2.4. End-of-File Conventions

End-of-file labeling shall comply with SFDU standards; specifically, fixed-size records are used, the header explicitly contains the record offset of each subelement of the dataset, and the size of each subelement can be computed from information in the header.

3. Access

3.1. Access Tools

Included on each CD-ROM volume will be a tool (derived from the MOC Ground Data System *readmsdp* program) that decompresses this format into a standard PDS-format image. The tool will be provided in source code form and as executables for several platforms.

3.2. Input/Output Protocols

None identified.

3.3. Timing and Sequencing Characteristics

None.

4. Detailed Interface Specifications

4.1. Labeling and Identification

The dataset ID is MGS-M-MOC-2-WASDP-L0-V1.0 for WA products and MGS-M-MOC-2-NASDP-L0-V1.0 for NA products.

Each product will have a file name of the form "*id*.IMQ", where the ID is not to exceed 8 characters, will start with an alphabetic character, and will consist only of alphanumeric characters. The file name will be unique across all MOC data product files. For mapping-phase images, the ID will be of the form PPPNNNNNN, where PPP is a mission phase descriptor and NNNNN is the image index within that mission phase. Case is not significant; under the Unix operating system, the names will be considered to be in all lower-case.

4.2. Structure and Organization Overview

All MOC images must be a multiple of 16 pixels in both width and height. Images are broken up into subimages (also called fragments), and each fragment is transmitted separately. Raw and predictively compressed images are reconstructed by concatenating all of their image fragments and then processing; transform compressed images are processed a fragment at a time.

A MOC data product consists of one image. A header identifies various properties of the image and contains a file offset to the compressed data portion of the image. The compressed data are then appended to the end of the file.

```
PDS_VERSION_ID                   PDS3
FILE_NAME                        "filename"
RECORD_TYPE                      FIXED_LENGTH
RECORD_BYTES                     nnnn
FILE_RECORDS                     nn
LABEL_RECORDS                    nn
^IMAGE                           nn
SPACECRAFT_NAME                  MARS_GLOBAL_SURVEYOR
MISSION_PHASE_NAME               MAPPING
TARGET_NAME                      MARS
INSTRUMENT_ID                    MOC
PRODUCER_ID                      MGS_MOC_TEAM
DATA_SET_ID                      MGS-M-MOC-2-NA/WA-SDP-L0-V1.0
PRODUCT_CREATION_TIME            yyyy-mm-ddThh:mm:ss.fff
SOFTWARE_NAME                    "id-string"
UPLOAD_ID                        "version-id"
PRODUCT_ID                       "product-id"
START_TIME                       yyyy-mm-ddThh:mm:ss.fff
IMAGE_TIME                       yyyy-mm-ddThh:mm:ss.fff
SPACECRAFT_CLOCK_START_COUNT     "sclk-string"
SPACECRAFT_CLOCK_STOP_COUNT      "N/A"
FOCAL_PLANE_TEMPERATURE          ff.fff
GAIN_MODE_ID                     "gain-id"
OFFSET_MODE_ID                   "offset-id"
LINE_EXPOSURE_DURATION           ff.ffffff
DOWNTRACK_SUMMING                nn
CROSSTRACK_SUMMING               nn
EDIT_MODE_ID                     "nnnn"
FILTER_NAME                      RED or BLUE
LINE_EXPOSURE_DURATION           ff.fff
RATIONALE_DESC                   string
DATA_QUALITY_DESC                "OK" or "ERROR"
ORBIT_NUMBER                     nnnnn
OBJECT                           IMAGE
ENCODING_TYPE                    "moc-compression-type"
LINES                            nnn
LINE_SAMPLES                     0
LINE_SUFFIX_BYTES                0
SAMPLE_TYPE                      UNSIGNED_INTEGER
SAMPLE_BITS                      8
SAMPLE_BIT_MASK                  2#11111111#
CHECKSUM                         16#xxxx#
END_OBJECT
END
```

4.3. Substructure Definition and Format

PDS_VERSION_ID
The PDS version number for the header format; e.g., PDS3.

FILE_NAME
The file name for these products; see above.

RECORD_TYPE
The record type; always FIXED_LENGTH for these products.

RECORD_BYTES
The number of bytes per record. For these products, 2048.

FILE_RECORDS
The total number of records in this file. The last record will be padded with zeros if necessary.

LABEL_RECORDS
The number of records used for header data. If needed, the last record of the header will be padded with blanks.

ˆIMAGE
A pointer to the starting record of the compressed image file.

SPACECRAFT_NAME
Always MARS_GLOBAL_SURVEYOR.

MISSION_PHASE_NAME
Name of the mission phase; e.g., MAPPING.

TARGET_NAME
The name of the target body; typically MARS.

PRODUCER_ID
Always MGS_MOC_TEAM.

DATA_SET_ID
MGS-M-MOC-2-WASDP-L0-V1.0 for WA products and MGS-M-MOC-2-NASDP-L0-V1.0 for NA products.

PRODUCT_CREATION_TIME
Time and date of this file's creation. Note that this time is the time of this file's creation in this format, and does not reflect the acquisition time or the time of any other processing that may be associated with this product.

SOFTWARE_NAME
Identifier of the version of the MOC Ground Data System software that created this product.

UPLOAD_ID
Identifier of the command file used to acquire this image.

PRODUCT_ID
(This field replaces the earlier IMAGE_ID field.) This uniquely identifies this MOC product among all MOC products. The MOC product ID format is CCCC/NNNNN, where CCCC is a string descriping the mission subphase and NNNNN is image number in that subphase; e.g., "FHGA/00013".

START_TIME, IMAGE_TIME
SCET (UTC) time at start of image acquisition, as commanded. These two fields are always the same. (IMAGE_TIME is included for compatibility with earlier non-MOC products.)

SPACECRAFT_CLOCK_START_COUNT
Value of spacecraft clock at the actual start of image acquisition. There may be small inconsistencies with START_TIME due to varying correlation between UTC and the spacecraft clock. For purposes of data analysis the spacecraft clock value should be used. The format of this field is compatible with

the NAIF Toolkit software (e.g., "00610499:32") The corresponding STOP_COUNT is not applicable because the timing of a MOC image, once started, is independent of the spacecraft clock.

The following information can be used, along with calibration files to be included on the volume, to calibrate each image. This information is in some sense redundant with that in the E-kernel.

FOCAL_PLANE_TEMPERATURE
: Temperature of focal plane of optical system associated with this image, in degrees Kelvin, at the start of image acquisition.

GAIN_MODE_ID
: The MOC gain setting in hexadecimal.

OFFSET_MODE_ID
: The MOC offset in integer steps of 5 DN.

LINE_EXPOSURE_DURATION
: Per-line exposure duration in units of milliseconds. The time a given line was acquired can be determined by multiplying the line exposure time by the number of previous lines and adding it to the image start time. Note that the NA implements downtrack summing by increasing the line time; for example, a 2X2 summed image has an actual line time twice that given by this field.

DOWNTRACK_SUMMING, CROSSTRACK_SUMMING
: The MOC can do pixel averaging in the instrument before transmission. For the NA, this must range from 1 (no summing) to 8x summing, and downtrack and crosstrack summing must be equal. For the WA, downtrack and crosstrack summing range from 1 to 127, and can be different.

EDIT_MODE_ID
: The edit mode is the first pixel of the CCD sampled for the image acquisition, and thus specifies the off-nadir look angle. For WA products, the special value 3456 indicates that the leading dark reference pixels were acquired as the first eight pixels of each line; the special value 3472 indicates that the trailing dark reference pixels were acquired as the last eight pixels of each line. For WA products, if dark pixels were acquired and compression was enabled, the dark reference pixels are compressed and included in the data. An EDIT_MODE_ID value of "0" refers to the first pixel in the array.

FILTER_NAME
: Either RED for the red Wide Angle or BLUE for the blue Wide Angle. Does not appear for NA products.

RATIONALE_DESC
: A text description of the scientific purpose for the acquisition of this image; e.g., "Monthly monitoring of aeolian features on summit of Pavonis Mons"

For some specific images, this string will contain a description of the image as actually received; for routine mapping operations, it will more likely be the goal of the image as targeted (which may not be met if the image missed its target significantly, the atmosphere was cloudy, gain parameters were set inappropriately, etc.)

DATA_QUALITY_DESC
: This field will be set to "OK" if all fragments of the image are received without detected checksum or sequence errors, and "ERROR" otherwise.

ORBIT_NUMBER
: The orbit number from the start of the mapping phase as defined by the MGS Project.

The following describe keywords found internal to the IMAGE object.

ENCODING_TYPE
: one of "NONE" for raw images, "MOC-PRED-*direction-table*" for predictive compression, "MOC-DCT-*requant*" for DCT compression, or "MOC-WHT-*requant*" for WHT compression.

LINES
: Number of lines in the decompressed image.

LINE_SAMPLES
: Number of samples per line in the decompressed image. (Each image in the file must have the same number of samples.)

LINE_PREFIX_BYTES
: Number of bytes of prefix information per line. This field is always 0 for MOC products.

LINE_SUFFIX_BYTES
: Number of bytes of suffix information per line. This field is always 0 for MOC products.

SAMPLE_TYPE
: Type of each sample; for MOC, always UNSIGNED_INTEGER.

SAMPLE_BITS
: Number of bits for each sample; for MOC, always 8.

SAMPLE_BIT_MASK
: Bit mask description for each sample; for MOC, always 2#11111111#.

CHECKSUM
: This is a checksum for the entire data part of this image, to be used for data validation. Because most MOC compression is lossy, there is not a unique decompressed image, so there is no way to provide a checksum for the decompressed data.

4.3.1. Header/Trailer Description Details

See above. No trailers are present.

4.3.2. Data Description Details

4.3.2.1. Geometry

Note that MOC images are acquired and compressed in row-major order by increasing time. The arrangement of CCDs and optics in the MOC somewhat complicates the mapping of pixel to surface feature. Suppose an image acquired while the spacecraft was moving south to north were displayed in left-to-right, top-to-bottom order on a monitor. For MOC A, the red WA image would have east at the left, and the NA and blue WA would have west at the left. The situation for the MOC B half-system is complex. The MOC B NA would have east at the left, because its CCD is flipped relative to MOC A's. The optical arrangements of the WAs are obviously still the same, but the wide angles are interchanged in a wiring sense on MOC B. However, the flight software compensates for this such that the WA images are the same from both systems.

The following table shows the compass direction on the planet that appears on the left side of an image as defined above.

	NA	WA red	WA blue
MOCA	west	east	west
MOCB	east	east	west

It is suggested that ancillary products be used to systematically display images in north-up, west-left form. The decompression tool does not perform this transformation.

4.3.2.2. Internal header

The compressed image portion of the file consists of the concatenated MOC Science Data Protocol (MSDP) "fragments" received from the instrument (see the MOC Flight Software User's Guide for details.) Each fragment begins with a 62-byte header and ends with a 1-byte checksum, according to the following format:

```
Offset  Length  Name    Definition
```

(Octet)	(Octet)		
0	2	SDID	The ID number of the entire image.
2	2	SDNUM	The subimage number of this datagram.
4	2	SDOFF	The offset downtrack of this datagram.
6	2	SDLINE	The length downtrack of this datagram.
8	5	SDTIME	The timestamp of the start of the entire image.
13	1	SDSTAT	Some of this datagram’s status.
14	17	SDCMD	The command that caused the entire image.
31	5	SDCTXT	The context image parameters.
36	2	SDGO	The camera gain and offset at the start of the entire image.
38	2	EMPTY	
40	2	SDDOWN	The number of lines downtrack in the entire image.
42	2	SDEDIT	The crosstrack editing performed.
44	8	SDCOMP	The compression table entry used for the entire image.
52	2	SDSENS	The sensor values associate with the entire image.
54	4	SDOTHER	The clocking rate of the camera CCD and dark reference pixel flag at the start of the entire image.
58	4	SDLEN	The number of octets in SDDAT part of this datagram.
62	SDLEN	SDDAT	The data portion of this datagram.
62+SDLEN	1	SDCS	The checksum redundancy of this datagram.

Note that all integer values appear in "little-endian" (i.e., least significant byte lower in memory) order.

For the purposes of decompressing the data, only the SDLEN and SDCOMP fields are used. See the source code for the decompression tool for details. Other fields are redundant with the labels of the file; the entire header is stored in this product only for simplicity.

4.3.2.3. Raw data

A raw MOC image is broken up into fragments containing 245760 (240K) bytes of image data (except for the last fragment in an image.) An individual fragment need not contain an integral number of lines of data. The entire image can be reconstructed by concatenating the data segments of all fragments.

Dark reference pixels can only be acquired for raw images. If they were acquired for the NA, they appear as the first four pixels on each line. For the WA, they appear as either the first eight or last eight pixels on each line.

4.3.2.4. Transform compression

A transform-compressed MOC image is broken up into a collection of 16x16-pixel regions called transform blocks, which are ordered in column-major order (top-to-bottom, then left-to-right.)

Each image is broken up into fragments such that each image fragment is a multiple of 16 lines in size and fits, decompressed, into no more than 240 Kbytes. Each compressed fragment is transmitted separately.

As transmitted, the transform block consists of the 256 Discrete Cosine Transform (DCT) or Walsh-Hadamard Transform (WHT) coefficients resulting from application of the 2D DCT or WHT to the original input pixel values. With the exception of the DC term, these coefficients are requantized (by division by a constant factor) and those coefficients sufficiently close to zero are "truncated" (omitted), starting with the high-frequency coefficients. Each coefficient position has one of 8 fixed Huffman encoding schemes assigned to it, and the coefficients are transmitted in encoded form. Truncation and transmission are done based on "radial" ordering; see Appendix A for the table mapping 1-D radial order to 2-D frequency order.

Note that each transform block is assigned to a group. Different groups are determined to be sufficiently different to have different encoding statistics. The number of groups for a given compressed fragment is set by ground command and must be in the range [1,8]. A group may be empty (that is, have no transform blocks in it.)

The DC coefficient is optimally requantized into 8 bits and is transmitted in unencoded form.

The coefficients, once decoded, are represented as 16-bit signed integers, with the exception of the DC coefficient, which should be treated as 16-bit unsigned.

The Huffman encoding schemes encode 16-bit signed values into variable-length bit strings with a maximum length of 24 bits. Coefficients that exceed the range of the encoding scheme are encoded with a distinguished "too negative" or "too positive" code followed by the 15 bits of the coefficient (since the sign is implied by the escape code, it is not output.) The coefficient encoding schemes are given in Appendix B.

The data part of the fragment consists of

```
for each block
   3 bits indicating the group the block was placed into.

for each group with non-zero occupancy
   16 bits minimum DC coefficient in group
   16 bits maximum DC coefficient in group

   for each coefficient in a transform block
         3 bits specifying the Huffman encoding to be used
         (from the 8 available)

   for each block in this group
         8 bits DC coefficient
         (using max and min DC, can be reconstructed into 16-bit form)

         8 bits number of zero-truncated coefficients
         for each untruncated coefficient (in radial order)
            1 to 24 bits Huffman encoded coefficient, or
            escape code and literal coefficient
```

After decoding, each AC coefficient must be multiplied by the requantization factor for this fragment, the block is reordered, and then the whole block is run through the inverse DCT or WHT to create the final 8-bit output block. A generic inverse DCT algorithm is given in Appendix C; we do not expect to use the WHT frequently during MOC operations, but it is included for completeness.

The final byte is padded with "0" bits.

4.3.2.5. Predictive compression

A predictively-compressed MOC image is broken up into fragments containing 240 Kbytes of compressed data (except for the last fragment in an image.) An individual fragment need not contain an integral number of lines of data. (The image width will be referred to as W.)

The data consists of two kinds of lines: compressed lines and sync lines. Sync lines are output every 128 lines and the first line output is a sync line. A sync line consists of

> 0 to 15 bits of "0" bits to pad the data stream to a word boundary
>
> 16 bits of sync code (two bytes, 0xca 0xf0)
>
> W 8-bit requantized image pixels

Errors in the downlink caused by dropped packets or bit errors can be compensated for by locating the next sync line (by searching forward for the 16-bit sync pattern) and restarting the decompression process. Otherwise, a single error could potentially ruin the rest of the image.

A compressed line consists of

> W 1- to 15-bit Huffman-coded difference values

Note that lines, either sync or compressed, can cross fragment boundaries.

Difference values are calculated as follows:

X: delta = cur - left
Y: delta = cur - up
XY: delta = cur + diag - up - left

where the pixels are oriented thus:

diag	up
left	cur

Though difference values are encoded statistically as signed 8-bit quantities with values from -128 to +127, they should be treated as unsigned 8-bit numbers in all differencing calculations. All calculations are performed modulo-256. This allows lossless encoding of all 8-bit input images.

Higher-rate, lossy predictive compression is supported by requantization of difference values. This requantization is performed by table lookup. Before a difference value is encoded, it is replaced by the value contained in the difference value's slot in the requantization table. Subsequently, the compressor behaves as though the difference value was the new, requantized value. (Note, however, that the encoding table lookup is done with the difference value prior to requantization. This means that the encoding table must have 256 entries even when requantization restricts the number of entries to less than 256.)

The fixed Huffman encoding tables and the requantization tables are given in Appendix D. An algorithm to convert the tables to tree form is given in Appendix E.

4.3.2.6. Global map swaths

Although files containing global map swaths have a single field for CROSSTRACK_SUMMING, variable summing is actually applied within the MOC to maintain approximately equal spatial resolution from nadir to limb. There are two defined modes for the global map: 7.5 km/pixel nominal (CROSSTRACK_SUMMING = 27) and 3.75 km/pixel (CROSSTRACK_SUMMING = 13). The variable crosstrack-summing tables used for these two modes are given in appendix F.

4.3.3. Data loss considerations

During the MGS mission to date, error-free transmission of the instrument data to Earth has not been provided. The MOC protocols (in particular, the formats for compressed image data, which are partially implemented in hardware) were designed for the bit error rates stated during mission planning and development. These rates were based on the link margin and Reed-Solomon encoding of the data, and were very low except during periods of equipment malfunction or poor weather at the DSN stations. In practice, many unanticipated error sources, most in the Earth segment of the communications link or caused by nonrandom operational sources, have significantly degraded the quality. As a result, considerable data losses were incurred in the image data. The majority of effort in archiving the pre-mapping data was expended to minimize the effects of this data loss.

MOC image data are broken up on transmission into 'packets' of approximately 1000 bytes. A typical data loss is that of one or two packets, due to uncorrectable bit errors caused by noise in the space-to-Earth communications path (rare), momentary loss of receiver lock caused by a transition between the one-way and two-way tracking modes, or loss in the Earth segment of the Deep Space Network.

For uncompressed images, a packet loss leads to loss of 'line sync' in the image. Since the amount of actual image data in a packet is variable and cannot be determined precisely without the packet, such errors must be corrected by hand. The majority of NA images were acquired using the lossless predictive compression mode of the MOC. However, when a packet is lost from this compressed data stream, the decompression algorithm cannot realign itself to the compressed pixel boundaries, and must skip ahead to the next sync marker, which occurs only every 128 lines in the image. The effect of decompressing the data between the site of packet loss and the next sync marker is unpredictable, but usually results in either semi-random variations in pixel brightness (with the general morphology of the original image still visible) or essentially random noise patterns.

A second type of loss is that of tens or hundreds of packets caused by bad weather, hardware failure, or operator error at the DSN stations, or miscommanding of the telemetry playback on the spacecraft. For these errors in a compressed data stream, over 128 lines of the image were lost, making it impossible to recover even the original downtrack size of the image.

The MOC ground software that produces the archival data, and the decompression tool provided, may perform some limited correction of these errors. Correct and complete reconstruction should only be expected if there are no detected checksum errors or sequence gaps in the data; i.e., if the DATA_QUALITY_DESC field is "OK".

4.4. Volume, Size, and Frequency Estimates

The total volume of MOC data to be returned was planned at the start of the mission to be approximately 45 GBytes. The extension of the mission due to the solar panel and high-gain antenna problems has affected this volume in a yet-to-be-determined manner. Volume returned varies as a function of the available data rate; see the Archive Policy and Data Management Plan for more details.

It is not known what fraction of total MOC return will be used for products of a particular type (e.g., NA or WA). It is also not known what average amounts of compression will be used, although a nominal value of 5:1 was expected prior to operations. In practice, to date the 2:1 lossless mode has been used for the majority of NA images.

5. Appendix A: radial order translation table

(IPSLib/reorder.static.h from the flight software.)

```
static uint8 trans[256] = {
  0,  1,  4,  9, 15, 22, 33, 43, 56, 71, 86,104,121,142,166,189,
  2,  3,  6, 11, 17, 26, 35, 45, 58, 73, 90,106,123,146,168,193,
  5,  7,  8, 13, 20, 28, 37, 50, 62, 75, 92,108,129,150,170,195,
 10, 12, 14, 19, 23, 31, 41, 52, 65, 81, 96,113,133,152,175,201,
 16, 18, 21, 24, 30, 39, 48, 59, 69, 83,100,119,137,158,181,203,
 25, 27, 29, 32, 40, 46, 54, 67, 79, 94,109,127,143,164,185,210,
 34, 36, 38, 42, 49, 55, 64, 76, 87,102,117,135,154,176,197,216,
 44, 47, 51, 53, 60, 68, 77, 85, 98,114,131,147,162,183,208,222,
 57, 61, 63, 66, 70, 80, 88, 99,112,124,140,159,179,199,214,227,
 72, 74, 78, 82, 84, 95,103,115,125,139,156,173,190,211,224,233,
 89, 91, 93, 97,101,110,118,132,141,157,171,186,206,220,231,239,
105,107,111,116,120,128,136,148,160,174,187,205,218,229,237,244,
122,126,130,134,138,144,155,163,180,191,207,219,226,235,242,248,
145,149,151,153,161,165,177,184,200,212,221,230,236,241,246,251,
167,169,172,178,182,188,198,209,215,225,232,238,243,247,250,253,
192,194,196,202,204,213,217,223,228,234,240,245,249,252,254,255,
};
```

6. Appendix B: transform coefficient Huffman code tables

(IPSLib/encodeCoefs.static.h from the flight software.)

```
/* Number of valid bits (LSBs) in each entry in "code0" */
static uint8 num0[25] = {
      24,      23,      20,      19,      16,      14,      13,      10,
       8,       6,       5,       3,       1,       2,       4,       7,
       9,      11,      12,      15,      17,      18,      21,      22,
      24,
};
```

```
/* Huffman code for encoding scheme 0, zero's code is index 12 */
static uint32 code0[25] = {
0xffffff,0x3fffff,0x07ffff,0x03ffff,0x007fff,0x001fff,0x000fff,0x0001ff,
0x00007f,0x00001f,0x00000f,0x000003,0x000000,0x000001,0x000007,0x00003f,
0x0000ff,0x0003ff,0x0007ff,0x003fff,0x00ffff,0x01ffff,0x0fffff,0x1fffff,
0x7fffff,
};

/* Number of valid bits (LSBs) in each entry in "code1" */
static uint8 num1[47] = {
      24,      24,      23,      22,      21,      20,      19,      18,
      17,      16,      15,      14,      13,      12,      11,      10,
       9,       8,       7,       6,       5,       4,       2,       2,
       2,       4,       5,       6,       7,       8,       9,      10,
      11,      12,      13,      14,      15,      16,      17,      18,
      19,      20,      21,      22,      23,      24,      24,
};

/* Huffman code for encoding scheme 1, zero's code is index 23 */
static uint32 code1[47] = {
0xffffff,0xbfffff,0x5fffff,0x2fffff,0x17ffff,0x0bffff,0x05ffff,0x02ffff,
0x017fff,0x00bfff,0x005fff,0x002fff,0x0017ff,0x000bff,0x0005ff,0x0002ff,
0x00017f,0x0000bf,0x00005f,0x00002f,0x000017,0x00000b,0x000002,0x000001,
0x000000,0x000003,0x000007,0x00000f,0x00001f,0x00003f,0x00007f,0x0000ff,
0x0001ff,0x0003ff,0x0007ff,0x000fff,0x001fff,0x003fff,0x007fff,0x00ffff,
0x01ffff,0x03ffff,0x07ffff,0x0fffff,0x1fffff,0x3fffff,0x7fffff,
};

/* Number of valid bits (LSBs) in each entry in "code2" */
static uint8 num2[69] = {
      24,      24,      23,      23,      22,      22,      21,      20,
      19,      19,      18,      17,      17,      16,      16,      15,
      14,      14,      13,      12,      11,      11,      10,       9,
       9,       8,       7,       7,       6,       6,       5,       4,
       4,       3,       2,       3,       3,       4,       5,       5,
       6,       7,       8,       8,       9,      10,      10,      11,
      12,      12,      13,      13,      14,      15,      15,      16,
      17,      18,      18,      19,      20,      20,      21,      21,
      22,      23,      23,      24,      24,
};

/* Huffman code for encoding scheme 2, zero's code is index 34 */
static uint32 code2[69] = {
0xffffff,0xfffffd,0x7ffffe,0x3ffffd,0x1ffffe,0x1fffff,0x0fffff,0x07fffe,
0x03fffe,0x03ffff,0x01fffd,0x00fffe,0x00ffff,0x007ffe,0x007fff,0x003fff,
0x001ffe,0x001fff,0x000fff,0x0007fe,0x0003fe,0x0003ff,0x0001fe,0x0000fe,
0x0000ff,0x00007d,0x00003e,0x00003d,0x00001d,0x00001f,0x00000d,0x000005,
0x000007,0x000001,0x000000,0x000003,0x000002,0x000006,0x00000f,0x00000e,
0x00001e,0x00003f,0x00007f,0x00007e,0x0000fd,0x0001ff,0x0001fd,0x0003fd,
0x0007ff,0x0007fd,0x000ffd,0x000ffe,0x001ffd,0x003ffd,0x003ffe,0x007ffd,
0x00fffd,0x01ffff,0x01fffe,0x03fffd,0x07ffff,0x07fffd,0x0ffffd,0x0ffffe,
0x1ffffd,0x3fffff,0x3ffffe,0x7ffffd,0x7fffff,
};

/* Number of valid bits (LSBs) in each entry in "code3" */
static uint8 num3[109] = {
      23,      24,      24,      23,      23,      22,      22,      22,
      21,      21,      21,      20,      20,      19,      19,      18,
      18,      18,      17,      17,      16,      16,      16,      15,
      15,      14,      14,      14,      13,      13,      13,      12,
      12,      11,      11,      10,      10,      10,       9,       9,
       9,       8,       8,       7,       7,       6,       6,       6,
       5,       5,       4,       4,       4,       3,       3,       3,
       4,       4,       5,       5,       5,       6,       6,       7,
       7,       7,       8,       8,       8,       9,       9,      10,
      10,      11,      11,      11,      12,      12,      12,      13,
      13,      14,      14,      15,      15,      15,      16,      16,
      17,      17,      17,      18,      18,      19,      19,      19,
      20,      20,      20,      21,      21,      22,      22,      22,
      23,      23,      24,      24,      23,
};

/* Huffman code for encoding scheme 3, zero's code is index 54 */
static uint32 code3[109] = {
0x7fffff,0xfffffd,0xdfffff,0x7ffffe,0x3ffffd,0x3ffffc,0x1ffffe,0x3ffffb,
0x0ffffe,0x0ffffd,0x0ffffb,0x07fffd,0x07ffff,0x03fffc,0x03ffff,0x01fffc,
0x01fffe,0x01ffff,0x00fffe,0x00fffd,0x007ffc,0x007ffd,0x007ffb,0x003ffc,
0x003fff,0x001ffc,0x001ffd,0x001fff,0x000ffc,0x000ffd,0x000ffb,0x0007fd,
0x0007fb,0x0003fe,0x0003fd,0x0001fc,0x0001fd,0x0001ff,0x0000fc,0x0000ff,
0x0000fb,0x00007d,0x00007b,0x00003c,0x00003d,0x00001c,0x00001e,0x00001b,
0x00000e,0x00000f,0x000004,0x000006,0x000003,0x000002,0x000001,0x000000,
0x000007,0x000005,0x00000b,0x00000d,0x00000c,0x00001f,0x00001d,0x00003b,
0x00003f,0x00003e,0x00007f,0x00007e,0x00007c,0x0000fd,0x0000fe,0x0001fb,
0x0001fe,0x0003fb,0x0003ff,0x0003fc,0x0007ff,0x0007fe,0x0007fc,0x000fff,
0x000ffe,0x001ffb,0x001ffe,0x003ffb,0x003ffd,0x003ffe,0x007fff,0x007ffe,
0x00fffb,0x00ffff,0x00fffc,0x01fffb,0x01fffd,0x03fffb,0x03fffd,0x03fffe,
0x07fffb,0x07fffe,0x07fffc,0x0fffff,0x0ffffc,0x1ffffb,0x1ffffd,0x1ffffc,
0x1fffff,0x3ffffe,0x5fffff,0x7ffffd,0x3fffff,
```

```
};

/* Number of valid bits (LSBs) in each entry in "code4" */
static uint8 num4[169] = {
      22,      24,      24,      24,      24,      23,      23,      23,
      23,      22,      22,      22,      22,      21,      21,      21,
      21,      20,      20,      20,      20,      19,      19,      19,
      19,      18,      18,      18,      18,      17,      17,      17,
      17,      16,      16,      16,      16,      15,      15,      15,
      15,      14,      14,      14,      14,      13,      13,      13,
      13,      12,      12,      12,      12,      11,      11,      11,
      11,      10,      10,      10,      10,       9,       9,       9,
       9,       8,       8,       8,       8,       7,       7,       7,
       7,       6,       6,       6,       6,       5,       5,       5,
       5,       4,       4,       4,       3,       4,       4,       4,
       5,       5,       5,       5,       6,       6,       6,       6,
       7,       7,       7,       7,       8,       8,       8,       8,
       9,       9,       9,       9,      10,      10,      10,      10,
      11,      11,      11,      11,      12,      12,      12,      12,
      13,      13,      13,      13,      14,      14,      14,      14,
      15,      15,      15,      15,      16,      16,      16,      16,
      17,      17,      17,      17,      18,      18,      18,      18,
      19,      19,      19,      19,      20,      20,      20,      20,
      21,      21,      21,      21,      22,      22,      22,      22,
      23,      23,      23,      23,      24,      24,      24,      24,
      22,
};

/* Huffman code for encoding scheme 4, zero’s code is index 84 */
static uint32 code4[169] = {
0x3fffff,0xf7ffff,0xe7ffff,0xfdffff,0xfffffe,0x27ffff,0x7bffff,0x3dffff,
0x5ffffe,0x17ffff,0x1bffff,0x3ffffc,0x2ffffe,0x0bffff,0x0dffff,0x17fffc,
0x17fffe,0x03ffff,0x0ffffd,0x0bfffc,0x0bfffe,0x03fffd,0x05fffd,0x05fffc,
0x05fffe,0x02ffff,0x02fffd,0x02fffc,0x02fffe,0x017fff,0x017ffd,0x017ffc,
0x017ffe,0x00bfff,0x00bffd,0x00bffc,0x00bffe,0x005fff,0x005ffd,0x005ffc,
0x005ffe,0x002fff,0x002ffd,0x002ffc,0x002ffe,0x0017ff,0x0017fd,0x0017fc,
0x0017fe,0x000bff,0x000bfd,0x000bfc,0x000bfe,0x0005ff,0x0005fd,0x0005fc,
0x0005fe,0x0002ff,0x0002fd,0x0002fc,0x0002fe,0x00017f,0x00017d,0x00017c,
0x00017e,0x0000bf,0x0000bd,0x0000bc,0x0000be,0x00005f,0x00005d,0x00005c,
0x00005e,0x00002f,0x00002d,0x00002c,0x00002e,0x000017,0x000015,0x000014,
0x000016,0x000009,0x000008,0x00000a,0x000003,0x000002,0x000000,0x000001,
0x000006,0x000004,0x000005,0x000007,0x00000e,0x00000c,0x00000d,0x00000f,
0x00001e,0x00001c,0x00001d,0x00001f,0x00003e,0x00003c,0x00003d,0x00003f,
0x00007e,0x00007c,0x00007d,0x00007f,0x0000fe,0x0000fc,0x0000fd,0x0000ff,
0x0001fe,0x0001fc,0x0001fd,0x0001ff,0x0003fe,0x0003fc,0x0003fd,0x0003ff,
0x0007fe,0x0007fc,0x0007fd,0x0007ff,0x000ffe,0x000ffc,0x000ffd,0x000fff,
0x001ffe,0x001ffc,0x001ffd,0x001fff,0x003ffe,0x003ffc,0x003ffd,0x003fff,
0x007ffe,0x007ffc,0x007ffd,0x007fff,0x00fffe,0x00fffc,0x00fffd,0x00ffff,
0x01fffe,0x01fffc,0x01fffd,0x01ffff,0x03fffe,0x03fffc,0x07fffd,0x05ffff,
0x07fffe,0x07fffc,0x0ffffc,0x0fffff,0x0ffffe,0x1ffffc,0x1dffff,0x07ffff,
0x1ffffe,0x3ffffe,0x3bffff,0x37ffff,0x7ffffe,0x7dffff,0x67ffff,0x77ffff,
0x1fffff,
};

/* Number of valid bits (LSBs) in each entry in "code5" */
static uint8 num5[247] = {
      21,      24,      24,      24,      24,      24,      24,      23,
      23,      23,      23,      23,      23,      22,      22,      22,
      22,      22,      22,      22,      21,      21,      21,      21,
      21,      20,      20,      20,      20,      20,      20,      19,
      19,      19,      19,      19,      19,      18,      18,      18,
      18,      18,      18,      17,      17,      17,      17,      17,
      17,      16,      16,      16,      16,      16,      16,      15,
      15,      15,      15,      15,      15,      14,      14,      14,
      14,      14,      14,      13,      13,      13,      13,      13,
      13,      12,      12,      12,      12,      12,      12,      11,
      11,      11,      11,      11,      11,      10,      10,      10,
      10,      10,      10,       9,       9,       9,       9,       9,
       9,       8,       8,       8,       8,       8,       8,       7,
       7,       7,       7,       7,       7,       6,       6,       6,
       6,       6,       6,       5,       5,       5,       5,       5,
       5,       5,       4,       4,       4,       5,       5,       5,
       5,       5,       5,       5,       6,       6,       6,       6,
       6,       6,       7,       7,       7,       7,       7,       7,
       8,       8,       8,       8,       8,       8,       9,       9,
       9,       9,       9,       9,      10,      10,      10,      10,
      10,      10,      11,      11,      11,      11,      11,      11,
      12,      12,      12,      12,      12,      12,      13,      13,
      13,      13,      13,      13,      14,      14,      14,      14,
      14,      14,      15,      15,      15,      15,      15,      15,
      16,      16,      16,      16,      16,      16,      17,      17,
      17,      17,      17,      17,      18,      18,      18,      18,
      18,      18,      19,      19,      19,      19,      19,      19,
      20,      20,      20,      20,      20,      20,      21,      21,
      21,      21,      21,      21,      22,      22,      22,      22,
      22,      22,      23,      23,      23,      23,      23,      23,
      24,      24,      24,      24,      24,      24,      21,
};
```

```
/* Huffman code for encoding scheme 5, zero’s code is index 123 */
static uint32 code5[247] = {
0x1fffff,0xfffffd,0xfffffe,0xfffffa,0xff7ffa,0xfffffc,0xfffff8,0x5ffffd,
0x3ffffd,0x7fbffe,0x3f7ffa,0x5ffffc,0x3ffffc,0x37ffff,0x3fdffd,0x1fbffe,
0x1f7ffa,0x2ffffc,0x2ffff8,0x1ffff8,0x0ffffd,0x0fbffe,0x0ffffa,0x17fffc,
0x17fff8,0x0bffff,0x07fffd,0x07bffe,0x077ffa,0x0bfffc,0x0bfff8,0x05ffff,
0x03fffd,0x03fffe,0x037ffa,0x05fffc,0x05fff8,0x02ffff,0x01dffd,0x01fffe,
0x017ffa,0x02fffc,0x02fff8,0x017fff,0x00fffd,0x00fffe,0x007ffa,0x017ffc,
0x017ff8,0x00bfff,0x005ffd,0x003ffe,0x00bffa,0x00bffc,0x00bff8,0x005fff,
0x001ffd,0x005ffe,0x005ffa,0x005ffc,0x005ff8,0x002fff,0x002ffd,0x002ffe,
0x002ffa,0x002ffc,0x002ff8,0x0017ff,0x0017fd,0x0017fe,0x0017fa,0x0017fc,
0x0017f8,0x000bff,0x000bfd,0x000bfe,0x000bfa,0x000bfc,0x000bf8,0x0005ff,
0x0005fd,0x0005fe,0x0005fa,0x0005fc,0x0005f8,0x0002ff,0x0002fd,0x0002fe,
0x0002fa,0x0002fc,0x0002f8,0x00017f,0x00017d,0x00017e,0x00017a,0x00017c,
0x000178,0x0000bf,0x0000bd,0x0000be,0x0000ba,0x0000bc,0x0000b8,0x00005f,
0x00005d,0x00005e,0x00005a,0x00005c,0x000058,0x00002f,0x00002d,0x00002e,
0x00002a,0x00002c,0x000028,0x000017,0x000015,0x000019,0x000016,0x000012,
0x000014,0x000010,0x00000b,0x000001,0x000003,0x000000,0x000004,0x000002,
0x000006,0x000009,0x000005,0x000007,0x000008,0x00000c,0x00000a,0x00000e,
0x00000d,0x00000f,0x000018,0x00001c,0x00001a,0x00001e,0x00001d,0x00001f,
0x000038,0x00003c,0x00003a,0x00003e,0x00003d,0x00003f,0x000078,0x00007c,
0x00007a,0x00007e,0x00007d,0x00007f,0x0000f8,0x0000fc,0x0000fa,0x0000fe,
0x0000fd,0x0000ff,0x0001f8,0x0001fc,0x0001fa,0x0001fe,0x0001fd,0x0001ff,
0x0003f8,0x0003fc,0x0003fa,0x0003fe,0x0003fd,0x0003ff,0x0007f8,0x0007fc,
0x0007fa,0x0007fe,0x0007fd,0x0007ff,0x000ff8,0x000ffc,0x000ffa,0x000ffe,
0x000ffd,0x000fff,0x001ff8,0x001ffc,0x001ffa,0x001ffe,0x003ffd,0x001fff,
0x003ff8,0x003ffc,0x003ffa,0x007ffe,0x007ffd,0x003fff,0x007ff8,0x007ffc,
0x00fffa,0x00bffe,0x00dffd,0x007fff,0x00fff8,0x00fffc,0x01fffa,0x01bffe,
0x01fffd,0x00ffff,0x01fff8,0x01fffc,0x03fffa,0x03bffe,0x03dffd,0x01ffff,
0x03fff8,0x03fffc,0x07fffa,0x07fffe,0x07dffd,0x03ffff,0x07fff8,0x07fffc,
0x0f7ffa,0x0ffffe,0x0fdffd,0x07ffff,0x0ffff8,0x0ffffc,0x1ffffa,0x1ffffe,
0x1fdffd,0x17ffff,0x3ffff8,0x1ffffc,0x3ffffa,0x3fbffe,0x3ffffe,0x1ffffd,
0x7ffff8,0x7ffffc,0x7f7ffa,0x7ffffa,0x7ffffe,0x7ffffd,0x0fffff,
};

/* Number of valid bits (LSBs) in each entry in "code6" */
static uint8 num6[395] = {
      21,      24,      24,      24,      24,      24,      24,      24,
      24,      24,      23,      23,      23,      23,      23,      23,
      23,      23,      23,      23,      22,      22,      22,      22,
      22,      22,      22,      22,      22,      22,      22,      21,
      21,      21,      21,      21,      21,      21,      21,      21,
      21,      20,      20,      20,      20,      20,      20,      20,
      20,      20,      20,      19,      19,      19,      19,      19,
      19,      19,      19,      19,      19,      18,      18,      18,
      18,      18,      18,      18,      18,      18,      18,      17,
      17,      17,      17,      17,      17,      17,      17,      17,
      17,      16,      16,      16,      16,      16,      16,      16,
      16,      16,      16,      15,      15,      15,      15,      15,
      15,      15,      15,      15,      15,      14,      14,      14,
      14,      14,      14,      14,      14,      14,      14,      13,
      13,      13,      13,      13,      13,      13,      13,      13,
      13,      12,      12,      12,      12,      12,      12,      12,
      12,      12,      12,      11,      11,      11,      11,      11,
      11,      11,      11,      11,      11,      10,      10,      10,
      10,      10,      10,      10,      10,      10,      10,       9,
       9,       9,       9,       9,       9,       9,       9,       9,
       9,       8,       8,       8,       8,       8,       8,       8,
       8,       8,       8,       7,       7,       7,       7,       7,
       7,       7,       7,       7,       7,       6,       6,       6,
       6,       6,       6,       6,       6,       6,       6,       6,
       5,       5,       5,       5,       5,       5,       5,       5,
       5,       5,       5,       6,       6,       6,       6,       6,
       6,       6,       6,       6,       6,       6,       7,       7,
       7,       7,       7,       7,       7,       7,       7,       7,
       8,       8,       8,       8,       8,       8,       8,       8,
       8,       8,       9,       9,       9,       9,       9,       9,
       9,       9,       9,       9,      10,      10,      10,      10,
      10,      10,      10,      10,      10,      10,      11,      11,
      11,      11,      11,      11,      11,      11,      11,      11,
      12,      12,      12,      12,      12,      12,      12,      12,
      12,      12,      13,      13,      13,      13,      13,      13,
      13,      13,      13,      13,      14,      14,      14,      14,
      14,      14,      14,      14,      14,      14,      15,      15,
      15,      15,      15,      15,      15,      15,      15,      15,
      16,      16,      16,      16,      16,      16,      16,      16,
      16,      16,      17,      17,      17,      17,      17,      17,
      17,      17,      17,      17,      18,      18,      18,      18,
      18,      18,      18,      18,      18,      18,      19,      19,
      19,      19,      19,      19,      19,      19,      19,      19,
      20,      20,      20,      20,      20,      20,      20,      20,
      20,      20,      21,      21,      21,      21,      21,      21,
      21,      21,      21,      21,      22,      22,      22,      22,
      22,      22,      22,      22,      22,      22,      23,      23,
      23,      23,      23,      23,      23,      23,      23,      23,
      23,      24,      24,      24,      24,      24,      24,      24,
      24,      24,      21,
};
```

```
/* Huffman code for encoding scheme 6, zero’s code is index 197 */
static uint32 code6[395] = {
0x1fffff,0xfffffe,0xfffffc,0xfffff8,0xffbff8,0xfffffd,0xfffff9,0xfffffb,
0xfffff3,0xfffff7,0x7ffffa,0x5ffffe,0x3ffffc,0x7fdffc,0x3ffff8,0x5ffffd,
0x3ffffd,0x6ffffb,0x3ffffb,0x5ffff7,0x1ffffa,0x3feffa,0x3fdffe,0x1fdffc,
0x1ffff8,0x2ffffd,0x2ffff9,0x0ffffb,0x1ffffb,0x37ffff,0x3fdff7,0x0feffa,
0x0fdffe,0x0fdffc,0x0ffff8,0x17fffd,0x17fff9,0x17fffb,0x17fff3,0x0ffff3,
0x0ffff7,0x07effa,0x07fffe,0x07fffc,0x07bff8,0x0bfffd,0x0bfff9,0x0bfffb,
0x0bfff3,0x0bffff,0x07dff7,0x03effa,0x03dffe,0x03fffc,0x03bff8,0x05fffd,
0x05fff9,0x05fffb,0x05fff3,0x05ffff,0x03fff7,0x01effa,0x01dffe,0x01fffc,
0x01bff8,0x02fffd,0x02fff9,0x02fffb,0x02fff3,0x02ffff,0x01dff7,0x00fffa,
0x00fffe,0x00dffc,0x00bff8,0x017ffd,0x017ff9,0x017ffb,0x017ff3,0x017fff,
0x00fff7,0x006ffa,0x007ffe,0x007ffc,0x003ff8,0x00bffd,0x00bff9,0x00bffb,
0x00bff3,0x00bfff,0x005ff7,0x003ffa,0x001ffe,0x003ffc,0x005ff8,0x005ffd,
0x005ff9,0x005ffb,0x005ff3,0x005fff,0x001ff7,0x001ffa,0x002ffe,0x002ffc,
0x002ff8,0x002ffd,0x002ff9,0x002ffb,0x002ff3,0x002fff,0x002ff7,0x0017fa,
0x0017fe,0x0017fc,0x0017f8,0x0017fd,0x0017f9,0x0017fb,0x0017f3,0x0017ff,
0x0017f7,0x000bfa,0x000bfe,0x000bfc,0x000bf8,0x000bfd,0x000bf9,0x000bfb,
0x000bf3,0x000bff,0x000bf7,0x0005fa,0x0005fe,0x0005fc,0x0005f8,0x0005fd,
0x0005f9,0x0005fb,0x0005f3,0x0005ff,0x0005f7,0x0002fa,0x0002fe,0x0002fc,
0x0002f8,0x0002fd,0x0002f9,0x0002fb,0x0002f3,0x0002ff,0x0002f7,0x00017a,
0x00017e,0x00017c,0x000178,0x00017d,0x000179,0x00017b,0x000173,0x00017f,
0x000177,0x0000ba,0x0000be,0x0000bc,0x0000b8,0x0000bd,0x0000b9,0x0000bb,
0x0000b3,0x0000bf,0x0000b7,0x00005a,0x00005e,0x00005c,0x000058,0x00005d,
0x000059,0x00005b,0x000053,0x00005f,0x000057,0x00002a,0x00002e,0x00002c,
0x000028,0x00002d,0x000029,0x000031,0x00002b,0x000023,0x00002f,0x000027,
0x000012,0x000016,0x000014,0x000010,0x000015,0x000001,0x000005,0x000000,
0x000004,0x000006,0x000002,0x000007,0x00000f,0x000003,0x00000b,0x000011,
0x000009,0x00000d,0x000008,0x00000c,0x00000e,0x00000a,0x000017,0x00001f,
0x000013,0x00001b,0x000019,0x00001d,0x000018,0x00001c,0x00001e,0x00001a,
0x000037,0x00003f,0x000033,0x00003b,0x000039,0x00003d,0x000038,0x00003c,
0x00003e,0x00003a,0x000077,0x00007f,0x000073,0x00007b,0x000079,0x00007d,
0x000078,0x00007c,0x00007e,0x00007a,0x0000f7,0x0000ff,0x0000f3,0x0000fb,
0x0000f9,0x0000fd,0x0000f8,0x0000fc,0x0000fe,0x0000fa,0x0001f7,0x0001ff,
0x0001f3,0x0001fb,0x0001f9,0x0001fd,0x0001f8,0x0001fc,0x0001fe,0x0001fa,
0x0003f7,0x0003ff,0x0003f3,0x0003fb,0x0003f9,0x0003fd,0x0003f8,0x0003fc,
0x0003fe,0x0003fa,0x0007f7,0x0007ff,0x0007f3,0x0007fb,0x0007f9,0x0007fd,
0x0007f8,0x0007fc,0x0007fe,0x0007fa,0x000ff7,0x000fff,0x000ff3,0x000ffb,
0x000ff9,0x000ffd,0x000ff8,0x000ffc,0x000ffe,0x000ffa,0x003ff7,0x001fff,
0x001ff3,0x001ffb,0x001ff9,0x001ffd,0x001ff8,0x001ffc,0x003ffe,0x002ffa,
0x007ff7,0x003fff,0x003ff3,0x003ffb,0x003ff9,0x003ffd,0x007ff8,0x005ffc,
0x005ffe,0x007ffa,0x00dff7,0x007fff,0x007ff3,0x007ffb,0x007ff9,0x007ffd,
0x00fff8,0x00fffc,0x00dffe,0x00effa,0x01fff7,0x00ffff,0x00fff3,0x00fffb,
0x00fff9,0x00fffd,0x01fff8,0x01dffc,0x01fffe,0x01fffa,0x03dff7,0x01ffff,
0x01fff3,0x01fffb,0x01fff9,0x01fffd,0x03fff8,0x03dffc,0x03fffe,0x03fffa,
0x07fff7,0x03ffff,0x03fff3,0x03fffb,0x03fff9,0x03fffd,0x07fff8,0x07dffc,
0x07dffe,0x07fffa,0x0fdff7,0x07ffff,0x07fff3,0x07fffb,0x07fff9,0x07fffd,
0x0fbff8,0x0ffffc,0x0ffffe,0x0ffffa,0x1fdff7,0x17ffff,0x1ffff3,0x1ffff9,
0x0ffff9,0x0ffffd,0x1fbff8,0x1ffffc,0x1fdffe,0x1feffa,0x3ffff7,0x1ffff7,
0x3ffff3,0x2ffffb,0x3ffff9,0x1ffffd,0x3fbff8,0x3fdffc,0x3ffffe,0x1ffffe,
0x3ffffa,0x7ffff7,0x7ffff3,0x7ffffb,0x7ffff9,0x7ffffd,0x7fbff8,0x7ffff8,
0x7ffffc,0x7ffffe,0x0fffff,
};

/* Number of valid bits (LSBs) in each entry in "code7" */
static uint8 num7[609] = {
      20,      24,      24,      24,      24,      24,      24,      24,
      24,      24,      24,      24,      24,      24,      24,      24,
      24,      23,      23,      23,      23,      23,      23,      23,
      23,      23,      23,      23,      23,      23,      23,      23,
      23,      22,      22,      22,      22,      22,      22,      22,
      22,      22,      22,      22,      22,      22,      22,      22,
      22,      21,      21,      21,      21,      21,      21,      21,
      21,      21,      21,      21,      21,      21,      21,      21,
      21,      20,      20,      20,      20,      20,      20,      20,
      20,      20,      20,      20,      20,      20,      20,      20,
      20,      19,      19,      19,      19,      19,      19,      19,
      19,      19,      19,      19,      19,      19,      19,      19,
      19,      18,      18,      18,      18,      18,      18,      18,
      18,      18,      18,      18,      18,      18,      18,      18,
      18,      17,      17,      17,      17,      17,      17,      17,
      17,      17,      17,      17,      17,      17,      17,      17,
      17,      16,      16,      16,      16,      16,      16,      16,
      16,      16,      16,      16,      16,      16,      16,      16,
      16,      15,      15,      15,      15,      15,      15,      15,
      15,      15,      15,      15,      15,      15,      15,      15,
      15,      14,      14,      14,      14,      14,      14,      14,
      14,      14,      14,      14,      14,      14,      14,      14,
      14,      13,      13,      13,      13,      13,      13,      13,
      13,      13,      13,      13,      13,      13,      13,      13,
      13,      12,      12,      12,      12,      12,      12,      12,
      12,      12,      12,      12,      12,      12,      12,      12,
      12,      11,      11,      11,      11,      11,      11,      11,
      11,      11,      11,      11,      11,      11,      11,      11,
      11,      10,      10,      10,      10,      10,      10,      10,
      10,      10,      10,      10,      10,      10,      10,      10,
      10,       9,       9,       9,       9,       9,       9,       9,
       9,       9,       9,       9,       9,       9,       9,       9,
```

```
      9,       8,       8,       8,       8,       8,       8,       8,
      8,       8,       8,       8,       8,       8,       8,       8,
      8,       7,       7,       7,       7,       7,       7,       7,
      7,       7,       7,       7,       7,       7,       7,       7,
      6,       6,       6,       6,       6,       6,       6,       6,
      6,       6,       6,       6,       6,       6,       6,       6,
      6,       6,       6,       6,       6,       6,       6,       6,
      6,       6,       6,       6,       6,       6,       6,       6,
      6,       7,       7,       7,       7,       7,       7,       7,
      7,       7,       7,       7,       7,       7,       7,       7,
      8,       8,       8,       8,       8,       8,       8,       8,
      8,       8,       8,       8,       8,       8,       8,       8,
      9,       9,       9,       9,       9,       9,       9,       9,
      9,       9,       9,       9,       9,       9,       9,       9,
     10,      10,      10,      10,      10,      10,      10,      10,
     10,      10,      10,      10,      10,      10,      10,      10,
     11,      11,      11,      11,      11,      11,      11,      11,
     11,      11,      11,      11,      11,      11,      11,      11,
     12,      12,      12,      12,      12,      12,      12,      12,
     12,      12,      12,      12,      12,      12,      12,      12,
     13,      13,      13,      13,      13,      13,      13,      13,
     13,      13,      13,      13,      13,      13,      13,      13,
     14,      14,      14,      14,      14,      14,      14,      14,
     14,      14,      14,      14,      14,      14,      14,      14,
     15,      15,      15,      15,      15,      15,      15,      15,
     15,      15,      15,      15,      15,      15,      15,      15,
     16,      16,      16,      16,      16,      16,      16,      16,
     16,      16,      16,      16,      16,      16,      16,      16,
     17,      17,      17,      17,      17,      17,      17,      17,
     17,      17,      17,      17,      17,      17,      17,      17,
     18,      18,      18,      18,      18,      18,      18,      18,
     18,      18,      18,      18,      18,      18,      18,      18,
     19,      19,      19,      19,      19,      19,      19,      19,
     19,      19,      19,      19,      19,      19,      19,      19,
     20,      20,      20,      20,      20,      20,      20,      20,
     20,      20,      20,      20,      20,      20,      20,      20,
     21,      21,      21,      21,      21,      21,      21,      21,
     21,      21,      21,      21,      21,      21,      21,      21,
     22,      22,      22,      22,      22,      22,      22,      22,
     22,      22,      22,      22,      22,      22,      22,      22,
     23,      23,      23,      23,      23,      23,      23,      23,
     23,      23,      23,      23,      23,      23,      23,      23,
     24,      24,      24,      24,      24,      24,      24,      24,
     24,      24,      24,      24,      24,      24,      24,      24,
     20,
};

/* Huffman code for encoding scheme 7, zero’s code is index 304 */
static uint32 code7[609] = {
0x0fffff,0xfffffd,0xeffffd,0xfffff5,0xfffff9,0xfffff1,0xfffffb,0xfffff3,
0xfffff7,0xfdffff,0xedffff,0xfffffe,0xfffffa,0xfffffc,0xf7fffc,0xfffff8,
0xfffff0,0x2ffffd,0x77fff5,0x3ffff9,0x5ffff9,0x3ffffb,0x7bfffb,0x3ffff7,
0x5ffff7,0x2dffff,0x7bfffe,0x3ffffe,0x5ffffa,0x3ffffc,0x7ffff4,0x3ffff0,
0x5ffff0,0x0ffffd,0x1ffff5,0x37fff9,0x3bfff1,0x1ffffb,0x1bfffb,0x2ffff3,
0x2ffff7,0x1dffff,0x1bfffe,0x3bfffa,0x3ffff2,0x1ffffc,0x1ffff8,0x2ffff8,
0x2ffff0,0x17fffd,0x0ffff5,0x0ffff9,0x0bfff1,0x0bfffb,0x1bfff3,0x1bfff7,
0x1bffff,0x1ffff6,0x0ffffe,0x0ffffa,0x0ffff2,0x0ffff4,0x17fff4,0x17fff8,
0x17fff0,0x0bfffd,0x0bfff5,0x0bfff9,0x07fff1,0x07fffb,0x07fff3,0x03fff7,
0x03ffff,0x03fffe,0x07fffe,0x07fffa,0x0bfff2,0x0bfffc,0x0bfff4,0x0bfff8,
0x0bfff0,0x05fffd,0x05fff5,0x05fff9,0x05fff1,0x05fffb,0x05fff3,0x05fff7,
0x01ffff,0x05fff6,0x05fffe,0x05fffa,0x05fff2,0x05fffc,0x05fff4,0x05fff8,
0x05fff0,0x02fffd,0x02fff5,0x02fff9,0x02fff1,0x02fffb,0x02fff3,0x02fff7,
0x02ffff,0x02fff6,0x02fffe,0x02fffa,0x02fff2,0x02fffc,0x02fff4,0x02fff8,
0x02fff0,0x017ffd,0x017ff5,0x017ff9,0x017ff1,0x017ffb,0x017ff3,0x017ff7,
0x017fff,0x017ff6,0x017ffe,0x017ffa,0x017ff2,0x017ffc,0x017ff4,0x017ff8,
0x017ff0,0x00bffd,0x00bff5,0x00bff9,0x00bff1,0x00bffb,0x00bff3,0x00bff7,
0x00bfff,0x00bff6,0x00bffe,0x00bffa,0x00bff2,0x00bffc,0x00bff4,0x00bff8,
0x00bff0,0x005ffd,0x005ff5,0x005ff9,0x005ff1,0x005ffb,0x005ff3,0x005ff7,
0x005fff,0x005ff6,0x005ffe,0x005ffa,0x005ff2,0x005ffc,0x005ff4,0x005ff8,
0x005ff0,0x002ffd,0x002ff5,0x002ff9,0x002ff1,0x002ffb,0x002ff3,0x002ff7,
0x002fff,0x002ff6,0x002ffe,0x002ffa,0x002ff2,0x002ffc,0x002ff4,0x002ff8,
0x002ff0,0x0017fd,0x0017f5,0x0017f9,0x0017f1,0x0017fb,0x0017f3,0x0017f7,
0x0017ff,0x0017f6,0x0017fe,0x0017fa,0x0017f2,0x0017fc,0x0017f4,0x0017f8,
0x0017f0,0x000bfd,0x000bf5,0x000bf9,0x000bf1,0x000bfb,0x000bf3,0x000bf7,
0x000bff,0x000bf6,0x000bfe,0x000bfa,0x000bf2,0x000bfc,0x000bf4,0x000bf8,
0x000bf0,0x0005fd,0x0005f5,0x0005f9,0x0005f1,0x0005fb,0x0005f3,0x0005f7,
0x0005ff,0x0005f6,0x0005fe,0x0005fa,0x0005f2,0x0005fc,0x0005f4,0x0005f8,
0x0005f0,0x0002fd,0x0002f5,0x0002f9,0x0002f1,0x0002fb,0x0002f3,0x0002f7,
0x0002ff,0x0002f6,0x0002fe,0x0002fa,0x0002f2,0x0002fc,0x0002f4,0x0002f8,
0x0002f0,0x00017d,0x000175,0x000179,0x000171,0x00017b,0x000173,0x000177,
0x00017f,0x000176,0x00017e,0x00017a,0x000172,0x00017c,0x000174,0x000178,
0x000170,0x0000bd,0x0000b5,0x0000b9,0x0000b1,0x0000bb,0x0000b3,0x0000b7,
0x0000bf,0x0000b6,0x0000be,0x0000ba,0x0000b2,0x0000bc,0x0000b4,0x0000b8,
0x0000b0,0x00005d,0x000055,0x000059,0x000051,0x00005b,0x000053,0x000057,
0x00005f,0x000056,0x00005e,0x00005a,0x000052,0x00005c,0x000054,0x000058,
0x00002d,0x000025,0x000029,0x000021,0x00002b,0x000023,0x000027,0x00002f,
0x000026,0x00002e,0x00002a,0x000022,0x00002c,0x000024,0x000028,0x000020,
0x000010,0x000000,0x000008,0x000004,0x00000c,0x000002,0x00000a,0x00000e,
```

```
0x000006,0x00000f,0x000007,0x000003,0x00000b,0x000001,0x000009,0x000005,
0x00000d,0x000018,0x000014,0x00001c,0x000012,0x00001a,0x00001e,0x000016,
0x00001f,0x000017,0x000013,0x00001b,0x000011,0x000019,0x000015,0x00001d,
0x000030,0x000038,0x000034,0x00003c,0x000032,0x00003a,0x00003e,0x000036,
0x00003f,0x000037,0x000033,0x00003b,0x000031,0x000039,0x000035,0x00003d,
0x000070,0x000078,0x000074,0x00007c,0x000072,0x00007a,0x00007e,0x000076,
0x00007f,0x000077,0x000073,0x00007b,0x000071,0x000079,0x000075,0x00007d,
0x0000f0,0x0000f8,0x0000f4,0x0000fc,0x0000f2,0x0000fa,0x0000fe,0x0000f6,
0x0000ff,0x0000f7,0x0000f3,0x0000fb,0x0000f1,0x0000f9,0x0000f5,0x0000fd,
0x0001f0,0x0001f8,0x0001f4,0x0001fc,0x0001f2,0x0001fa,0x0001fe,0x0001f6,
0x0001ff,0x0001f7,0x0001f3,0x0001fb,0x0001f1,0x0001f9,0x0001f5,0x0001fd,
0x0003f0,0x0003f8,0x0003f4,0x0003fc,0x0003f2,0x0003fa,0x0003fe,0x0003f6,
0x0003ff,0x0003f7,0x0003f3,0x0003fb,0x0003f1,0x0003f9,0x0003f5,0x0003fd,
0x0007f0,0x0007f8,0x0007f4,0x0007fc,0x0007f2,0x0007fa,0x0007fe,0x0007f6,
0x0007ff,0x0007f7,0x0007f3,0x0007fb,0x0007f1,0x0007f9,0x0007f5,0x0007fd,
0x000ff0,0x000ff8,0x000ff4,0x000ffc,0x000ff2,0x000ffa,0x000ffe,0x000ff6,
0x000fff,0x000ff7,0x000ff3,0x000ffb,0x000ff1,0x000ff9,0x000ff5,0x000ffd,
0x001ff0,0x001ff8,0x001ff4,0x001ffc,0x001ff2,0x001ffa,0x001ffe,0x001ff6,
0x001fff,0x001ff7,0x001ff3,0x001ffb,0x001ff1,0x001ff9,0x001ff5,0x001ffd,
0x003ff0,0x003ff8,0x003ff4,0x003ffc,0x003ff2,0x003ffa,0x003ffe,0x003ff6,
0x003fff,0x003ff7,0x003ff3,0x003ffb,0x003ff1,0x003ff9,0x003ff5,0x003ffd,
0x007ff0,0x007ff8,0x007ff4,0x007ffc,0x007ff2,0x007ffa,0x007ffe,0x007ff6,
0x007fff,0x007ff7,0x007ff3,0x007ffb,0x007ff1,0x007ff9,0x007ff5,0x007ffd,
0x00fff0,0x00fff8,0x00fff4,0x00fffc,0x00fff2,0x00fffa,0x00fffe,0x00fff6,
0x00ffff,0x00fff7,0x00fff3,0x00fffb,0x00fff1,0x00fff9,0x00fff5,0x00fffd,
0x01fff0,0x01fff8,0x01fff4,0x01fffc,0x01fff2,0x01fffa,0x01fffe,0x01fff6,
0x03fff6,0x01fff7,0x01fff3,0x01fffb,0x01fff1,0x01fff9,0x01fff5,0x01fffd,
0x03fff0,0x03fff8,0x03fff4,0x03fffc,0x03fff2,0x07fff2,0x03fffa,0x07fff6,
0x05ffff,0x07fff7,0x03fff3,0x03fffb,0x03fff1,0x03fff9,0x03fff5,0x03fffd,
0x07fff0,0x07fff8,0x07fff4,0x07fffc,0x0ffffc,0x0bfffa,0x0bfffe,0x0ffff6,
0x0bffff,0x0bfff7,0x0bfff3,0x0ffffb,0x0ffff1,0x07fff9,0x07fff5,0x07fffd,
0x0ffff0,0x0ffff8,0x1ffff4,0x17fffc,0x1ffff2,0x1bfffa,0x1ffffe,0x0dffff,
0x0ffff7,0x0ffff3,0x1ffff3,0x1ffff1,0x1bfff1,0x17fff9,0x17fff5,0x1ffffd,
0x1ffff0,0x3ffff8,0x3ffff4,0x37fffc,0x1ffffa,0x3ffffa,0x3bfffe,0x3dffff,
0x1ffff7,0x3ffff3,0x3bfffb,0x3ffff1,0x1ffff9,0x3ffff5,0x37fff5,0x3ffffd,
0x7ffff0,0x7ffff8,0x77fffc,0x7ffffc,0x7ffffa,0x7ffffe,0x6dffff,0x7dffff,
0x7ffff7,0x7ffff3,0x7ffffb,0x7ffff1,0x7ffff9,0x7ffff5,0x6ffffd,0x7ffffd,
0x07ffff,
};

/* Size of each Huffman encoding scheme */
static uint16 sizes[8] = {
        25, 47, 69, 109, 169, 247, 395, 609,
};
```

7. Appendix C: inverse DCT and WHT algorithms

The fast DCT algorithm is described in "A Fast Computational Algorithm for the Discrete Cosine Transform" by Wen-Hsiung Chen, C. Harrison Smith, and S. C. Fralick, *IEEE Transactions On Communications*, Vol. COM-25, No. 9, September 1977, pp. 1004-1009. This is a simple floating-point implementation of that algorithm (˜mc/src/mocprot/downlink/image/xdecomp/invFdct16x16.c on host barsoom at MSSS) and is not intended to be an example of the best possible way to implement the algorithm or to write programs.

```
#define MULTDOUBLE(r,v,c)      (r) = (v) * (c);

static double cosineDouble[16] = {
        1.00000000000000000000e+00,
        9.95184726672196890000e-01,
        9.80785280403230440000e-01,
        9.56940335732208870000e-01,
        9.23879532511286750000e-01,
        8.81921264348355040000e-01,
        8.31469612302545240000e-01,
        7.73010453362736970000e-01,
        7.07106781186547530000e-01,
        6.34393284163645500000e-01,
        5.55570233019602220000e-01,
        4.71396736825997660000e-01,
        3.82683432365089760000e-01,
        2.90284677254462360000e-01,
        1.95090322016128270000e-01,
        9.80171403295606040000e-02,
};

static void DCTinv16Double(in,out) double *in,*out; {
double tmp[16];
register double tmp1,tmp2;
```

```
tmp[0]  =  in[0];
   tmp[1]  =  in[8];
   tmp[2]  =  in[4];
   tmp[3]  =  in[12];
   tmp[4]  =  in[2];
   tmp[5]  =  in[10];
   tmp[6]  =  in[6];
   tmp[7]  =  in[14];
   MULTDOUBLE(tmp1,in[1],cosineDouble[15]); MULTDOUBLE(tmp2,in[15],cosineDouble[1]);
   tmp[8]  = tmp1 - tmp2;
   MULTDOUBLE(tmp1,in[9],cosineDouble[7]); MULTDOUBLE(tmp2,in[7],cosineDouble[9]);
   tmp[9]  = tmp1 - tmp2;
   MULTDOUBLE(tmp1,in[5],cosineDouble[11]); MULTDOUBLE(tmp2,in[11],cosineDouble[5]);
   tmp[10] = tmp1 - tmp2;
   MULTDOUBLE(tmp1,in[13],cosineDouble[3]); MULTDOUBLE(tmp2,in[3],cosineDouble[13]);
   tmp[11] = tmp1 - tmp2;
   MULTDOUBLE(tmp1,in[3],cosineDouble[3]); MULTDOUBLE(tmp2,in[13],cosineDouble[13]);
   tmp[12] = tmp1 + tmp2;
   MULTDOUBLE(tmp1,in[11],cosineDouble[11]); MULTDOUBLE(tmp2,in[5],cosineDouble[5]);
   tmp[13] = tmp1 + tmp2;
   MULTDOUBLE(tmp1,in[7],cosineDouble[7]); MULTDOUBLE(tmp2,in[9],cosineDouble[9]);
   tmp[14] = tmp1 + tmp2;
   MULTDOUBLE(tmp1,in[15],cosineDouble[15]); MULTDOUBLE(tmp2,in[1],cosineDouble[1]);
   tmp[15] = tmp1 + tmp2;

   out[0]  =  tmp[0];
   out[1]  =  tmp[1];
   out[2]  =  tmp[2];
   out[3]  =  tmp[3];
   MULTDOUBLE(tmp1,tmp[4],cosineDouble[14]); MULTDOUBLE(tmp2,tmp[7],cosineDouble[2]);
   out[4]  = tmp1 - tmp2;
   MULTDOUBLE(tmp1,tmp[5],cosineDouble[6]); MULTDOUBLE(tmp2,tmp[6],cosineDouble[10]);
   out[5]  = tmp1 - tmp2;
   MULTDOUBLE(tmp1,tmp[6],cosineDouble[6]); MULTDOUBLE(tmp2,tmp[5],cosineDouble[10]);
   out[6]  = tmp1 + tmp2;
   MULTDOUBLE(tmp1,tmp[7],cosineDouble[14]); MULTDOUBLE(tmp2,tmp[4],cosineDouble[2]);
   out[7]  = tmp1 + tmp2;
   out[8]  =  tmp[8]  + tmp[9];
   out[9]  = -tmp[9]  + tmp[8];
   out[10] = -tmp[10] + tmp[11];
   out[11] =  tmp[11] + tmp[10];
   out[12] =  tmp[12] + tmp[13];
   out[13] = -tmp[13] + tmp[12];
   out[14] = -tmp[14] + tmp[15];
   out[15] =  tmp[15] + tmp[14];

   tmp1    =  out[0] + out[1];
   MULTDOUBLE(tmp[0],tmp1,cosineDouble[8]);
   tmp1    = -out[1] + out[0];
   MULTDOUBLE(tmp[1],tmp1,cosineDouble[8]);
   MULTDOUBLE(tmp1,out[2],cosineDouble[12]); MULTDOUBLE(tmp2,out[3],cosineDouble[4]);
   tmp[2]  = tmp1 - tmp2;
   MULTDOUBLE(tmp1,out[3],cosineDouble[12]); MULTDOUBLE(tmp2,out[2],cosineDouble[4]);
   tmp[3]  = tmp1 + tmp2;
   tmp[4]  =   out[4]      +      out[5];
   tmp[5]  =  -out[5]      +      out[4];
   tmp[6]  =  -out[6]      +      out[7];
   tmp[7]  =   out[7]      +      out[6];
   tmp[8]  =   out[8];
   MULTDOUBLE(tmp1,out[9],cosineDouble[4]); MULTDOUBLE(tmp2,out[14],cosineDouble[12]);
   tmp[9]  = -tmp1 + tmp2;
   MULTDOUBLE(tmp1,out[10],cosineDouble[12]); MULTDOUBLE(tmp2,out[13],cosineDouble[4]);
   tmp[10] = -tmp1 - tmp2;
   tmp[11] =   out[11];
   tmp[12] =   out[12];
   MULTDOUBLE(tmp1,out[13],cosineDouble[12]); MULTDOUBLE(tmp2,out[10],cosineDouble[4]);
   tmp[13] = tmp1 - tmp2;
   MULTDOUBLE(tmp1,out[14],cosineDouble[4]); MULTDOUBLE(tmp2,out[9],cosineDouble[12]);
   tmp[14] = tmp1 + tmp2;
   tmp[15] =   out[15];

   out[0]  =   tmp[0]  + tmp[3];
   out[1]  =   tmp[1]  + tmp[2];
   out[2]  =  -tmp[2]  + tmp[1];
   out[3]  =  -tmp[3]  + tmp[0];
   out[4]  =   tmp[4];
   tmp1    =  -tmp[5]  + tmp[6];
   MULTDOUBLE(out[5],tmp1,cosineDouble[8]);
   tmp1    =   tmp[6]  + tmp[5];
   MULTDOUBLE(out[6],tmp1,cosineDouble[8]);
   out[7]  =   tmp[7];
   out[8]  =   tmp[8]  + tmp[11];
   out[9]  =   tmp[9]  + tmp[10];
   out[10] =  -tmp[10] + tmp[9];
   out[11] =  -tmp[11] + tmp[8];
   out[12] =  -tmp[12] + tmp[15];
   out[13] =  -tmp[13] + tmp[14];
   out[14] =   tmp[14] + tmp[13];
```

```
     out[15] =   tmp[15] + tmp[12];

        tmp[0]  =   out[0]  + out[7];
        tmp[1]  =   out[1]  + out[6];
        tmp[2]  =   out[2]  + out[5];
        tmp[3]  =   out[3]  + out[4];
        tmp[4]  =  -out[4]  + out[3];
        tmp[5]  =  -out[5]  + out[2];
        tmp[6]  =  -out[6]  + out[1];
        tmp[7]  =  -out[7]  + out[0];
        tmp[8]  =   out[8];
        tmp[9]  =   out[9];
        tmp1    =  -out[10] + out[13];
        MULTDOUBLE(tmp[10],tmp1,cosineDouble[8]);
        tmp1    =  -out[11] + out[12];
        MULTDOUBLE(tmp[11],tmp1,cosineDouble[8]);
        tmp1    =   out[12] + out[11];
        MULTDOUBLE(tmp[12],tmp1,cosineDouble[8]);
        tmp1    =   out[13] + out[10];
        MULTDOUBLE(tmp[13],tmp1,cosineDouble[8]);
        tmp[14] =   out[14];
        tmp[15] =   out[15];

        out[0]  =  tmp[0]  + tmp[15];
        out[1]  =  tmp[1]  + tmp[14];
        out[2]  =  tmp[2]  + tmp[13];
        out[3]  =  tmp[3]  + tmp[12];
        out[4]  =  tmp[4]  + tmp[11];
        out[5]  =  tmp[5]  + tmp[10];
        out[6]  =  tmp[6]  + tmp[9];
        out[7]  =  tmp[7]  + tmp[8];
        out[8]  = -tmp[8]  + tmp[7];
        out[9]  = -tmp[9]  + tmp[6];
        out[10] = -tmp[10] + tmp[5];
        out[11] = -tmp[11] + tmp[4];
        out[12] = -tmp[12] + tmp[3];
        out[13] = -tmp[13] + tmp[2];
        out[14] = -tmp[14] + tmp[1];
        out[15] = -tmp[15] + tmp[0];
}

void invFdct16x16(in,out) int16 *in,*out; {
uint32 i,j;
double data[256],*scanData,*other;
double temp;

        scanData = data;

        *(scanData++) = (uint16)(*(in++));

        for (i = 1; i < 256; i++) {
                *(scanData++) = *(in++);
        };

        for (i = 0, scanData = data; i < 16; i++, scanData += 16) {
                DCTinv16Double(scanData,scanData);
        };

        for (i = 0, scanData = data, other = data+16; i < 16; i++, other += 17) {
        double *scanOther;

                scanData += i+1;
                scanOther = other;

                for (j = i+1; j < 16; j++, scanData++, scanOther += 16) {
                             temp       = *scanData;
                             *scanData  = *scanOther;
                             *scanOther = temp;
                };
        };

        for (i = 0, scanData = data; i < 16; i++, scanData += 16) {
                DCTinv16Double(scanData,scanData);
        };

        for (i = 0, scanData = data, other = data+16; i < 16; i++, other += 17) {
        double *scanOther;

                scanData += i+1;
                scanOther = other;

                for (j = i+1; j < 16; j++, scanData++, scanOther += 16) {
                             temp       = *scanData;
                             *scanData  = *scanOther;
                             *scanOther = temp;
                };
        };
```

```
        for (i = 0, scanData = data; i < 256; i++) {
           int16 cur;
                     cur = *(scanData++) / 127.0 + 0.5;

                     if (cur < 0) {
                                  cur = 0;
                     };

                     if (cur > 255) {
                                  cur = 255;
                     };

                     *(out++) = cur;
           };
}
```

For a discussion of the Walsh-Hadamard transform, see any book on image compression, for example R.J. Clarke, TRANSFORM CODING OF IMAGES, Academic Press, 1985.

This example module calculates a "sequency" ordered, two dimensional inverse Walsh-Hadamard transform (WHT) on 16 x 16 blocks of data. It is done as two one dimensional transforms (one of the rows followed by one of the columns). Each one dimensional transform is implemented as a 16 point, 4 stage "butterfly".

These routines, taken from the MOC flight software, have been highly optimized to produce very fast 32000 executable code but still use C for coding (thus allowing compilation on other machines).

```
#include stdio.h

#include "fs.h"

/*
 * This defines a four input (and output), two stage "butterfly"
 * calculation done completely in registers (once the data is read from
 * memory.  Four input and two stages was picked to maximize the use of
 * the 32000's registers.  Eight of these are required to do a 16 point,
 * one dimensional WHT.  The "simple" formulas for this "butterfly" are:
 *
 *      n0 = i0 + i1
 *      n1 = i0 - i1          First stage
 *      n2 = i2 + i3
 *      n3 = i2 - i3
 *
 *      o0 = n0 + n2
 *      o1 = n1 + n3          Second stage
 *      o2 = n0 - n2
 *      o3 = n1 - n3
 *
 * All data (in and out) is assumed to be 16 bit integers.  "in" is the
 * base address of the input data array and "ii" is the scaling factor to
 * use on the next four indexes into "in" (this allows moving by rows or
 * columns through a two dimensional array stored as a one dimensional set
 * of numbers).  "i0", "i1", "i2", and "i3" are the unscaled indexes into
 * "in".  "out" is the base address of the output data array and "oi" is
 * the scaling factor to use on the next four indexes into "out" (this
 * allows moving by rows or columns through a two dimensional array stored
 * as a one dimensional set of numbers).  "o0", "o1", "o2", and "o3" are
 * the unscaled indexes into "out".
 */

#define BUTTERFLY4(in,ii,i0,i1,i2,i3,out,oi,o0,o1,o2,o3)\
        {                                    \
                 register int32 t0,t1,t2,t3,t4;\
                                             \
                 /* Load input into registers */\
                 t0 = in[(ii)*(i0)];         \
                 t1 = in[(ii)*(i1)];         \
                 t2 = in[(ii)*(i2)];         \
                 t3 = in[(ii)*(i3)];         \
                                             \
                 /* Do first stage */        \
                 t4 = t0;                    \
                 t4 += t1;                   \
                 t0 -= t1;                   \
                                             \
                 t1 = t2;                    \
                 t1 += t3;                   \
                 t2 -= t3;                   \
                                             \
                 /* Do second stage */       \
                 t3 = t4;                    \
                 t3 += t1;                   \
                 t4 -= t1;                   \
```

```
                                        \
                t1 = t0;                \
                t1 += t2;               \
                t0 -= t2;               \
                                        \
                /* Store results from registers */\
                out[(oi)*(o0)] = t3;    \
                out[(oi)*(o1)] = t1;    \
                out[(oi)*(o2)] = t4;    \
                out[(oi)*(o3)] = t0;    \
        }

static void invFwht16_row(in,out) register int32 *in,*out; {
/*
 * This function does a 16 point, one dimensional inverse WHT on 16, 32
 * bit integers stored as a vector (as in the rows of a two dimensional
 * array) and puts the results in a 32 bit integer vector.  The transform
 * is not normalized but is in "sequency" order.
 *
 * pre:
 *      "in" - the 16 inputs stored as 32 bit integers in a vector.
 * post:
 *      "out" - the 16 outputs stored as 32 bit integers in a vector.
 */
int32 data[32];                         /* Temporary storage used between stages */
register int32 *tmp;                    /* Register pointer to the temporary storage */

        /* Point at temporary storage */
        tmp = data;

        /* Perform first two stages of 16 point butterfly */
        BUTTERFLY4(in , 1, 0, 1, 2, 3,tmp, 1, 0, 1, 2, 3);
        BUTTERFLY4(in , 1, 4, 5, 6, 7,tmp, 1, 4, 5, 6, 7);
        BUTTERFLY4(in , 1, 8, 9,10,11,tmp, 1, 8, 9,10,11);
        BUTTERFLY4(in , 1,12,13,14,15,tmp, 1,12,13,14,15);

        /*
        * Perform last two stages of 16 point butterfly and store in
        * "sequency" order
        */
        BUTTERFLY4(tmp, 1, 0, 4, 8,12,out, 1, 0, 3, 1, 2);
        BUTTERFLY4(tmp, 1, 1, 5, 9,13,out, 1,15,12,14,13);
        BUTTERFLY4(tmp, 1, 2, 6,10,14,out, 1, 7, 4, 6, 5);
        BUTTERFLY4(tmp, 1, 3, 7,11,15,out, 1, 8,11, 9,10);
}

static void invFwht16_col(in,out) register int32 *in,*out; {
/*
 * This function does a 16 point, one dimensional inverse WHT on 16, 32
 * bit integers stored as a vector in every 16th location (as in the
 * columns of a two dimensional array stored as a one dimensional array by
 * rows) and puts the results out in a similar manner.  The transform is
 * not normalized but is in "sequency" order.
 *
 * pre:
 *      "in" - the 16 inputs stored as 32 bit integers in every 16th location.
 * post:
 *      "out" - the 16 outputs stored as 32 bit integers in every 16th location.
 */
int32 data[16];                         /* Temporary storage used between stages */
register int32 *tmp;                    /* Register pointer to the temporary storage */

        /* Point at temporary storage */
        tmp = data;

        /* Perform first two stages of 16 point butterfly */
        BUTTERFLY4(in ,16, 0, 1, 2, 3,tmp, 1, 0, 1, 2, 3);
        BUTTERFLY4(in ,16, 4, 5, 6, 7,tmp, 1, 4, 5, 6, 7);
        BUTTERFLY4(in ,16, 8, 9,10,11,tmp, 1, 8, 9,10,11);
        BUTTERFLY4(in ,16,12,13,14,15,tmp, 1,12,13,14,15);

        /*
        * Perform last two stages of 16 point butterfly and store in
        * "sequency" order
        */
        BUTTERFLY4(tmp, 1, 0, 4, 8,12,out,16, 0, 3, 1, 2);
        BUTTERFLY4(tmp, 1, 1, 5, 9,13,out,16,15,12,14,13);
        BUTTERFLY4(tmp, 1, 2, 6,10,14,out,16, 7, 4, 6, 5);
        BUTTERFLY4(tmp, 1, 3, 7,11,15,out,16, 8,11, 9,10);
}

void invFwht16x16(in,out) register int16 *in,*out; {
/*
 * This function does a "sequency" ordered WHT on a 16 x 16 array of data
 * (stored as 16 bit integers) stored in 256 contiguous locations.  The
 * transform is normalized.  The input is assumed to be 16 bit signed
 * integers EXCEPT for the DC entry which is be treated as UNSIGNED.  The
 * result is stored in a 16 x 16 array of the same structure.  The output
```

```
 * is all 8 bit, unsigned integers.
 *
 * pre:
 *      "in" - the 16 x 16 input block data stored as 16 bit integers.
 * post:
 *      "out" - the 16 x 16 output block data stored as 16 bit integers.
 */
uint32 i;                                   /* Generic looping variable */
int32 data[256];              /* Temporary storage for transform */

        /* Convert 16 bit integers to 32 bit integers */
        {
        register int16 *scanIn;
        register int32 *scanData;

        scanIn = in;
        scanData = data;

        *(scanData++) = (uint16)(*(scanIn++));

        for (i = 1; i < 256; i++) {
                 *(scanData++) = *(scanIn++);
        };
        };

        {
        register int32 *scanData;           /* Current row start in "data" */

        /*
        * Pass each row in "data" array (as a vector of size 16) to the
        * 16 point, 1D inverse WHT and store the results in contiguous
        * 16 point locations in "data".  At completion all rows have been
        * inverse transformed in one dimension.
        */
        for (i = 0, scanData = data; i < 16; i++, scanData += 16) {
                 invFwht16_row(scanData,scanData);
        };
        };

        {
        register int32 *scanData;           /* Current column start in "data" */

        /*
        * Inverse transform each column in the 16 x 16 block stored by rows
        * as a 256 point vector.
        */
        for (i = 0, scanData = data; i < 16; i++, scanData++) {
                 invFwht16_col(scanData,scanData);
        };
        };

        /* Convert 32 bit integers to 16 bit integers */
        {
        register int32 *scanData;
        register int16 *scanOut;

        scanData = data;
        scanOut = out;

        for (i = 0; i < 256; i++) {
        register int32 cur;
                 cur = *(scanData++);


                 cur >>= 8;

                 if (cur < 0) {
                              cur = 0;
                 };

                 if (cur > 255) {
                              cur = 255;
                 };

                 *(scanOut++) = cur;
        };
        };
}
```

8. Appendix D: predictive Huffman code tables

(IPS/predcode.h from the flight software.)

```
/* IDENTITY; NO COMPRESSION -- dumped from 'default.code' */
uint16 Code0Bits[256] = {
0x0000, 0x0001, 0x0002, 0x0003, 0x0004, 0x0005, 0x0006, 0x0007,
0x0008, 0x0009, 0x000a, 0x000b, 0x000c, 0x000d, 0x000e, 0x000f,
0x0010, 0x0011, 0x0012, 0x0013, 0x0014, 0x0015, 0x0016, 0x0017,
0x0018, 0x0019, 0x001a, 0x001b, 0x001c, 0x001d, 0x001e, 0x001f,
0x0020, 0x0021, 0x0022, 0x0023, 0x0024, 0x0025, 0x0026, 0x0027,
0x0028, 0x0029, 0x002a, 0x002b, 0x002c, 0x002d, 0x002e, 0x002f,
0x0030, 0x0031, 0x0032, 0x0033, 0x0034, 0x0035, 0x0036, 0x0037,
0x0038, 0x0039, 0x003a, 0x003b, 0x003c, 0x003d, 0x003e, 0x003f,
0x0040, 0x0041, 0x0042, 0x0043, 0x0044, 0x0045, 0x0046, 0x0047,
0x0048, 0x0049, 0x004a, 0x004b, 0x004c, 0x004d, 0x004e, 0x004f,
0x0050, 0x0051, 0x0052, 0x0053, 0x0054, 0x0055, 0x0056, 0x0057,
0x0058, 0x0059, 0x005a, 0x005b, 0x005c, 0x005d, 0x005e, 0x005f,
0x0060, 0x0061, 0x0062, 0x0063, 0x0064, 0x0065, 0x0066, 0x0067,
0x0068, 0x0069, 0x006a, 0x006b, 0x006c, 0x006d, 0x006e, 0x006f,
0x0070, 0x0071, 0x0072, 0x0073, 0x0074, 0x0075, 0x0076, 0x0077,
0x0078, 0x0079, 0x007a, 0x007b, 0x007c, 0x007d, 0x007e, 0x007f,
0x0080, 0x0081, 0x0082, 0x0083, 0x0084, 0x0085, 0x0086, 0x0087,
0x0088, 0x0089, 0x008a, 0x008b, 0x008c, 0x008d, 0x008e, 0x008f,
0x0090, 0x0091, 0x0092, 0x0093, 0x0094, 0x0095, 0x0096, 0x0097,
0x0098, 0x0099, 0x009a, 0x009b, 0x009c, 0x009d, 0x009e, 0x009f,
0x00a0, 0x00a1, 0x00a2, 0x00a3, 0x00a4, 0x00a5, 0x00a6, 0x00a7,
0x00a8, 0x00a9, 0x00aa, 0x00ab, 0x00ac, 0x00ad, 0x00ae, 0x00af,
0x00b0, 0x00b1, 0x00b2, 0x00b3, 0x00b4, 0x00b5, 0x00b6, 0x00b7,
0x00b8, 0x00b9, 0x00ba, 0x00bb, 0x00bc, 0x00bd, 0x00be, 0x00bf,
0x00c0, 0x00c1, 0x00c2, 0x00c3, 0x00c4, 0x00c5, 0x00c6, 0x00c7,
0x00c8, 0x00c9, 0x00ca, 0x00cb, 0x00cc, 0x00cd, 0x00ce, 0x00cf,
0x00d0, 0x00d1, 0x00d2, 0x00d3, 0x00d4, 0x00d5, 0x00d6, 0x00d7,
0x00d8, 0x00d9, 0x00da, 0x00db, 0x00dc, 0x00dd, 0x00de, 0x00df,
0x00e0, 0x00e1, 0x00e2, 0x00e3, 0x00e4, 0x00e5, 0x00e6, 0x00e7,
0x00e8, 0x00e9, 0x00ea, 0x00eb, 0x00ec, 0x00ed, 0x00ee, 0x00ef,
0x00f0, 0x00f1, 0x00f2, 0x00f3, 0x00f4, 0x00f5, 0x00f6, 0x00f7,
0x00f8, 0x00f9, 0x00fa, 0x00fb, 0x00fc, 0x00fd, 0x00fe, 0x00ff,
};

uint8 Code0Len[256] = {
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
};

/* dumped from 'exp01.code' */
uint16 Code1Bits[256] = {
0x0000, 0x0001, 0x000d, 0x0055, 0x00f5, 0x0375, 0x0135, 0x1135,
0x5a75, 0x1a75, 0x6a75, 0x2a75, 0x4a75, 0x0a75, 0x7275, 0x3275,
0x5275, 0x1275, 0x6275, 0x2275, 0x4275, 0x0275, 0x7c75, 0x3c75,
0x5c75, 0x1c75, 0x6c75, 0x2c75, 0x4c75, 0x0c75, 0x7475, 0x3475,
0x5475, 0x1475, 0x6475, 0x2475, 0x4475, 0x0475, 0x7875, 0x3875,
0x5875, 0x1875, 0x6875, 0x2875, 0x4875, 0x0875, 0x7075, 0x3075,
0x5075, 0x1075, 0x6075, 0x2075, 0x4075, 0x0075, 0x7fb5, 0x3fb5,
0x5fb5, 0x1fb5, 0x6fb5, 0x2fb5, 0x4fb5, 0x0fb5, 0x77b5, 0x37b5,
0x57b5, 0x17b5, 0x67b5, 0x27b5, 0x47b5, 0x07b5, 0x7bb5, 0x3bb5,
0x5bb5, 0x1bb5, 0x6bb5, 0x2bb5, 0x4bb5, 0x0bb5, 0x73b5, 0x33b5,
0x53b5, 0x13b5, 0x63b5, 0x23b5, 0x43b5, 0x03b5, 0x7db5, 0x3db5,
0x5db5, 0x1db5, 0x6db5, 0x2db5, 0x4db5, 0x0db5, 0x75b5, 0x35b5,
0x55b5, 0x15b5, 0x65b5, 0x25b5, 0x45b5, 0x05b5, 0x79b5, 0x39b5,
0x59b5, 0x19b5, 0x69b5, 0x29b5, 0x49b5, 0x09b5, 0x71b5, 0x31b5,
0x51b5, 0x11b5, 0x61b5, 0x21b5, 0x41b5, 0x01b5, 0x7eb5, 0x3eb5,
0x5eb5, 0x1eb5, 0x3a75, 0x2eb5, 0x4eb5, 0x6eb5, 0x6675, 0x1675,
0x5675, 0x16b5, 0x66b5, 0x26b5, 0x46b5, 0x06b5, 0x7ab5, 0x3ab5,
0x5ab5, 0x1ab5, 0x6ab5, 0x2ab5, 0x4ab5, 0x0ab5, 0x72b5, 0x32b5,
0x52b5, 0x12b5, 0x62b5, 0x22b5, 0x42b5, 0x02b5, 0x7cb5, 0x3cb5,
0x5cb5, 0x1cb5, 0x6cb5, 0x2cb5, 0x4cb5, 0x0cb5, 0x74b5, 0x34b5,
0x54b5, 0x14b5, 0x64b5, 0x24b5, 0x44b5, 0x04b5, 0x78b5, 0x38b5,
0x58b5, 0x18b5, 0x68b5, 0x28b5, 0x48b5, 0x08b5, 0x70b5, 0x30b5,
0x50b5, 0x10b5, 0x60b5, 0x20b5, 0x40b5, 0x00b5, 0x0eb5, 0x3f35,
0x7f35, 0x1f35, 0x6f35, 0x2f35, 0x4f35, 0x0f35, 0x7735, 0x3735,
0x5735, 0x1735, 0x6735, 0x2735, 0x4735, 0x0735, 0x7b35, 0x3b35,
0x5b35, 0x1b35, 0x6b35, 0x2b35, 0x4b35, 0x0b35, 0x7335, 0x3335,
0x5335, 0x1335, 0x6335, 0x2335, 0x5f35, 0x4335, 0x7d35, 0x3d35,
0x5d35, 0x1d35, 0x6d35, 0x2d35, 0x4d35, 0x0d35, 0x7535, 0x3535,
0x5535, 0x1535, 0x6535, 0x0335, 0x2535, 0x0535, 0x7935, 0x3935,
0x5935, 0x1935, 0x6935, 0x4535, 0x2935, 0x0935, 0x7135, 0x4935,
0x5135, 0x3135, 0x56b5, 0x36b5, 0x76b5, 0x7a75, 0x0675, 0x4675,
```

```
0x2675, 0x3675, 0x0e75, 0x0175, 0x0035, 0x0015, 0x0005, 0x0003,
};

uint8 Code1Len[256] = {
 1,  3,  4,  7,  8, 10, 13, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 12, 10,  9,  7,  5,  2,
};

/* dumped from 'exp02.code' */
uint16 Code2Bits[256] = {
0x0002, 0x0000, 0x0005, 0x000c, 0x0011, 0x003c, 0x00fc, 0x0101,
0x0081, 0x0181, 0x0d81, 0x0dc1, 0x09c1, 0x0ec1, 0x76c1, 0x36c1,
0x56c1, 0x16c1, 0x66c1, 0x26c1, 0x46c1, 0x06c1, 0x7ac1, 0x3ac1,
0x5ac1, 0x1ac1, 0x6ac1, 0x2ac1, 0x4ac1, 0x0ac1, 0x72c1, 0x32c1,
0x52c1, 0x12c1, 0x62c1, 0x22c1, 0x42c1, 0x02c1, 0x7cc1, 0x3cc1,
0x5cc1, 0x1cc1, 0x6cc1, 0x2cc1, 0x4cc1, 0x0cc1, 0x74c1, 0x34c1,
0x54c1, 0x14c1, 0x64c1, 0x24c1, 0x44c1, 0x04c1, 0x78c1, 0x38c1,
0x58c1, 0x18c1, 0x68c1, 0x28c1, 0x48c1, 0x08c1, 0x70c1, 0x30c1,
0x50c1, 0x10c1, 0x60c1, 0x20c1, 0x40c1, 0x00c1, 0x7f41, 0x3f41,
0x5f41, 0x1f41, 0x6f41, 0x2f41, 0x4f41, 0x0f41, 0x7741, 0x3741,
0x5741, 0x1741, 0x6741, 0x2741, 0x4741, 0x0741, 0x7b41, 0x3b41,
0x5b41, 0x1b41, 0x6b41, 0x2b41, 0x4b41, 0x0b41, 0x7341, 0x3341,
0x5341, 0x1341, 0x6341, 0x2341, 0x4341, 0x0341, 0x7d41, 0x3d41,
0x5d41, 0x1d41, 0x6d41, 0x2d41, 0x4d41, 0x0d41, 0x7541, 0x3541,
0x5541, 0x1541, 0x6541, 0x2541, 0x4541, 0x0541, 0x7941, 0x3941,
0x7ec1, 0x6ec1, 0x21c1, 0x41c1, 0x4ec1, 0x5941, 0x61c1, 0x11c1,
0x51c1, 0x1141, 0x6141, 0x2141, 0x4141, 0x0141, 0x7e41, 0x3e41,
0x5e41, 0x1e41, 0x6e41, 0x2e41, 0x4e41, 0x0e41, 0x7641, 0x3641,
0x5641, 0x1641, 0x6641, 0x2641, 0x4641, 0x0641, 0x7a41, 0x3a41,
0x5a41, 0x1a41, 0x6a41, 0x2a41, 0x4a41, 0x0a41, 0x7241, 0x3241,
0x5241, 0x1241, 0x6241, 0x2241, 0x4241, 0x0241, 0x7c41, 0x3c41,
0x5c41, 0x1c41, 0x6c41, 0x2c41, 0x4c41, 0x0c41, 0x7441, 0x3441,
0x5441, 0x1441, 0x1941, 0x2941, 0x4941, 0x0441, 0x7841, 0x3841,
0x5841, 0x1841, 0x6841, 0x2841, 0x4841, 0x0841, 0x7041, 0x3041,
0x5041, 0x1041, 0x6041, 0x2041, 0x4041, 0x0041, 0x7f81, 0x3f81,
0x5f81, 0x1f81, 0x6f81, 0x2f81, 0x4f81, 0x2441, 0x0f81, 0x3781,
0x5781, 0x1781, 0x6781, 0x2781, 0x4781, 0x0781, 0x7b81, 0x3b81,
0x5b81, 0x1b81, 0x6b81, 0x7781, 0x2b81, 0x0b81, 0x7381, 0x3381,
0x5381, 0x1381, 0x4b81, 0x6381, 0x4381, 0x2381, 0x0381, 0x31c1,
0x4441, 0x6441, 0x7141, 0x0941, 0x2ec1, 0x6941, 0x1ec1, 0x5ec1,
0x3ec1, 0x01c1, 0x5141, 0x3141, 0x19c1, 0x05c1, 0x0581, 0x03c1,
0x0281, 0x0001, 0x007c, 0x0021, 0x001c, 0x0009, 0x0004, 0x0003,
};

uint8 Code2Len[256] = {
 2,  3,  3,  5,  5,  7,  8,  9, 10, 11, 12, 12, 13, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 14, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 13, 12, 12, 10, 10,  9,  8,  6,  6,  4,  4,  2,
};

/* dumped from 'exp04.code' */
uint16 Code3Bits[256] = {
0x0004, 0x0006, 0x0007, 0x0005, 0x000b, 0x000d, 0x0018, 0x001d,
0x0038, 0x007d, 0x00f8, 0x0003, 0x0001, 0x0183, 0x01c3, 0x0343,
0x02c3, 0x0683, 0x0e81, 0x08c3, 0x0181, 0x0c83, 0x1181, 0x04c3,
0x5f41, 0x1f41, 0x6f41, 0x2f41, 0x4f41, 0x0f41, 0x7741, 0x3741,
0x5741, 0x1741, 0x6741, 0x2741, 0x4741, 0x0741, 0x7b41, 0x3b41,
0x5b41, 0x1b41, 0x6b41, 0x2b41, 0x4b41, 0x0b41, 0x7341, 0x3341,
0x5341, 0x1341, 0x6341, 0x2341, 0x4341, 0x0341, 0x7d41, 0x3d41,
0x5d41, 0x1d41, 0x6d41, 0x2d41, 0x4d41, 0x0d41, 0x7541, 0x3541,
0x5541, 0x1541, 0x6541, 0x2541, 0x4541, 0x0541, 0x7941, 0x3941,
```

```
0x5941, 0x1941, 0x6941, 0x2941, 0x4941, 0x0941, 0x7141, 0x3141,
0x5141, 0x1141, 0x6141, 0x2141, 0x4141, 0x0141, 0x7e41, 0x3e41,
0x5e41, 0x1e41, 0x6e41, 0x2e41, 0x4e41, 0x0e41, 0x7641, 0x3641,
0x5641, 0x1641, 0x6641, 0x2641, 0x4641, 0x0641, 0x7a41, 0x3a41,
0x5a41, 0x1a41, 0x6a41, 0x2a41, 0x4a41, 0x0a41, 0x7241, 0x4883,
0x7f41, 0x5883, 0x1883, 0x0483, 0x7883, 0x3f41, 0x3241, 0x3c41,
0x5c41, 0x1c41, 0x2483, 0x4483, 0x4c41, 0x0c41, 0x6483, 0x1483,
0x5483, 0x1441, 0x6441, 0x2441, 0x4441, 0x0441, 0x7841, 0x3841,
0x5841, 0x1841, 0x6841, 0x2841, 0x4841, 0x0841, 0x7041, 0x3041,
0x5041, 0x1041, 0x6041, 0x2041, 0x4041, 0x0041, 0x7f81, 0x3f81,
0x5f81, 0x1f81, 0x6f81, 0x2f81, 0x4f81, 0x0f81, 0x7781, 0x3781,
0x5781, 0x1781, 0x5241, 0x0241, 0x4781, 0x7c41, 0x6781, 0x3b81,
0x5b81, 0x1b81, 0x6b81, 0x2b81, 0x4b81, 0x0b81, 0x7381, 0x3381,
0x5381, 0x1381, 0x6381, 0x2381, 0x4381, 0x0381, 0x7d81, 0x3d81,
0x5d81, 0x1d81, 0x6d81, 0x2781, 0x0781, 0x7b81, 0x2d81, 0x3581,
0x5581, 0x1581, 0x6581, 0x2581, 0x4581, 0x0581, 0x7981, 0x7581,
0x4d81, 0x1981, 0x6981, 0x2981, 0x4981, 0x5981, 0x0981, 0x3181,
0x7181, 0x24c3, 0x3981, 0x0d81, 0x2241, 0x6241, 0x1241, 0x0083,
0x4083, 0x2083, 0x6083, 0x2883, 0x4241, 0x6883, 0x1083, 0x5083,
0x3083, 0x3883, 0x5441, 0x3441, 0x7441, 0x2c41, 0x6c41, 0x7083,
0x0883, 0x3483, 0x14c3, 0x1c83, 0x0cc3, 0x00c3, 0x0681, 0x0283,
0x0281, 0x0143, 0x0081, 0x0043, 0x0101, 0x00c1, 0x0078, 0x003d,
0x0023, 0x0021, 0x0013, 0x0011, 0x0008, 0x0009, 0x0000, 0x0002,
};

uint8 Code3Len[256] = {
 3,  3,  3,  4,  4,  5,  6,  6,  7,  7,  8,  8,  9,  9,  9, 10,
10, 11, 12, 12, 13, 13, 14, 14, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 14, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 13, 13, 12, 12, 12, 11,
11, 10, 10,  9,  9,  8,  8,  7,  6,  6,  5,  5,  5,  4,  4,  3,
};

/* dumped from 'exp06.code' */
uint16 Code4Bits[256] = {
0x0005, 0x0007, 0x0004, 0x0001, 0x0010, 0x000a, 0x0009, 0x0018,
0x001a, 0x0019, 0x0038, 0x003a, 0x0039, 0x0078, 0x007a, 0x0000,
0x0002, 0x0006, 0x0300, 0x0102, 0x0306, 0x0480, 0x0482, 0x0486,
0x0280, 0x0282, 0x0682, 0x0a80, 0x1680, 0x1d46, 0x0b46, 0x3e80,
0x2d46, 0x2680, 0x0680, 0x3646, 0x5646, 0x1646, 0x6646, 0x2646,
0x4646, 0x0646, 0x7a46, 0x3a46, 0x5a46, 0x1a46, 0x6a46, 0x2a46,
0x4a46, 0x0a46, 0x7246, 0x3246, 0x5246, 0x1246, 0x6246, 0x2246,
0x4246, 0x0246, 0x7c46, 0x3c46, 0x5c46, 0x1c46, 0x6c46, 0x2c46,
0x4c46, 0x0c46, 0x7446, 0x3446, 0x5446, 0x1446, 0x6446, 0x2446,
0x4446, 0x0446, 0x7846, 0x3846, 0x5846, 0x1846, 0x6846, 0x2846,
0x4846, 0x0846, 0x7046, 0x3046, 0x5046, 0x1046, 0x6046, 0x2046,
0x4046, 0x0046, 0x7f86, 0x3f86, 0x5f86, 0x1f86, 0x6f86, 0x2f86,
0x4f86, 0x0f86, 0x7786, 0x0546, 0x2546, 0x4546, 0x6786, 0x2786,
0x4786, 0x0786, 0x7b86, 0x3b86, 0x5b86, 0x1b86, 0x6b86, 0x2b86,
0x7646, 0x3786, 0x7386, 0x1546, 0x6546, 0x1386, 0x6386, 0x2386,
0x4386, 0x0386, 0x3546, 0x5546, 0x5d86, 0x1d86, 0x7546, 0x0d46,
0x4d46, 0x0d86, 0x7586, 0x3586, 0x5586, 0x1586, 0x6586, 0x2586,
0x4586, 0x0586, 0x7986, 0x3986, 0x5986, 0x1986, 0x6986, 0x2986,
0x4986, 0x5786, 0x4b86, 0x3186, 0x5186, 0x1186, 0x6186, 0x0b86,
0x0986, 0x0186, 0x7e86, 0x3e86, 0x5e86, 0x1e86, 0x6e86, 0x2e86,
0x4e86, 0x0e86, 0x7686, 0x3686, 0x5686, 0x1686, 0x6686, 0x2686,
0x7186, 0x2186, 0x7a86, 0x4186, 0x4686, 0x1a86, 0x6a86, 0x2a86,
0x4a86, 0x0a86, 0x7286, 0x0686, 0x3a86, 0x5a86, 0x3286, 0x2286,
0x4286, 0x5286, 0x6680, 0x0286, 0x6286, 0x1286, 0x1786, 0x0e46,
0x4e46, 0x2e46, 0x4d86, 0x2d86, 0x6d86, 0x3d86, 0x7d86, 0x5386,
0x3386, 0x6e46, 0x1e46, 0x5e46, 0x3e46, 0x7e46, 0x0146, 0x4146,
0x2146, 0x6146, 0x1146, 0x5146, 0x3146, 0x7146, 0x0946, 0x4946,
0x2946, 0x6946, 0x1946, 0x5946, 0x3946, 0x7946, 0x3346, 0x4680,
0x1346, 0x1e80, 0x1b46, 0x0346, 0x0e80, 0x1a80, 0x0e82, 0x0a82,
0x0746, 0x0086, 0x0082, 0x0080, 0x0106, 0x0302, 0x0100, 0x0182,
0x0180, 0x00c6, 0x00fa, 0x00f8, 0x0079, 0x0042, 0x0040, 0x0026,
0x0022, 0x0020, 0x0016, 0x0012, 0x0008, 0x000e, 0x000c, 0x0003,
};

uint8 Code4Len[256] = {
 3,  3,  4,  4,  5,  5,  5,  6,  6,  6,  7,  7,  7,  8,  8,  9,
 9,  9, 10, 10, 10, 11, 11, 11, 12, 12, 12, 13, 13, 13, 13, 14,
14, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
```

```
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 15,
14, 14, 13, 13, 13, 13, 12, 12, 11, 11, 11, 11, 10, 10, 10,  9,
 9,  8,  8,  8,  7,  7,  7,  6,  6,  6,  5,  5,  5,  4,  4,  3,
};

/* dumped from ’exp08.code’ */
uint16 Code5Bits[256] = {
0x0008, 0x000c, 0x0006, 0x000d, 0x0007, 0x0012, 0x0009, 0x000b,
0x0010, 0x001a, 0x0019, 0x001b, 0x0040, 0x004a, 0x0041, 0x007b,
0x00f0, 0x00fa, 0x0003, 0x00c3, 0x000a, 0x0001, 0x0083, 0x0100,
0x010a, 0x028a, 0x0183, 0x0300, 0x030a, 0x0301, 0x0043, 0x0480,
0x008a, 0x0081, 0x0a43, 0x1080, 0x048a, 0x0679, 0x0643, 0x1e43,
0x148a, 0x3a79, 0x0c80, 0x1643, 0x4079, 0x0079, 0x7f81, 0x3f81,
0x5f81, 0x1f81, 0x6f81, 0x2f81, 0x4f81, 0x0f81, 0x7781, 0x3781,
0x5781, 0x1781, 0x6781, 0x2781, 0x4781, 0x0781, 0x7b81, 0x3b81,
0x5b81, 0x1b81, 0x6b81, 0x2b81, 0x4b81, 0x0b81, 0x7381, 0x3381,
0x5381, 0x1381, 0x6381, 0x2381, 0x4381, 0x0381, 0x7d81, 0x3d81,
0x5d81, 0x1d81, 0x6d81, 0x2d81, 0x4d81, 0x0d81, 0x7581, 0x3581,
0x5581, 0x1581, 0x6581, 0x2581, 0x4581, 0x0581, 0x6c79, 0x6079,
0x7c79, 0x3c79, 0x6279, 0x2279, 0x3279, 0x5279, 0x7181, 0x3181,
0x5181, 0x1181, 0x6181, 0x2079, 0x7981, 0x0181, 0x7e81, 0x3e81,
0x5e81, 0x1e81, 0x6e81, 0x0a79, 0x7279, 0x0e81, 0x7681, 0x3681,
0x5681, 0x1681, 0x2a79, 0x4a79, 0x4681, 0x0681, 0x6a79, 0x1a79,
0x5a79, 0x1a81, 0x6a81, 0x2a81, 0x4a81, 0x3981, 0x2181, 0x3281,
0x5281, 0x1281, 0x6281, 0x2281, 0x4181, 0x0a81, 0x7c81, 0x3c81,
0x5c81, 0x1c81, 0x6c81, 0x2c81, 0x4c81, 0x0c81, 0x7481, 0x3481,
0x5481, 0x7281, 0x4281, 0x2481, 0x0281, 0x1481, 0x7881, 0x3881,
0x5881, 0x1881, 0x6881, 0x6481, 0x4481, 0x2881, 0x748a, 0x4881,
0x0881, 0x4c80, 0x348a, 0x0481, 0x6981, 0x1981, 0x5981, 0x1079,
0x5079, 0x3079, 0x7079, 0x1c79, 0x4981, 0x2981, 0x5c79, 0x0879,
0x4879, 0x2879, 0x6879, 0x0279, 0x0981, 0x4279, 0x1879, 0x5879,
0x3879, 0x1279, 0x5a81, 0x3a81, 0x7a81, 0x2681, 0x6681, 0x4e81,
0x2e81, 0x7879, 0x0479, 0x4479, 0x2479, 0x6479, 0x1479, 0x5479,
0x3479, 0x7479, 0x0c79, 0x4c79, 0x2c79, 0x3643, 0x2e79, 0x0e79,
0x2c80, 0x0e43, 0x1e79, 0x1679, 0x1c80, 0x0080, 0x0243, 0x0c8a,
0x088a, 0x0880, 0x0443, 0x0701, 0x070a, 0x0700, 0x0383, 0x0101,
0x0280, 0x0143, 0x0179, 0x018a, 0x0180, 0x0000, 0x00f9, 0x007a,
0x0070, 0x003b, 0x0039, 0x003a, 0x0030, 0x0023, 0x0021, 0x002a,
0x0020, 0x0013, 0x0011, 0x0002, 0x000f, 0x0005, 0x000e, 0x0004,
};

uint8 Code5Len[256] = {
 4,  4,  4,  4,  4,  5,  5,  5,  6,  6,  6,  6,  7,  7,  7,  7,
 8,  8,  8,  8,  9,  9,  9, 10, 10, 10, 10, 11, 11, 11, 11, 12,
12, 12, 12, 13, 13, 13, 13, 13, 14, 14, 15, 14, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 14, 14, 14, 14, 13, 13, 13, 13, 13, 12, 12,
12, 12, 11, 11, 11, 11, 10, 10, 10,  9,  9,  9,  9,  9,  8,  8,
 8,  7,  7,  7,  7,  6,  6,  6,  6,  5,  5,  5,  4,  4,  4,  4,
};

/* dumped from ’exp10.code’ */
uint16 Code6Bits[256] = {
0x000a, 0x000e, 0x0005, 0x000f, 0x0008, 0x001c, 0x0011, 0x000b,
0x0010, 0x0014, 0x0012, 0x0019, 0x0023, 0x0040, 0x0044, 0x0032,
0x0039, 0x003b, 0x0070, 0x0084, 0x0072, 0x00f9, 0x00fb, 0x01f0,
0x01f4, 0x00f2, 0x0103, 0x0000, 0x0204, 0x0002, 0x0009, 0x0183,
0x0500, 0x0304, 0x0302, 0x0509, 0x0383, 0x0900, 0x0f04, 0x0702,
0x0003, 0x0f83, 0x0004, 0x0804, 0x1f02, 0x1789, 0x1783, 0x0f00,
0x2102, 0x0102, 0x1803, 0x3100, 0x1100, 0x3803, 0x6e89, 0x2e89,
0x4e89, 0x0e89, 0x7689, 0x3689, 0x5689, 0x1689, 0x6689, 0x2689,
0x4689, 0x0689, 0x7a89, 0x3a89, 0x5a89, 0x1a89, 0x6a89, 0x2a89,
0x4a89, 0x0a89, 0x7289, 0x3289, 0x5289, 0x1289, 0x6289, 0x2289,
0x4289, 0x0289, 0x7c89, 0x3c89, 0x5c89, 0x1c89, 0x6c89, 0x2c89,
0x4c89, 0x0c89, 0x7589, 0x3589, 0x6d89, 0x2d89, 0x7d89, 0x3d89,
0x6389, 0x2389, 0x7389, 0x3389, 0x2b89, 0x4b89, 0x6889, 0x3e89,
0x5e89, 0x0889, 0x7089, 0x3089, 0x5089, 0x1089, 0x6089, 0x2089,
0x4089, 0x0089, 0x7f09, 0x1b89, 0x6b89, 0x1f09, 0x6f09, 0x2f09,
0x4f09, 0x0f09, 0x3b89, 0x5b89, 0x5709, 0x1709, 0x7b89, 0x0789,
0x4789, 0x7709, 0x7b09, 0x3b09, 0x5b09, 0x1b09, 0x6b09, 0x2b09,
0x4b09, 0x3709, 0x4709, 0x3309, 0x0709, 0x0b09, 0x6309, 0x2309,
0x4309, 0x7309, 0x5309, 0x0309, 0x7102, 0x3102, 0x5804, 0x1309,
0x5489, 0x1e89, 0x7489, 0x1489, 0x6489, 0x7e89, 0x5589, 0x0189,
```

```
0x4189, 0x2189, 0x6189, 0x0d89, 0x4489, 0x2489, 0x4d89, 0x1189,
0x5189, 0x3189, 0x7189, 0x1d89, 0x7889, 0x0489, 0x5d89, 0x0989,
0x4989, 0x2989, 0x6989, 0x0389, 0x5889, 0x3889, 0x4389, 0x1989,
0x5989, 0x3989, 0x7989, 0x1389, 0x1889, 0x5389, 0x0589, 0x4589,
0x2589, 0x0b89, 0x4889, 0x2709, 0x6709, 0x2889, 0x3489, 0x5f09,
0x3f09, 0x6589, 0x1589, 0x5100, 0x1804, 0x7100, 0x2789, 0x3804,
0x1102, 0x2f00, 0x0783, 0x0803, 0x0f02, 0x1004, 0x1f00, 0x0100,
0x0f89, 0x0902, 0x0704, 0x0700, 0x0403, 0x0109, 0x0502, 0x0404,
0x0300, 0x0203, 0x0209, 0x0202, 0x0104, 0x0200, 0x0083, 0x01f2,
0x00f4, 0x00f0, 0x007b, 0x0079, 0x0082, 0x0074, 0x0080, 0x0043,
0x0049, 0x0042, 0x0034, 0x0030, 0x001b, 0x0029, 0x0022, 0x0024,
0x0020, 0x0013, 0x0001, 0x000c, 0x0018, 0x0007, 0x000d, 0x0006,
};

uint8 Code6Len[256] = {
 4,  4,  4,  4,  5,  5,  5,  5,  6,  6,  6,  6,  6,  7,  7,  7,
 7,  7,  8,  8,  8,  8,  8,  9,  9,  9,  9, 10, 10, 10, 10, 10,
11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 13, 13, 13, 13, 13, 14,
14, 14, 14, 15, 15, 14, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15,
15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 14,
14, 14, 13, 13, 13, 13, 13, 13, 12, 12, 12, 12, 11, 11, 11, 11,
11, 10, 10, 10, 10, 10,  9,  9,  9,  9,  8,  8,  8,  8,  8,  7,
 7,  7,  7,  7,  6,  6,  6,  6,  6,  5,  5,  5,  5,  4,  4,  4,
};

/* LOSSY -- dumped from ’tmspread10.code’ */
uint16 Code7Bits[256] = {
0x0005, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001, 0x0001,
0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001,
0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001,
0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001,
0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001,
0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001,
0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001, 0x0001,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d, 0x001d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d, 0x000d,
0x000d, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003, 0x0003,
0x0003, 0x0003, 0x0003, 0x0002, 0x0002, 0x0002, 0x0002, 0x0002,
};

uint8 Code7Len[256] = {
 4,  2,  2,  2,  2,  2,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
 3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
 3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
 3,  3,  3,  3,  3,  3,  3,  3,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,  5,
 5,  5,  5,  5,  5,  5,  5,  5,  5,  2,  2,  2,  2,  2,  2,  2,
 2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
 2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
 2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
};
```

```
/* dumped from 'tmspread10.requant' */
uint8 Code7Requant[256] = {
  0,   1,   1,   1,   1,   1,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,
 10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,
 10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,  10,
 10,  10,  10,  10,  10,  10,  10,  10, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156,
156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156,
156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156,
156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156, 156,
156, 156, 156, 156, 156, 156, 156, 156, 156, 246, 246, 246, 246, 246, 246, 246,
246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246,
246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246,
246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 246, 255, 255, 255, 255, 255,
};

/* dumped from 'default.requant' */
/* REFINE this could be calculated instead of put into PROM (obviously) */
uint8 CodeIdentRequant[256] = {
  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,  15,
 16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27,  28,  29,  30,  31,
 32,  33,  34,  35,  36,  37,  38,  39,  40,  41,  42,  43,  44,  45,  46,  47,
 48,  49,  50,  51,  52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,
 64,  65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,
 80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95,
 96,  97,  98,  99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127,
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159,
160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175,
176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191,
192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223,
224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255,
};
```

9. Appendix E: converting Huffman tables to tree form

```
typedef struct ht_node {
    int value;
    struct ht_node *zero, *one;
} Huffman_node;

Huffman_node *ht_insert(root, value, code, len)
Huffman_node *root;
int value, code, len;
{
    int bit;
    Huffman_node **branch;

    if(!root) {
     root = (Huffman_node *) malloc(sizeof(Huffman_node));
     root->zero = root->one = 0;
    }

    if(len == 0) {
     root->value = value;
    }
    else {
     bit = code&0x1;
     if(bit == 0) branch = &root->zero;
     else branch = &root->one;

     if(*branch == 0) {
```

```
        *branch = (Huffman_node *) malloc(sizeof(Huffman_node));
          (*branch)->zero = 0;
          (*branch)->one = 0;
     }
     ht_insert(*branch, value, code>>1, len-1);
    }
    return root;
}

Huffman_node *ht_tree_gen_predictive(i)
int i;
{
    Huffman_node *tree = 0;
    uint16 *code;
    uint8 *len;
    uint8 *requant;

    code = CodeBitsVec[i];
    len = CodeLenVec[i];
    requant = CodeRequantVec[i];

    tree = ht_insert(tree, requant[0], code[0], len[0]);

    for(i = 1; i < 128; i++) {
     if(requant[i] != requant[i-1])
       tree = ht_insert(tree, requant[i], code[i], len[i]);
    }

    tree = ht_insert(tree, requant[255], code[255], len[255]);

    for(i = 254; i >= 128; i--) {
     if(requant[i] != requant[i+1])
       tree = ht_insert(tree, requant[i], code[i], len[i]);
    }
    return tree;
}

Huffman_node *ht_tree_gen_transform(i)
int i;
{
    Huffman_node *tree = 0;
    uint32 *code;
    uint8 *len;
    int NCodes;

    code = CodeBitsVec[i];
    len = CodeLenVec[i];
    NCodes = CodeSizes[i];

    for(i = 0; i < NCodes; i++) {
     tree = ht_insert(tree, i, code[i], len[i]);
    }
    return tree;
```

}

10. Appendix F: global map crosstrack summing tables

These tables map output pixel in the global map to the number of input pixels summed and the offset of these pixels from the starting pixel. Using the EDIT_MODE_ID to determine the start pixel of the image, the mapping to and from detector pixel and global map pixel is determined.

The tables have four columns: the output pixel number starting from 0, the summing for that pixel, and the starting and ending hardware pixel offsets relative to EDIT_MODE_ID that are summed to form the output pixel.

7.5 km/pixel nominal resolution; output width 384.

0	1	0	0
1	1	1	1
2	1	2	2
3	1	3	3
4	1	4	4
5	1	5	5
6	1	6	6
7	1	7	7
8	1	8	8
9	1	9	9
10	1	10	10
11	1	11	11
12	1	12	12
13	1	13	13
14	1	14	14
15	1	15	15
16	1	16	16
17	1	17	17
18	1	18	18
19	1	19	19
20	1	20	20
21	1	21	21
22	1	22	22
23	1	23	23
24	1	24	24
25	1	25	25
26	1	26	26
27	1	27	27
28	1	28	28
29	1	29	29
30	1	30	30
31	1	31	31
32	1	32	32
33	1	33	33
34	1	34	34
35	1	35	35
36	1	36	36
37	1	37	37
38	1	38	38
39	1	39	39

40	1	40	40
41	1	41	41
42	1	42	42
43	1	43	43
44	1	44	44
45	1	45	45
46	1	46	46
47	1	47	47
48	1	48	48
49	2	49	50
50	2	51	52
51	2	53	54
52	2	55	56
53	2	57	58
54	2	59	60
55	2	61	62
56	2	63	64
57	2	65	66
58	2	67	68
59	2	69	70
60	2	71	72
61	2	73	74
62	2	75	76
63	2	77	78
64	2	79	80
65	2	81	82
66	2	83	84
67	2	85	86
68	2	87	88
69	2	89	90
70	2	91	92
71	2	93	94
72	3	95	97
73	3	98	100
74	3	101	103
75	3	104	106
76	3	107	109
77	3	110	112
78	3	113	115
79	3	116	118
80	3	119	121
81	3	122	124
82	3	125	127
83	3	128	130
84	3	131	133
85	3	134	136
86	3	137	139
87	3	140	142
88	4	143	146
89	4	147	150
90	4	151	154
91	4	155	158
92	4	159	162

93	4	163	166
94	4	167	170
95	4	171	174
96	4	175	178
97	4	179	182
98	4	183	186
99	4	187	190
100	5	191	195
101	5	196	200
102	5	201	205
103	5	206	210
104	5	211	215
105	5	216	220
106	5	221	225
107	5	226	230
108	5	231	235
109	6	236	241
110	6	242	247
111	6	248	253
112	6	254	259
113	7	260	266
114	7	267	273
115	7	274	280
116	7	281	287
117	7	288	294
118	7	295	301
119	7	302	308
120	8	309	316
121	8	317	324
122	8	325	332
123	8	333	340
124	8	341	348
125	8	349	356
126	9	357	365
127	9	366	374
128	9	375	383
129	9	384	392
130	9	393	401
131	10	402	411
132	10	412	421
133	10	422	431
134	10	432	441
135	10	442	451
136	11	452	462
137	11	463	473
138	11	474	484
139	11	485	495
140	12	496	507
141	13	508	520
142	13	521	533
143	13	534	546
144	13	547	559
145	14	560	573

146	14	574	587
147	14	588	601
148	14	602	615
149	15	616	630
150	15	631	645
151	15	646	660
152	16	661	676
153	16	677	692
154	16	693	708
155	17	709	725
156	17	726	742
157	18	743	760
158	18	761	778
159	19	779	797
160	19	798	816
161	19	817	835
162	20	836	855
163	20	856	875
164	20	876	895
165	21	896	916
166	21	917	937
167	21	938	958
168	22	959	980
169	22	981	1002
170	23	1003	1025
171	24	1026	1049
172	24	1050	1073
173	24	1074	1097
174	25	1098	1122
175	25	1123	1147
176	25	1148	1172
177	25	1173	1197
178	26	1198	1223
179	26	1224	1249
180	26	1250	1275
181	26	1276	1301
182	26	1302	1327
183	27	1328	1354
184	27	1355	1381
185	27	1382	1408
186	27	1409	1435
187	27	1436	1462
188	27	1463	1489
189	27	1490	1516
190	27	1517	1543
191	27	1544	1570
192	27	1571	1597
193	27	1598	1624
194	27	1625	1651
195	27	1652	1678
196	27	1679	1705
197	27	1706	1732
198	27	1733	1759

199	27	1760	1786
200	27	1787	1813
201	26	1814	1839
202	26	1840	1865
203	26	1866	1891
204	26	1892	1917
205	26	1918	1943
206	25	1944	1968
207	25	1969	1993
208	25	1994	2018
209	25	2019	2043
210	24	2044	2067
211	24	2068	2091
212	24	2092	2115
213	23	2116	2138
214	22	2139	2160
215	22	2161	2182
216	21	2183	2203
217	21	2204	2224
218	21	2225	2245
219	20	2246	2265
220	20	2266	2285
221	20	2286	2305
222	19	2306	2324
223	19	2325	2343
224	19	2344	2362
225	18	2363	2380
226	18	2381	2398
227	17	2399	2415
228	17	2416	2432
229	16	2433	2448
230	16	2449	2464
231	16	2465	2480
232	15	2481	2495
233	15	2496	2510
234	15	2511	2525
235	14	2526	2539
236	14	2540	2553
237	14	2554	2567
238	14	2568	2581
239	13	2582	2594
240	13	2595	2607
241	13	2608	2620
242	13	2621	2633
243	12	2634	2645
244	11	2646	2656
245	11	2657	2667
246	11	2668	2678
247	11	2679	2689
248	10	2690	2699
249	10	2700	2709
250	10	2710	2719
251	10	2720	2729

252	10	2730	2739
253	9	2740	2748
254	9	2749	2757
255	9	2758	2766
256	9	2767	2775
257	9	2776	2784
258	8	2785	2792
259	8	2793	2800
260	8	2801	2808
261	8	2809	2816
262	8	2817	2824
263	8	2825	2832
264	7	2833	2839
265	7	2840	2846
266	7	2847	2853
267	7	2854	2860
268	7	2861	2867
269	7	2868	2874
270	7	2875	2881
271	6	2882	2887
272	6	2888	2893
273	6	2894	2899
274	6	2900	2905
275	5	2906	2910
276	5	2911	2915
277	5	2916	2920
278	5	2921	2925
279	5	2926	2930
280	5	2931	2935
281	5	2936	2940
282	5	2941	2945
283	5	2946	2950
284	4	2951	2954
285	4	2955	2958
286	4	2959	2962
287	4	2963	2966
288	4	2967	2970
289	4	2971	2974
290	4	2975	2978
291	4	2979	2982
292	4	2983	2986
293	4	2987	2990
294	4	2991	2994
295	4	2995	2998
296	3	2999	3001
297	3	3002	3004
298	3	3005	3007
299	3	3008	3010
300	3	3011	3013
301	3	3014	3016
302	3	3017	3019
303	3	3020	3022
304	3	3023	3025

305	3	3026	3028
306	3	3029	3031
307	3	3032	3034
308	3	3035	3037
309	3	3038	3040
310	3	3041	3043
311	3	3044	3046
312	2	3047	3048
313	2	3049	3050
314	2	3051	3052
315	2	3053	3054
316	2	3055	3056
317	2	3057	3058
318	2	3059	3060
319	2	3061	3062
320	2	3063	3064
321	2	3065	3066
322	2	3067	3068
323	2	3069	3070
324	2	3071	3072
325	2	3073	3074
326	2	3075	3076
327	2	3077	3078
328	2	3079	3080
329	2	3081	3082
330	2	3083	3084
331	2	3085	3086
332	2	3087	3088
333	2	3089	3090
334	2	3091	3092
335	1	3093	3093
336	1	3094	3094
337	1	3095	3095
338	1	3096	3096
339	1	3097	3097
340	1	3098	3098
341	1	3099	3099
342	1	3100	3100
343	1	3101	3101
344	1	3102	3102
345	1	3103	3103
346	1	3104	3104
347	1	3105	3105
348	1	3106	3106
349	1	3107	3107
350	1	3108	3108
351	1	3109	3109
352	1	3110	3110
353	1	3111	3111
354	1	3112	3112
355	1	3113	3113
356	1	3114	3114
357	1	3115	3115

358	1	3116	3116
359	1	3117	3117
360	1	3118	3118
361	1	3119	3119
362	1	3120	3120
363	1	3121	3121
364	1	3122	3122
365	1	3123	3123
366	1	3124	3124
367	1	3125	3125
368	1	3126	3126
369	1	3127	3127
370	1	3128	3128
371	1	3129	3129
372	1	3130	3130
373	1	3131	3131
374	1	3132	3132
375	1	3133	3133
376	1	3134	3134
377	1	3135	3135
378	1	3136	3136
379	1	3137	3137
380	1	3138	3138
381	1	3139	3139
382	1	3140	3140
383	1	3141	3141

3.75 km/pixel nominal resolution; output width 768.

0	1	0	0
1	1	1	1
2	1	2	2
3	1	3	3
4	1	4	4
5	1	5	5
6	1	6	6
7	1	7	7
8	1	8	8
9	1	9	9
10	1	10	10
11	1	11	11
12	1	12	12
13	1	13	13
14	1	14	14
15	1	15	15
16	1	16	16
17	1	17	17
18	1	18	18
19	1	19	19
20	1	20	20
21	1	21	21
22	1	22	22
23	1	23	23

24	1	24	24
25	1	25	25
26	1	26	26
27	1	27	27
28	1	28	28
29	1	29	29
30	1	30	30
31	1	31	31
32	1	32	32
33	1	33	33
34	1	34	34
35	1	35	35
36	1	36	36
37	1	37	37
38	1	38	38
39	1	39	39
40	1	40	40
41	1	41	41
42	1	42	42
43	1	43	43
44	1	44	44
45	1	45	45
46	1	46	46
47	1	47	47
48	1	48	48
49	1	49	49
50	1	50	50
51	1	51	51
52	1	52	52
53	1	53	53
54	1	54	54
55	1	55	55
56	1	56	56
57	1	57	57
58	1	58	58
59	1	59	59
60	1	60	60
61	1	61	61
62	1	62	62
63	1	63	63
64	1	64	64
65	1	65	65
66	1	66	66
67	1	67	67
68	1	68	68
69	1	69	69
70	1	70	70
71	1	71	71
72	1	72	72
73	1	73	73
74	1	74	74
75	1	75	75
76	1	76	76

77	1	77	77
78	1	78	78
79	1	79	79
80	1	80	80
81	1	81	81
82	1	82	82
83	1	83	83
84	1	84	84
85	1	85	85
86	1	86	86
87	1	87	87
88	1	88	88
89	1	89	89
90	1	90	90
91	1	91	91
92	1	92	92
93	1	93	93
94	1	94	94
95	1	95	95
96	1	96	96
97	1	97	97
98	1	98	98
99	1	99	99
100	1	100	100
101	1	101	101
102	1	102	102
103	1	103	103
104	1	104	104
105	1	105	105
106	1	106	106
107	1	107	107
108	1	108	108
109	1	109	109
110	1	110	110
111	1	111	111
112	1	112	112
113	1	113	113
114	1	114	114
115	1	115	115
116	1	116	116
117	1	117	117
118	1	118	118
119	1	119	119
120	1	120	120
121	1	121	121
122	1	122	122
123	1	123	123
124	1	124	124
125	1	125	125
126	1	126	126
127	1	127	127
128	1	128	128
129	1	129	129

130	1	130	130
131	1	131	131
132	1	132	132
133	1	133	133
134	1	134	134
135	1	135	135
136	1	136	136
137	1	137	137
138	1	138	138
139	1	139	139
140	1	140	140
141	1	141	141
142	1	142	142
143	1	143	143
144	1	144	144
145	1	145	145
146	1	146	146
147	1	147	147
148	1	148	148
149	1	149	149
150	1	150	150
151	1	151	151
152	1	152	152
153	1	153	153
154	1	154	154
155	1	155	155
156	1	156	156
157	1	157	157
158	1	158	158
159	1	159	159
160	1	160	160
161	1	161	161
162	1	162	162
163	1	163	163
164	2	164	165
165	2	166	167
166	2	168	169
167	2	170	171
168	2	172	173
169	2	174	175
170	2	176	177
171	2	178	179
172	2	180	181
173	2	182	183
174	2	184	185
175	2	186	187
176	2	188	189
177	2	190	191
178	2	192	193
179	2	194	195
180	2	196	197
181	2	198	199
182	2	200	201

183	2	202	203
184	2	204	205
185	2	206	207
186	2	208	209
187	2	210	211
188	2	212	213
189	2	214	215
190	2	216	217
191	2	218	219
192	2	220	221
193	2	222	223
194	2	224	225
195	2	226	227
196	2	228	229
197	2	230	231
198	2	232	233
199	2	234	235
200	2	236	237
201	2	238	239
202	2	240	241
203	2	242	243
204	2	244	245
205	2	246	247
206	2	248	249
207	2	250	251
208	2	252	253
209	2	254	255
210	2	256	257
211	3	258	260
212	3	261	263
213	3	264	266
214	3	267	269
215	3	270	272
216	3	273	275
217	3	276	278
218	3	279	281
219	3	282	284
220	3	285	287
221	3	288	290
222	3	291	293
223	3	294	296
224	3	297	299
225	3	300	302
226	3	303	305
227	3	306	308
228	3	309	311
229	3	312	314
230	3	315	317
231	3	318	320
232	3	321	323
233	3	324	326
234	4	327	330
235	4	331	334

236	4	335	338
237	4	339	342
238	4	343	346
239	4	347	350
240	4	351	354
241	4	355	358
242	4	359	362
243	4	363	366
244	4	367	370
245	4	371	374
246	4	375	378
247	4	379	382
248	4	383	386
249	4	387	390
250	4	391	394
251	4	395	398
252	4	399	402
253	4	403	406
254	4	407	410
255	4	411	414
256	4	415	418
257	4	419	422
258	5	423	427
259	5	428	432
260	5	433	437
261	5	438	442
262	5	443	447
263	5	448	452
264	5	453	457
265	5	458	462
266	5	463	467
267	5	468	472
268	5	473	477
269	5	478	482
270	5	483	487
271	5	488	492
272	5	493	497
273	5	498	502
274	5	503	507
275	5	508	512
276	6	513	518
277	6	519	524
278	6	525	530
279	6	531	536
280	6	537	542
281	6	543	548
282	6	549	554
283	6	555	560
284	6	561	566
285	6	567	572
286	6	573	578
287	6	579	584
288	7	585	591

289	7	592	598
290	7	599	605
291	7	606	612
292	7	613	619
293	7	620	626
294	7	627	633
295	7	634	640
296	7	641	647
297	7	648	654
298	7	655	661
299	7	662	668
300	7	669	675
301	7	676	682
302	8	683	690
303	8	691	698
304	8	699	706
305	8	707	714
306	8	715	722
307	8	723	730
308	8	731	738
309	8	739	746
310	8	747	754
311	8	755	762
312	8	763	770
313	9	771	779
314	9	780	788
315	9	789	797
316	9	798	806
317	9	807	815
318	9	816	824
319	9	825	833
320	9	834	842
321	9	843	851
322	9	852	860
323	10	861	870
324	10	871	880
325	10	881	890
326	10	891	900
327	10	901	910
328	10	911	920
329	10	921	930
330	10	931	940
331	10	941	950
332	10	951	960
333	10	961	970
334	10	971	980
335	11	981	991
336	11	992	1002
337	11	1003	1013
338	11	1014	1024
339	11	1025	1035
340	11	1036	1046
341	12	1047	1058

342	12	1059	1070
343	12	1071	1082
344	12	1083	1094
345	12	1095	1106
346	12	1107	1118
347	12	1119	1130
348	12	1131	1142
349	12	1143	1154
350	12	1155	1166
351	12	1167	1178
352	12	1179	1190
353	12	1191	1202
354	12	1203	1214
355	12	1215	1226
356	13	1227	1239
357	13	1240	1252
358	13	1253	1265
359	13	1266	1278
360	13	1279	1291
361	13	1292	1304
362	13	1305	1317
363	13	1318	1330
364	13	1331	1343
365	13	1344	1356
366	13	1357	1369
367	13	1370	1382
368	13	1383	1395
369	13	1396	1408
370	13	1409	1421
371	13	1422	1434
372	13	1435	1447
373	13	1448	1460
374	13	1461	1473
375	13	1474	1486
376	13	1487	1499
377	13	1500	1512
378	13	1513	1525
379	13	1526	1538
380	13	1539	1551
381	13	1552	1564
382	13	1565	1577
383	13	1578	1590
384	13	1591	1603
385	13	1604	1616
386	13	1617	1629
387	13	1630	1642
388	13	1643	1655
389	13	1656	1668
390	13	1669	1681
391	13	1682	1694
392	13	1695	1707
393	13	1708	1720
394	13	1721	1733

395	13	1734	1746
396	13	1747	1759
397	13	1760	1772
398	13	1773	1785
399	13	1786	1798
400	13	1799	1811
401	13	1812	1824
402	13	1825	1837
403	13	1838	1850
404	13	1851	1863
405	13	1864	1876
406	13	1877	1889
407	13	1890	1902
408	13	1903	1915
409	13	1916	1928
410	13	1929	1941
411	13	1942	1954
412	12	1955	1966
413	12	1967	1978
414	12	1979	1990
415	12	1991	2002
416	12	2003	2014
417	12	2015	2026
418	12	2027	2038
419	12	2039	2050
420	12	2051	2062
421	12	2063	2074
422	12	2075	2086
423	12	2087	2098
424	12	2099	2110
425	12	2111	2122
426	12	2123	2134
427	11	2135	2145
428	11	2146	2156
429	11	2157	2167
430	11	2168	2178
431	11	2179	2189
432	11	2190	2200
433	10	2201	2210
434	10	2211	2220
435	10	2221	2230
436	10	2231	2240
437	10	2241	2250
438	10	2251	2260
439	10	2261	2270
440	10	2271	2280
441	10	2281	2290
442	10	2291	2300
443	10	2301	2310
444	10	2311	2320
445	9	2321	2329
446	9	2330	2338
447	9	2339	2347

448	9	2348	2356
449	9	2357	2365
450	9	2366	2374
451	9	2375	2383
452	9	2384	2392
453	9	2393	2401
454	9	2402	2410
455	8	2411	2418
456	8	2419	2426
457	8	2427	2434
458	8	2435	2442
459	8	2443	2450
460	8	2451	2458
461	8	2459	2466
462	8	2467	2474
463	8	2475	2482
464	8	2483	2490
465	8	2491	2498
466	7	2499	2505
467	7	2506	2512
468	7	2513	2519
469	7	2520	2526
470	7	2527	2533
471	7	2534	2540
472	7	2541	2547
473	7	2548	2554
474	7	2555	2561
475	7	2562	2568
476	7	2569	2575
477	7	2576	2582
478	7	2583	2589
479	7	2590	2596
480	6	2597	2602
481	6	2603	2608
482	6	2609	2614
483	6	2615	2620
484	6	2621	2626
485	6	2627	2632
486	6	2633	2638
487	6	2639	2644
488	6	2645	2650
489	6	2651	2656
490	6	2657	2662
491	6	2663	2668
492	5	2669	2673
493	5	2674	2678
494	5	2679	2683
495	5	2684	2688
496	5	2689	2693
497	5	2694	2698
498	5	2699	2703
499	5	2704	2708
500	5	2709	2713

501	5	2714	2718
502	5	2719	2723
503	5	2724	2728
504	5	2729	2733
505	5	2734	2738
506	5	2739	2743
507	5	2744	2748
508	5	2749	2753
509	5	2754	2758
510	4	2759	2762
511	4	2763	2766
512	4	2767	2770
513	4	2771	2774
514	4	2775	2778
515	4	2779	2782
516	4	2783	2786
517	4	2787	2790
518	4	2791	2794
519	4	2795	2798
520	4	2799	2802
521	4	2803	2806
522	4	2807	2810
523	4	2811	2814
524	4	2815	2818
525	4	2819	2822
526	4	2823	2826
527	4	2827	2830
528	4	2831	2834
529	4	2835	2838
530	4	2839	2842
531	4	2843	2846
532	4	2847	2850
533	4	2851	2854
534	3	2855	2857
535	3	2858	2860
536	3	2861	2863
537	3	2864	2866
538	3	2867	2869
539	3	2870	2872
540	3	2873	2875
541	3	2876	2878
542	3	2879	2881
543	3	2882	2884
544	3	2885	2887
545	3	2888	2890
546	3	2891	2893
547	3	2894	2896
548	3	2897	2899
549	3	2900	2902
550	3	2903	2905
551	3	2906	2908
552	3	2909	2911
553	3	2912	2914

554	3	2915	2917
555	3	2918	2920
556	3	2921	2923
557	2	2924	2925
558	2	2926	2927
559	2	2928	2929
560	2	2930	2931
561	2	2932	2933
562	2	2934	2935
563	2	2936	2937
564	2	2938	2939
565	2	2940	2941
566	2	2942	2943
567	2	2944	2945
568	2	2946	2947
569	2	2948	2949
570	2	2950	2951
571	2	2952	2953
572	2	2954	2955
573	2	2956	2957
574	2	2958	2959
575	2	2960	2961
576	2	2962	2963
577	2	2964	2965
578	2	2966	2967
579	2	2968	2969
580	2	2970	2971
581	2	2972	2973
582	2	2974	2975
583	2	2976	2977
584	2	2978	2979
585	2	2980	2981
586	2	2982	2983
587	2	2984	2985
588	2	2986	2987
589	2	2988	2989
590	2	2990	2991
591	2	2992	2993
592	2	2994	2995
593	2	2996	2997
594	2	2998	2999
595	2	3000	3001
596	2	3002	3003
597	2	3004	3005
598	2	3006	3007
599	2	3008	3009
600	2	3010	3011
601	2	3012	3013
602	2	3014	3015
603	2	3016	3017
604	1	3018	3018
605	1	3019	3019
606	1	3020	3020

607	1	3021	3021
608	1	3022	3022
609	1	3023	3023
610	1	3024	3024
611	1	3025	3025
612	1	3026	3026
613	1	3027	3027
614	1	3028	3028
615	1	3029	3029
616	1	3030	3030
617	1	3031	3031
618	1	3032	3032
619	1	3033	3033
620	1	3034	3034
621	1	3035	3035
622	1	3036	3036
623	1	3037	3037
624	1	3038	3038
625	1	3039	3039
626	1	3040	3040
627	1	3041	3041
628	1	3042	3042
629	1	3043	3043
630	1	3044	3044
631	1	3045	3045
632	1	3046	3046
633	1	3047	3047
634	1	3048	3048
635	1	3049	3049
636	1	3050	3050
637	1	3051	3051
638	1	3052	3052
639	1	3053	3053
640	1	3054	3054
641	1	3055	3055
642	1	3056	3056
643	1	3057	3057
644	1	3058	3058
645	1	3059	3059
646	1	3060	3060
647	1	3061	3061
648	1	3062	3062
649	1	3063	3063
650	1	3064	3064
651	1	3065	3065
652	1	3066	3066
653	1	3067	3067
654	1	3068	3068
655	1	3069	3069
656	1	3070	3070
657	1	3071	3071
658	1	3072	3072
659	1	3073	3073

660	1	3074	3074
661	1	3075	3075
662	1	3076	3076
663	1	3077	3077
664	1	3078	3078
665	1	3079	3079
666	1	3080	3080
667	1	3081	3081
668	1	3082	3082
669	1	3083	3083
670	1	3084	3084
671	1	3085	3085
672	1	3086	3086
673	1	3087	3087
674	1	3088	3088
675	1	3089	3089
676	1	3090	3090
677	1	3091	3091
678	1	3092	3092
679	1	3093	3093
680	1	3094	3094
681	1	3095	3095
682	1	3096	3096
683	1	3097	3097
684	1	3098	3098
685	1	3099	3099
686	1	3100	3100
687	1	3101	3101
688	1	3102	3102
689	1	3103	3103
690	1	3104	3104
691	1	3105	3105
692	1	3106	3106
693	1	3107	3107
694	1	3108	3108
695	1	3109	3109
696	1	3110	3110
697	1	3111	3111
698	1	3112	3112
699	1	3113	3113
700	1	3114	3114
701	1	3115	3115
702	1	3116	3116
703	1	3117	3117
704	1	3118	3118
705	1	3119	3119
706	1	3120	3120
707	1	3121	3121
708	1	3122	3122
709	1	3123	3123
710	1	3124	3124
711	1	3125	3125
712	1	3126	3126

713	1	3127	3127
714	1	3128	3128
715	1	3129	3129
716	1	3130	3130
717	1	3131	3131
718	1	3132	3132
719	1	3133	3133
720	1	3134	3134
721	1	3135	3135
722	1	3136	3136
723	1	3137	3137
724	1	3138	3138
725	1	3139	3139
726	1	3140	3140
727	1	3141	3141
728	1	3142	3142
729	1	3143	3143
730	1	3144	3144
731	1	3145	3145
732	1	3146	3146
733	1	3147	3147
734	1	3148	3148
735	1	3149	3149
736	1	3150	3150
737	1	3151	3151
738	1	3152	3152
739	1	3153	3153
740	1	3154	3154
741	1	3155	3155
742	1	3156	3156
743	1	3157	3157
744	1	3158	3158
745	1	3159	3159
746	1	3160	3160
747	1	3161	3161
748	1	3162	3162
749	1	3163	3163
750	1	3164	3164
751	1	3165	3165
752	1	3166	3166
753	1	3167	3167
754	1	3168	3168
755	1	3169	3169
756	1	3170	3170
757	1	3171	3171
758	1	3172	3172
759	1	3173	3173
760	1	3174	3174
761	1	3175	3175
762	1	3176	3176
763	1	3177	3177
764	1	3178	3178
765	1	3179	3179

766	1	3180	3180
767	1	3181	3181

Contents

1. Introduction 1
1.1. Purpose 1
1.2. Scope 1
1.3. Applicable Documents 1
1.4. Functional Description 1
1.4.1. Data Content Summary 1
1.4.2. Source and Transfer Method 1
1.4.3. Recipients and Utilization 1
1.4.4. Pertinent Relationships with Other Interfaces 2
1.5. Assumptions and Constraints 2
2. Environment 2
2.1. Hardware Characteristics and Limitations 2
2.1.1. Special Equipment and Device Interfaces 2
2.1.2. Special Setup Requirements 2
2.2. Interface Medium Characteristics 2
2.3. Failure Protection, Detection, and Recovery 2
2.4. End-of-File Conventions 2
3. Access 2
3.1. Access Tools 2
3.2. Input/Output Protocols 2
3.3. Timing and Sequencing Characteristics 2
4. Detailed Interface Specifications 2
4.1. Labeling and Identification 2
4.2. Structure and Organization Overview 3
4.3. Substructure Definition and Format 4
4.3.1. Header/Trailer Description Details 6
4.3.2. Data Description Details 6
4.3.3. Data loss considerations 9
4.4. Volume, Size, and Frequency Estimates 10
5. Appendix A: radial order translation table 10
6. Appendix B: transform coefficient Huffman code tables 10
7. Appendix C: inverse DCT and WHT algorithms 16
8. Appendix D: predictive Huffman code tables 21
9. Appendix E: converting Huffman tables to tree form 27
10. Appendix F: global map crosstrack summing tables 29