INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND ASSOCIATED AUDIO

ISO/IEC JTC1/SC29/WG11 N0808

11 November 1994

Draft Technical Report

INFORMATION TECHNOLOGY -

GENERIC CODING OF MOVING PICTURES AND ASSOCIATED AUDIO
Recommendation H.262

ISO/IEC 11172-5

Draft Technical Report

FOREWORD PAGE PROVIDED BY ISO.

ISO (THE INTERNATIONAL ORGANISATION FOR STANDARDISATION) AND IEC (THE INTERNATIONAL ELECTROTECHNICAL COMMISSION) FORM THE SPECIALISED SYSTEM FOR WORLD-WIDE STANDARDISATION. NATIONAL BODIES THAT ARE MEMBERS OF ISO AND IEC PARTICIPATE IN THE DEVELOPMENT OF INTERNATIONAL STANDARDS THROUGH TECHNICAL COMMITTEES ESTABLISHED BY THE RESPECTIVE ORGANISATION TO DEAL WITH PARTICULAR FIELDS OF TECHNICAL ACTIVITY. ISO AND IEC TECHNICAL COMMITTEES COLLABORATE IN FIELDS OF MUTUAL INTEREST. OTHER INTERNATIONAL ORGANISATIONS, GOVERNMENTAL AND NON-GOVERNMENTAL, IN LIAISON WITH ISO AND IEC, ALSO TAKE PART IN THE WORK.

IN THE FIELD OF INFORMATION TECHNOLOGY, ISO AND IEC HAVE ESTABLISHED A JOINT TECHNICAL COMMITTEE, ISO/IEC JTC1. DRAFT INTERNATIONAL STANDARDS ADOPTED BY THE JOINT TECHNICAL COMMITTEE ARE CIRCULATED TO NATIONAL BODIES FOR VOTING. PUBLICATION AS AN INTERNATIONAL STANDARD REQUIRES APPROVAL BY AT LEAST 75% OF THE NATIONAL BODIES CASTING A VOTE.

ISO/IEC 11172 WAS PREPARED BY ISO/IEC JTC1/SC29/WG11 ALSO KNOWN AS MPEG (MOVING PICTURES EXPERT GROUP). MPEG WAS FORMED IN 1988 TO ESTABLISH AN INTERNATIONAL STANDARD FOR THE CODED REPRESENTATION OF MOVING PICTURES AND ASSOCIATED AUDIO STORED ON DIGITAL STORAGE MEDIA. PARTS 1, 2 AND 3 OF ISO/IEC 11172 WERE UNANIMOUSLY APPROVED BY THE PARTICIPATING NATIONAL BODIES IN NOVEMBER 1992.

ISO/IEC 11172 IS PUBLISHED IN FIVE PARTS. PART 1 - SYSTEMS - SPECIFIES THE SYSTEM CODING LAYER OF THE STANDARD. IT DEFINES A MULTIPLEXED STRUCTURE FOR COMBINING AUDIO AND VIDEO DATA AND MEANS OF REPRESENTING THE TIMING INFORMATION NEEDED TO REPLAY SYNCHRONIZED SEQUENCES IN REAL-TIME. PART 2 - VIDEO - SPECIFIES THE CODED REPRESENTATION OF VIDEO DATA AND THE DECODING PROCESS REQUIRED TO RECONSTRUCT PICTURES. PART 3 - AUDIO - SPECIFIES THE CODED REPRESENTATION OF AUDIO DATA AND THE DECODING PROCESS REQUIRED TO RECONSTRUCT AUDIO SIGNALS. PART 4 - COMPLIANCE TESTING - SPECIFIES THE PROCEDURES FOR DETERMINING THE CHARACTERISTICS OF CODED BITSTREAMS AND THE DECODING PROCESS AND FOR TESTING COMPLIANCE WITH THE REQUIREMENTS STATED IN PARTS 1, 2 AND 3. PART 5 -TECHNICAL REPORT -- PROVIDES A SOFTWARE EXAMPLE OF AN ENCODER AND DECODER IMPLEMENTATION FOR ALL THREE PARTS.

ISO/IEC 11172-1 ALL ANNEXES ARE INFORMATIVE AND CONTAIN NO NORMATIVE REQUIREMENTS.

ISO/IEC 11172-2 ANNEX A, ANNEX B AND ANNEX C CONTAIN NORMATIVE REQUIREMENTS AND ARE AN INTEGRAL PART OF THIS STANDARD. ANNEX D AND ANNEX E ARE INFORMATIVE AND CONTAIN NO NORMATIVE REQUIREMENTS.

ISO/IEC 11172-3 ANNEX A AND ANNEX B CONTAIN NORMATIVE REQUIREMENTS AND ARE AN INTEGRAL PART OF THIS STANDARD. ALL OTHER ANNEXES ARE INFORMATIVE AND CONTAIN NO NORMATIVE REQUIREMENTS.

PART 4 OF ISO/IEC 11172 IS STILL IN PREPARATION.

PART 5 OF ISO/IEC 11172 IS STILL IN PREPARATION.

INFORMATION TECHNOLOGY -- CODING OF MOVING PICTURES AND ASSOCIATED AUDIO FOR DIGITAL STORAGE MEDIA AT UP TO ABOUT 1,5 MBIT/S

DTR 11172-5: SOFTWARE SIMULATION

Section 1: General

1.1 Scope

This technical report describes the software simulation of ISO/IEC 11172 Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s-- Part 1 (Systems), Part 2 (Video), Part 3 (Audio). The encoder and decoder examples are not intended for speed, rather their primary goal is to be understood by novices to the MPEG standard.

1.2 Normative references

The following International Standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 11172. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 11172 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 11172-1:1993 *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 1: Systems.*

ISO/IEC 11172-3:1993 *Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3 Audio.*

CCIR Recommendation 601-2 *Encoding parameters of digital television for studios*.

CCIR Report 624-4 *Characteristics of systems for monochrome and colour television.*

CCIR Recommendation 648 *Recording of audio signals.*

CCIR Report 955-2 *Sound broadcasting by satellite for portable and mobile receivers, including Annex IV Summary description of Advanced Digital System II.*

CCITT Recommendation J.17 *Pre-emphasis used on Sound-Programme Circuits.*

IEEE Draft Standard P1180/D2 1990 *Specification for the implementation of 8x 8 inverse discrete cosine transform".*

IEC publication 908:1987 *CD Digital Audio System.*

Section 2: Technical elements

2.1 Definitions

For the purposes of ISO/IEC 11172, the following definitions apply. If specific to a part, this is noted in square brackets.

2.1.1 ac coefficient [video]: Any DCT coefficient for which the frequency in one or both dimensions is non-zero.

2.1.2 access unit [system]: In the case of compressed audio an access unit is an audio access unit. In the case of compressed video an access unit is the coded representation of a picture.

2.1.3 adaptive segmentation [audio]: A subdivision of the digital representation of an audio signal in variable segments of time.

2.1.4 adaptive bit allocation [audio]: The assignment of bits to subbands in a time and frequency varying fashion according to a psychoacoustic model.

2.1.5 adaptive noise allocation [audio]: The assignment of coding noise to frequency bands in a time and frequency varying fashion according to a psychoacoustic model.

2.1.6 alias [audio]: Mirrored signal component resulting from sub-Nyquist sampling.

2.1.7 analysis filterbank [audio]: Filterbank in the encoder that transforms a broadband PCM audio signal into a set of subsampled subband samples.

2.1.8 audio access unit [audio]: For Layers I and II an audio access unit is defined as the smallest part of the encoded bitstream which can be decoded by itself, where decoded means "fully reconstructed sound". For Layer III an audio access unit is part of the bitstream that is decodable with the use of previously acquired main information.

2.1.9 audio buffer [audio]: A buffer in the system target decoder for storage of compressed audio data.

2.1.10 audio sequence [audio]: A non-interrupted series of audio frames in which the following parameters are not changed:

- ID
- Layer
- Sampling Frequency
- For Layer I and II: Bitrate index

2.1.11 backward motion vector [video]: A motion vector that is used for motion compensation from a reference picture at a later time in display order.

2.1.12 Bark [audio]: Unit of critical band rate. The Bark scale is a non-linear mapping of the frequency scale over the audio range closely corresponding with the frequency selectivity of the human ear across the band.

2.1.13 bidirectionally predictive-coded picture; B-picture [video]: A picture that is coded using motion compensated prediction from a past and/or future reference picture.

2.1.14 bitrate: The rate at which the compressed bitstream is delivered from the storage medium to the input of a decoder.

2.1.15 block companding [audio]: Normalizing of the digital representation of an audio signal within a certain time period.

2.1.16 block [video]: An 8-row by 8-column orthogonal block of pels.

2.1.17 bound [audio]: The lowest subband in which intensity stereo coding is used.

2.1.18 byte aligned: A bit in a coded bitstream is byte-aligned if its position is a multiple of 8-bits from the first bit in the stream.

2.1.19 byte: Sequence of 8-bits.

2.1.20 channel: A digital medium that stores or transports an ISO/IEC 11172 stream.

2.1.21 channel [audio]: The left and right channels of a stereo signal

2.1.22 chrominance (component) [video]: A matrix, block or single pel representing one of the two colour difference signals related to the primary colours in the manner defined in CCIR Rec 601. The symbols used for the colour difference signals are Cr and Cb.

2.1.23 coded audio bitstream [audio]: A coded representation of an audio signal as specified in ISO/IEC 11172-3.

2.1.24 coded video bitstream [video]: A coded representation of a series of one or more pictures as specified in this part of ISO/IEC 11172.

2.1.25 coded order [video]: The order in which the pictures are stored and decoded. This order is not necessarily the same as the display order.

2.1.26 coded representation: A data element as represented in its encoded form.

2.1.27 coding parameters [video]: The set of user-definable parameters that characterize a coded video bitstream. Bitstreams are characterised by coding parameters. Decoders are characterised by the bitstreams that they are capable of decoding.

2.1.28 component [video]: A matrix, block or single pel from one of the three matrices (luminance and two chrominance) that make up a picture.

2.1.29 compression: Reduction in the number of bits used to represent an item of data.

2.1.30 constant bitrate coded video [video]: A compressed video bitstream with a constant average bitrate.

2.1.31 constant bitrate: Operation where the bitrate is constant from start to finish of the compressed bitstream.

2.1.32 constrained parameters [video]: The values of the set of coding parameters defined in 2.4.3.2.

2.1.33 constrained system parameter stream (CSPS) [system]: An ISO/IEC 11172 multiplexed stream for which the constraints defined in 2.4.6 of ISO/IEC 11172-1 apply.

2.1.34 CRC: Cyclic redundancy code.

2.1.35 critical band rate [audio]: Psychoacoustic function of frequency. At a given audible frequency it is proportional to the number of critical bands below that frequency. The units of the critical band rate scale are Barks.

2.1.36 critical band [audio]: Psychoacoustic measure in the spectral domain which corresponds to the frequency selectivity of the human ear. This selectivity is expressed in Bark.

2.1.37 data element: An item of data as represented before encoding and after decoding.

2.1.38 dc-coefficient [video]: The DCT coefficient for which the frequency is zero in both dimensions.

2.1.39 dc-coded picture; D-picture [video]: A picture that is coded using only information from itself. Of the DCT coefficients in the coded representation, only the dc-coefficients are present.

2.1.40 DCT coefficient: The amplitude of a specific cosine basis function.

2.1.41 decoded stream: The decoded reconstruction of a compressed bitstream.

2.1.42 decoder input buffer [video]: The first-in first-out (FIFO) buffer specified in the video buffering verifier.

2.1.43 decoder input rate [video]: The data rate specified in the video buffering verifier and encoded in the coded video bitstream.

2.1.44 decoder: An embodiment of a decoding process.

2.1.45 decoding (process): The process defined in ISO/IEC 11172 that reads an input coded bitstream and produces decoded pictures or audio samples.

2.1.46 decoding time-stamp; DTS [system]: A field that may be present in a packet header that indicates the time that an access unit is decoded in the system target decoder.

2.1.47 de-emphasis [audio]: Filtering applied to an audio signal after storage or transmission to undo a linear distortion due to emphasis.

2.1.48 dequantization [video]: The process of rescaling the quantized DCT coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse DCT.

2.1.49 digital storage media; DSM: A digital storage or transmission device or system.

2.1.50 discrete cosine transform; DCT [video]: Either the forward discrete cosine transform or the inverse discrete cosine transform. The DCT is an invertible, discrete orthogonal transformation. The inverse DCT is defined in annex A.

2.1.51 display order [video]: The order in which the decoded pictures should be displayed. Normally this is the same order in which they were presented at the input of the encoder.

2.1.52 dual channel mode [audio]: A mode, where two audio channels with independent programme contents (e.g. bilingual) are encoded within one bitstream. The coding process is the same as for the stereo mode.

2.1.53 editing: The process by which one or more compressed bitstreams are manipulated to produce a new compressed bitstream. Conforming edited bitstreams must meet the requirements defined in this part of ISO/IEC 11172.

2.1.54 elementary stream [system]: A generic term for one of the coded video, coded audio or other coded bitstreams.

2.1.55 emphasis [audio]: Filtering applied to an audio signal before storage or transmission to improve the signal-to-noise ratio at high frequencies.

2.1.56 encoder: An embodiment of an encoding process.

2.1.57 encoding (process): A process, not specified in ISO/IEC 11172, that reads a stream of input pictures or audio samples and produces a valid coded bitstream as defined in ISO/IEC 11172.

2.1.58 entropy coding: Variable length lossless coding of the digital representation of a signal to reduce redundancy.

2.1.59 fast forward playback [video]: The process of displaying a sequence, or parts of a sequence, of pictures in display-order faster than real-time.

2.1.60 FFT: Fast Fourier Transformation. A fast algorithm for performing a discrete Fourier transform (an orthogonal transform).

2.1.61 filterbank [audio]: A set of band-pass filters covering the entire audio frequency range.

2.1.62 fixed segmentation [audio]: A subdivision of the digital representation of an audio signal into fixed segments of time.

2.1.63 forbidden: The term "forbidden" when used in the clauses defining the coded bitstream indicates that the value shall never be used. This is usually to avoid emulation of start codes.

2.1.64 forced updating [video]: The process by which macroblocks are intra-coded from time-to-time to ensure that mismatch errors between the inverse DCT processes in encoders and decoders cannot build up excessively.

2.1.65 forward motion vector [video]: A motion vector that is used for motion compensation from a reference picture at an earlier time in display order.

2.1.66 frame [audio]: A part of the audio signal that corresponds to audio PCM samples from an Audio Access Unit.

2.1.67 free format [audio]: Any bitrate other than the defined bitrates that is less than the maximum valid bitrate for each layer.

2.1.68 future reference picture [video]: The future reference picture is the reference picture that occurs at a later time than the current picture in display order.

2.1.69 granules [Layer II] [audio]: The set of 3 consecutive subband samples from all 32 subbands that are considered together before quantization. They correspond to 96 PCM samples.

2.1.70 granules [Layer III] [audio]: 576 frequency lines that carry their own side information.

2.1.71 group of pictures [video]: A series of one or more coded pictures intended to assist random access. The group of pictures is one of the layers in the coding syntax defined in this part of ISO/IEC 11172.

2.1.72 Hann window [audio]: A time function applied sample-by-sample to a block of audio samples before Fourier transformation.

2.1.73 Huffman coding: A specific method for entropy coding.

2.1.74 hybrid filterbank [audio]: A serial combination of subband filterbank and MDCT.

2.1.75 IMDCT [audio]: Inverse Modified Discrete Cosine Transform.

2.1.76 intensity stereo [audio]: A method of exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on retaining at high frequencies only the energy envelope of the right and left channels.

2.1.77 interlace [video]: The property of conventional television pictures where alternating lines of the picture represent different instances in time.

2.1.78 intra coding [video]: Coding of a macroblock or picture that uses information only from that macroblock or picture.

2.1.79 intra-coded picture; I-picture [video]: A picture coded using information only from itself.

2.1.80 ISO/IEC 11172 (multiplexed) stream [system]: A bitstream composed of zero or more elementary streams combined in the manner defined in ISO/IEC 11172-1.

2.1.81 joint stereo coding [audio]: Any method that exploits stereophonic irrelevance or stereophonic redundancy.

2.1.82 joint stereo mode [audio]: A mode of the audio coding algorithm using joint stereo coding.

2.1.83 layer [audio]: One of the levels in the coding hierarchy of the audio system defined in ISO/IEC 11172-3.

2.1.84 layer [video and systems]: One of the levels in the data hierarchy of the video and system specifications defined in ISO/IEC 11172-1 and this part of ISO/IEC 11172.

2.1.85 luminance (component) [video]: A matrix, block or single pel representing a monochrome representation of the signal and related to the primary colours in the manner defined in CCIR Rec 601. The symbol used for luminance is Y.

2.1.86 macroblock [video]: The four 8 by 8 blocks of luminance data and the two corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the pel data and sometimes to the coded representation of the pel values and other data elements defined in the macroblock layer of the syntax defined in this part of ISO/IEC 11172. The usage is clear from the context.

2.1.87 mapping [audio]: Conversion of an audio signal from time to frequency domain by subband filtering and/or by MDCT.

2.1.88 masking [audio]: A property of the human auditory system by which an audio signal cannot be perceived in the presence of another audio signal .

2.1.89 masking threshold [audio]: A function in frequency and time below which an audio signal cannot be perceived by the human auditory system.

2.1.90 MDCT [audio]: Modified Discrete Cosine Transform.

2.1.91 motion compensation [video]: The use of motion vectors to improve the efficiency of the prediction of pel values. The prediction uses motion vectors to provide offsets into the past and/or future reference pictures containing previously decoded pel values that are used to form the prediction error signal.

2.1.92 motion estimation [video]: The process of estimating motion vectors during the encoding process.

2.1.93 motion vector [video]: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture to the coordinates in a reference picture.

2.1.94 MS stereo [audio]: A method of exploiting stereo irrelevance or redundancy in stereophonic audio programmes based on coding the sum and difference signal instead of the left and right channels.

2.1.95 non-intra coding [video]: Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.

2.1.96 non-tonal component [audio]: A noise-like component of an audio signal.

2.1.97 Nyquist sampling: Sampling at or above twice the maximum bandwidth of a signal.

2.1.98 pack [system]: A pack consists of a pack header followed by one or more packets. It is a layer in the system coding syntax described in ISO/IEC 11172-1.

2.1.99 packet data [system]: Contiguous bytes of data from an elementary stream present in a packet.

2.1.100 packet header [system]: The data structure used to convey information about the elementary stream data contained in the packet data.

2.1.101 packet [system]: A packet consists of a header followed by a number of contiguous bytes from an elementary data stream. It is a layer in the system coding syntax described in ISO/IEC 11172-1.

2.1.102 padding [audio]: A method to adjust the average length in time of an audio frame to the duration of the corresponding PCM samples, by conditionally adding a slot to the audio frame.

2.1.103 past reference picture [video]: The past reference picture is the reference picture that occurs at an earlier time than the current picture in display order.

2.1.104 pel aspect ratio [video]: The ratio of the nominal vertical height of pel on the display to its nominal horizontal width.

2.1.105 pel [video]: Picture element.

2.1.106 picture period [video]: The reciprocal of the picture rate.

2.1.107 picture rate [video]: The nominal rate at which pictures should be output from the decoding process.

2.1.108 picture [video]: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. The Picture layer is one of the layers in the coding syntax defined in this part of ISO/IEC 11172. Note that the term "picture" is always used in ISO/IEC 11172 in preference to the terms field or frame.

2.1.109 polyphase filterbank [audio]: A set of equal bandwidth filters with special phase interrelationships, allowing for an efficient implementation of the filterbank.

2.1.110 prediction [video]: The use of a predictor to provide an estimate of the pel value or data element currently being decoded.

2.1.111 predictive-coded picture; P-picture [video]: A picture that is coded using motion compensated prediction from the past reference picture.

2.1.112 prediction error [video]: The difference between the actual value of a pel or data element and its predictor.

2.1.113 predictor [video]: A linear combination of previously decoded pel values or data elements.

2.1.114 presentation time-stamp; PTS [system]: A field that may be present in a packet header that indicates the time that a presentation unit is presented in the system target decoder.

2.1.115 presentation unit; PU [system]: A decoded audio access unit or a decoded picture.

2.1.116 psychoacoustic model [audio]: A mathematical model of the masking behaviour of the human auditory system.

2.1.117 quantization matrix [video]: A set of sixty-four 8-bit values used by the dequantizer.

2.1.118 quantized DCT coefficients [video]: DCT coefficients before dequantization. A variable length coded representation of quantized DCT coefficients is stored as part of the compressed video bitstream.

2.1.119 quantizer scalefactor [video]: A data element represented in the bitstream and used by the decoding process to scale the dequantization.

2.1.120 random access: The process of beginning to read and decode the coded bitstream at an arbitrary point.

2.1.121 reference picture [video]: Reference pictures are the nearest adjacent I- or P-pictures to the current picture in display order.

2.1.122 reorder buffer [video]: A buffer in the system target decoder for storage of a reconstructed I-picture or a reconstructed P-picture.

2.1.123 requantization [audio]: Decoding of coded subband samples in order to recover the original quantized values.

2.1.124 reserved: The term "reserved" when used in the clauses defining the coded bitstream indicates that the value may be used in the future for ISO/IEC defined extensions.

2.1.125 reverse playback [video]: The process of displaying the picture sequence in the reverse of display order.

2.1.126 scalefactor band [audio]: A set of frequency lines in Layer III which are scaled by one scalefactor.

2.1.127 scalefactor index [audio]: A numerical code for a scalefactor.

2.1.128 scalefactor [audio]: Factor by which a set of values is scaled before quantization.

2.1.129 sequence header [video]: A block of data in the coded bitstream containing the coded representation of a number of data elements.

2.1.130 side information: Information in the bitstream necessary for controlling the decoder.

2.1.131 skipped macroblock [video]: A macroblock for which no data are stored.

2.1.132 slice [video]: A series of macroblocks. It is one of the layers of the coding syntax defined in this part of ISO/IEC 11172.

2.1.133 slot [audio]: A slot is an elementary part in the bitstream. In Layer I a slot equals four bytes, in Layers II and III one byte.

2.1.134 source stream: A single non-multiplexed stream of samples before compression coding.

2.1.135 spreading function [audio]: A function that describes the frequency spread of masking.

2.1.136 start codes [system and video]: 32-bit codes embedded in that coded bitstream that are unique. They are used for several purposes including identifying some of the layers in the coding syntax.

2.1.137 STD input buffer [system]: A first-in first-out buffer at the input of the system target decoder for storage of compressed data from elementary streams before decoding.

2.1.138 stereo mode [audio]: Mode, where two audio channels which form a stereo pair (left and right) are encoded within one bitstream. The coding process is the same as for the dual channel mode.

2.1.139 stuffing (bits); stuffing (bytes) : Code-words that may be inserted into the compressed bitstream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream.

2.1.140 subband [audio]: Subdivision of the audio frequency band.

2.1.141 subband filterbank [audio]: A set of band filters covering the entire audio frequency range. In ISO/IEC 11172-3 the subband filterbank is a polyphase filterbank.

2.1.142 subband samples [audio]: The subband filterbank within the audio encoder creates a filtered and subsampled representation of the input audio stream. The filtered samples are called subband samples. From 384 time-consecutive input audio samples, 12 time-consecutive subband samples are generated within each of the 32 subbands.

2.1.143 syncword [audio]: A 12-bit code embedded in the audio bitstream that identifies the start of a frame.

2.1.144 synthesis filterbank [audio]: Filterbank in the decoder that reconstructs a PCM audio signal from subband samples.

2.1.145 system header [system]: The system header is a data structure defined in ISO/IEC 11172-1 that carries information summarising the system characteristics of the ISO/IEC 11172 multiplexed stream.

2.1.146 system target decoder; STD [system]: A hypothetical reference model of a decoding process used to describe the semantics of an ISO/IEC 11172 multiplexed bitstream.

2.1.147 time-stamp [system]: A term that indicates the time of an event.

2.1.148 triplet [audio]: A set of 3 consecutive subband samples from one subband. A triplet from each of the 32 subbands forms a granule.

2.1.149 tonal component [audio]: A sinusoid-like component of an audio signal.

2.1.150 variable bitrate: Operation where the bitrate varies with time during the decoding of a compressed bitstream.

2.1.151 variable length coding; VLC: A reversible procedure for coding that assigns shorter code-words to frequent events and longer code-words to less frequent events.

2.1.152 video buffering verifier; VBV [video]: A hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.

2.1.153 video sequence [video]: A series of one or more groups of pictures. It is one of the layers of the coding syntax defined in this part of ISO/IEC 11172.

2.1.154 zig-zag scanning order [video]: A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.

2.2 Symbols and abbreviations

CPB	Constrained Parameters Bitstreams
CBP	Coded Block Pattern
DCT	Discrete Cosine Transform
DFD	Displaced frame difference. Macroblock prediction error.
FDCT	Foward DCT
FLC	Fixed Length Code
IDCT	Inverse DCT
M	Distance in picture periods between successive reference pictures (I- or P- pictures).
Macroblock row	A continuous strip of macroblocks along a common horizontal strip.
MB	Macroblock
N	Distance in picture periods between successive I-pictures.
Oddification	normative mismatch control method
PMV	Predictions (for) Motion Vectors
VLC	Variable Length Code
SMPTE	Society for Motion Pictures and Television Engineers

Section 3: Software simulations

3.1 Systems simulation description

3.1.1 Encoder

The ISO/IEC 11172-1 systems multiplexer, version 0.3, creates an application-specific multiplexed bitstream compliant to ISO/IEC 11172-1 according to the Video CD specification [x]. The program is written in ANSI C. To run this code it is expected that the following two files exist in the current directory in which the program *cdisys* executes from:

video.mpg
audio.mpg

video.mpg and *audio.mpg* are MPEG-1 audio and video elementary coded bitstreams, respectively. The elementary stream files must adhere to the following coding parameters:

Table 1. Elementary stream parameters for CD-I application encoder

Video	**Audio**
Display order GOP structure: IBBPBBPBBPBBPBB bitrate = 1.150 Mbit/sec	44,100 samples per second Stereo bitrate = 0.224 Mbit/sec

Command line template

cdisys [npackets] [mplexrate]

npackets -- the total number of packets to multiplex together,
good for short tests.

mplexrate -- used to control the ratio of audio/video packets.

Command line examples:

cdisys Encoder will parse through the audio and video streams to produce a system stream based on default parameters.
cdisys 100 Generates a system stream with only 100 packets in it

cdisys 100 0.2 Generates a system stream with 100 packets, also sets the multiplex ratio of audio to video at 5:1

3.1.2 Decoder

At present, the decoder acts as a simple elementary bitstream demultiplexor for Systems bitstreams containing one video and audio elementary stream. The program is written in the flex and byacc parser language.

Compiling:
The program must be compiled with BYACC and FLEX. Both programs are widely available under the Open Software Foundation’s GNU toolkit. FLEX is chosen for its ability to handle 8 bit oriented words.

Execution:

The program *sysdec* reads the System stream from *stdin* . The command line template is:

sysdec < system.stream

Two files, audio.mpg and video.mpg, are written by *sysdec*.

3.2 Video simulation description

3.2.1 User information

The programs *mpeg1encoder* and *mpeg1decoder* have been successfully ported to several platforms including 386+ PC's (MS-DOS) and Sun Sparcstations (Sun Unix). The GNU gcc compiler is recommended on most platforms, especially MS-DOS. Few modifications of the provided makefile should be neccessary.

Usage

Both the encoder and decoder are initialized and controlled by a user edited script file, *mpeg1encode.par* and *mpeg1decode.par* respectively. The format script for the MPEG-1 encoder is:

Example	Meaning
src	/* prefix of source pictures file name */
rec	/* reconstructed pictures file name prefix */
3	/* source picture file format: 0=SIF, 1=TGA, 2=PPM 3=YUV */
1	/* recon picture file format: 0=SIF, 1=TGA, 2=PPM 3=YUV */
status	/* file name where status data is to be written */
test.mpg	/* file name for coded bitstream */
0	/* first picture in sequence */
3	/* last picture in sequence */
352	/* source horizontal size (luminance samples) */
240	/* source vertical size (luminance samples) */
12	/* pel aspect ratio code (see ISO/IEC 11172-2 2.4.3.2) */
5	/* frame rate code (see ISO/IEC 11172-2 2.4.3.2) */
1200.0	/* bit rate in multiples of kilobits/sec */
7.0	/* maximum vbv delay in picture period units */
327680	/* maxiumum vbv buffer size in units of bits */
1	/* constrained parameter flag */
15	/* Group of picture size / distance between I pictures */
3	/* distance between P pictures (M factor) */
7	/* maximum f code (motion vector / search distance) */
1	/* half pel enable ? */
1	/* use motion vector files ? */

The example script above causes the encoder to generate a sequence 4 pictures long, 352x240 picture size, 30 frames/sec, 1.2 Mbit/sec, N = 15, and M = 3, half pel accuracy motion vectors. Below is an example decoder script:

test.mpg	/* name of file containing coded bit stream */
rec	/* output path and file prefix of reconstructed pictures */
3	/* output picture format: 0=SIF, 1=TGA, 2=PPM, 3=YUV */
-	/* name of file containing trace output, -=stdout */
2	/* trace level (0: no tracing) */

3.3 Audio simulation description

Two psychoacoustic models are included in the encoder. Model I is a simple tonal and noise masking threshold generator. Model II employs cochlear masking threshold generators. The encoder currently only processes Layers I and II. The decoder processes Layers I, II and III. Mono, Stereo, and Joint Stereo are supported.

3.3.1 User information

The MPEG/audio software package consists of:

- 22 data files tables
- 9 source files (*.c)
- 4 definitions files (*.h)
- 3 test bitstreams
- * makefiles

Running the Software

To run this software, compile the programs to form an encoder executable file, musicin, and a decoder executable file, musicout. To run the program, type the name of the executable file followed by a carriage return. The program will prompt you to input the appropriate parameters. The sound input file for the encoder should be sound data, monophonic or stereophonic, sampled at 32, 44,1, or 48 kHz with 16 bits per sample. For stereophonic data the left channel sample should precede the right channel sample. The sound output file of the decoder will be the same format as the sound input file used by the decoder, except for possible byte order differences if the encoder and decoder programs are run on different computer systems which have different byte ordering conventions.

Special notes for MSDOS users:

a. The default bitrate option does not work.
b. The input/output filename defaults not compatible with MSDOS.
c. Use the large memory model for compilation.

Notes on the Software

The encoder and decoder software are configured to output the coded audio bitstreams as a string of hexadecimal ascii characters. For greater compression efficiency, compile flag, BS_FORMAT, in common.h can be switched to configure the bitstream reading and writing routines to process raw binary bitstreams.

3.3.2 Encoder

The text below give an outline of the encoding process. Each major step of the process is followed by the names of the software routines, within parentheses, which implement that step.

main(argc, argv) Obtain, set, and print out the encoding parameters to be used.
(obtain_parameters, parse_args, print_config, hdr_to_frps)
Read audio data, filter with sliding window to get 32 subband samples per channel.
(get_audio,window_subband,filter_subband)
If joint stereo mode, combine left and right channels for subbands above #jsbound#.
(*_combine_LR) where * is "I" for layer 1 and "II" for layer 2.
Calculate scalefactors for the frame, and if layer 2, also calculate scalefactor select information.
(*_scale_factor_calc)
Calculate psychoacoustic masking levels using selected psychoacoustic model.
(*_Psycho_One for psychoacoustic model 1 and psycho_anal for model 2)
Perform iterative bit allocation for subbands with low mask_to_noise ratios using masking levels from step 4.
(*_main_bit_allocation)
If error protection flag is active, add redundancy for error protection. (*_CRC_calc)
Pack bit allocation, scalefactors, and scalefactor select information (layer 2) onto bitstream.
(*_encode_bit_alloc,*_encode_scale,II_transmission_pattern)

Quantize subbands and pack them into bitstream
(*_subband_quantization, *_sample_encoding)

3.3.3 Decoder

The text below give an outline of the decoding process. Each major step of the process is followed by the names of the software routines, within parentheses, which implement that step,

main(argc, argv)
Find synchronization word and decode header modes (seek_sync,hdr_to_frps)
Decode bit allocation field (*_decode_bitalloc) note: * is "I" for layer 1, "II" for layer 2 and "III" for layer 3.
Decode scale factor fields (*_decode_scale)
If error protection bit is enabled, check CRC and attempt to recover if there is an error
(*_CRC_calc,recover_CRC_error)
read number of compressed audio code bits as indicated by the bit allocation field
(*_buffer_sample)
Decode and dequantize each subband sample (*_dequantize_sample)
Denormalize each audio sample by multiplying by appropriate scale factor (*_denormalize_sample)
convert subband samples to time domain audio samples (SubBandSynthesis)
Output audio data to file (out_fifo)

3.3.4 Bitstream verification procedure

The following verification bitstreams will be available at [ISO contact].

orig.mpg	- The original, coded MPEG/audio bitstream
deco.dec	- The audio data resulting from decoding orig.mpg
renc.dec	- The encoded MPEG/audio bitstream obtained by encoding deco.dec

The software is functioning properly if the following equations hold:
a. decoded(orig.mpg) == deco.dec
byte-swapping of deco.dec will be necessary for this
equation to hold for little-endian computers
b. encoded(deco.dec) == renc.mpg

(encode with the default options except for the following:
48 kHz sampling rate and 256 kbits/sec coded bit rate)

If the bitstream tests fail, make sure that the following variable types have at least the precision listed below:

integer	-	16 or 32bits
float	-	32 bits
double	-	64 bits.

Annex A
(informative)

Systems - code listings

A.1 Introduction

This Annex contains the C source code listings for the CD 11172-5 Systems codec.

Table A.1 List of encoder files

filename	description
README	informative file
bits.h	collection of macros for setting individual bits within a word
cdisys.c	main() routines. computes CD-I mulplexing strategy
cdisys.p	prototypes for cdisys.c
makefile	example makefile for GNU gcc Unix and MS-DOS compilers
outbits.c	low-level routines for multiplexing bits to System stream
outbits.p	prototypes for outbits.c
strstrn.c	comparison routines for string matching functions
strstrn.p	prototypes for strstrn.c

Table A.2 List of decoder files

filename	description
README	informative file
makefile	example makefile for GNU lex and yak compilers.
mpeg1sys.lex	pack demulitplexing routines
mpeg1sys.yac	packet demuliplexing routines
y.tab.h	parsing constants for start codes, prefices, et al.

A.2 Encoder

A.2.1 bits.h

```
#define BITSET(x,y)  (x&(1L<<y))
#define SETBIT(x,y) (x = (x| (1<<y)))
#define UNSETBIT(x,y) (x = (x & (0xFF^(1<<y))))
```

A.2.2 cdisys.c

```
/*
{*NAME*}
    __FILE__ -- Routines for generating CDI mpeg1 ISO11172-1 stream

{*SYNOPSIS*}
    __FILE__

{*CLASS*}
    files:generic

{*DESCRIPTION*}
    This program is intended to create ISO11172-1 compliant system
    streams.  Its orginal incarnation was to create only CD-I compliant
    (video CD) streams, however, the parameter file now allows generic
    streams to also be created.  CD-I (Video CD) really only causes
    one problem in compliance and that is in the 20 byte audio padding.

{*ENVIRONMENT*}

{*SYSTEM REQUIREMENTS*}
    unix -- originally developed under both Sun 4.1.x and Linux 1.x

{*COMPILATION INSTRUCTIONS*}
    make binary

{*RESOURCE CONTROL*}
    $Id: cdisys.c,v 1.10 1994/09/27 22:20:38 quandt Exp quandt $
    $Log: cdisys.c,v $
 * Revision 1.10  1994/09/27  22:20:38  quandt
 * func:cont
 * Still changing how this thing really should work
 *
 * Revision 1.9  1994/05/14  00:54:58  quandt
 * misc:bug
 * Fixed many bugs mostly pertaining to the number of AU's per packet.
 * I'm still not sure if they are right, but I think they are closer
 *
 * Revision 1.8  1994/05/02  18:08:00  quandt
 * func:cont
 * This version does work for the CD32, even with PTS/DTS ignored.
 *
 * Revision 1.7  1994/03/04  21:57:43  quandt
 * func:cont
 * Finished the maxsector count, allows a fixed number of sectors to be .
 *
 * Revision 1.6  1994/03/03  01:29:49  quandt
 * func:cont
 * Added ability to limit the number of sectors outputted
 *
 * Revision 1.5  1994/02/28  19:02:59  quandt
 * bug:misc
 * Changed time when to output first audio pack.  The first try
 * would end up with the audio buffer overflowing.
 *
 * Revision 1.4  1994/02/28  17:49:13  quandt
 * func:cont
 * All now complies to the HEURIS prototype standards (and compiles)
 *

{*SEE ALSO*}
```

```
{*BUGS*}
    EOF is not handled properly at this point

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/


#include <stdio.h>
#include <unistd.h>

/* kludge code for dos, so that I have 32 bit ints to work with  */
/* #define int long */


#define MAXLINEBUF 150

#define PADDINGSTREAMID 0xBE

#define AUDIOVIDEO 0
#define AUDIOONLY 1
#define VIDEOONLY 2
#define ENDOFINPUT 3
#define NOTPICT -1   /* picture types */
#define IFRAME 0x08
#define PFRAME 0x10
#define BFRAME 0x18

#define PACKSTARTCODE 0x1BAL
#define MARKERBIT 0x01L
#define PADDINGBYTE 0xFF
#define STUFFINGBYTE 0xFF
#define SYSTEMHEADERSTARTCODE 0x1BBL
#define PACKETSTARTCODE 0x01L
#define ISOENDCODE 0x01B9L

#define MAXSYSBUF (46.0*1024.0 + 0.1)
#define MAXVIDBUF (46.0*1024.0 + 0.1)
#define MAXAUDBUF (4.0*1024.0 + 0.1)
#define MINSYSBUF (0.0 - 0.1)
#define MINVIDBUF (0.0 - 0.1)
#define MINAUDBUF (0.0 - 0.1)


#define CLOCKFREQ 90000.0    /* 90,000 HZ clock */

#define AUDIOAU "\x00\x00\x00\x01"
#define SIZEOFAUDIOAU 4  /* length in bytes */
#define VIDEOAU "\x00\x00\x01\x00"
#define SIZEOFVIDEOAU 4  /* length in bytes */
#define AUDIOSAMPLESPERAU 1152   /* audio samples per AU */



FILE *outfile;

struct inparams
    {
    /* parameters supplied by the user */

    FILE *audio;
    float audiorate;  /* bits per second */
    unsigned char audiostreamid;
    int audiobound;

    FILE *video;
    float videorate; /* bits per second */
    unsigned char videostreamid;
    int videobound;

    float audiosamplerate;   /* 48000, 44100, 38000, etc */
```

```
    int packetheaderlength;
    int packheaderlength;
    int packetlength;
    int video_M;
    float videoframesec;
    float packsize;
    float dsmpacksize;    /* digital storage media, ie this may be larger  */
                         /* then the actual user data, sector headers, etc */
    float vidpacketsize; /* size of a video packet */
    float viddatasize;   /* data contained within a video packet */
    float audpacketsize; /* size of a audio packet */
    float auddatasize;   /* data contained within an audio packet */
    int packetsperpack;


    int maxsectors;  /* if supplied, the maximum number of sectors to output*/

    int runforever;      /* boolean flag if set continue the encoder
                             even on EOF of all of the inputs, ie use
                             padding packets*/
    int muxrate;
    int firstAudioPTS;
    int firstVideoPTS;
    int video_lock;
    int audio_lock;
    int CSPS_flag;
    int fixed_flag;      /* fixed bitrate or not */
    int CDI_type;        /* if set then output CDI type headers and audio padding */
    int VCD_nulls;       /* if set  then include VCD type 20 NULLS after all audio packs */
    };

struct calcparams
    {
    /* calculated parameters from the user supplied ones */
    int Rmux;
    int muxrate; /* rate at which data arrive to decoder, 50 byte units */
    int ratebound;    /* just Rmux in 50 byte units, ie, divided by 50 */
    long SCR;        /* this is only 32 bits, that's good enough for now, needs to be 33 for
real long sequences */
    int byteSCR; /* number of bytes output at current SCR value */
    float videoaudioratio;   /* number of video packets output to 1 audio */
    int videostartupdelay;   /* video startup delay */
    int audiostartupdelay;   /* audio startup delay */
    int videoPTSdelay;   /*const value added to I/P frames for reorder delay */
    int videoPTSconst;   /* values added to old PTS value to get new one */
    int audioPTSconst;
    long videoPTS;       /* Last PTS value for respective stream */
    long audioPTS;
    };


#include <cdisys.p>
#include <strstrn.p>
/*
{*NAME*}
    main -- Main driver for the CDI MPEG1 ISO11172-1 system encoder

{*SYNOPSIS*}
    int main(int argc, char *argv[])

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Main routine for producing a mpeg system compliant stream.  All
    the input is taken from a parameter file.

{*ALGORITHM*}
    Get all the parameters
    Produce a stream

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
```

```
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int main(int argc, char *argv[])
    {
    struct inparams UserParams; /* user supplied parameters */
    struct calcparams CalcParams;/* calculated from the user params */


    if (GetParams(argc,argv,&UserParams, &CalcParams) != 0)
        {
        fprintf(stderr,"Error reading input parameters\n");
        exitroutine();
        }

    ProcessStreams(&UserParams, &CalcParams);

    /* orginally opened in ReadUserParams */
    fclose(outfile);
    }

/*
{*NAME*}
    ProcessStreams -- CDI ISO11172-1 stream encoder.

{*SYNOPSIS*}
    int ProcessStreams(struct inparams *UserParams, \* User parameters *\
        struct calcparams *CalcParams)   \* Calculated Parameters *\

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    This is the actual work function of this program.  It loops
    until the end of processing.  It spits out either a audio pack,
    video pack or padding pack based on buffer fullness.

{*ALGORITHM*}
    If CDI
        output the CDI type headers -- 1 for audio, 1 for video
    else
        output a normal system type header

    While continue processing
        Output a pack based on buffer fullness, priority given to audio.
        Update and check the buffers.

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int ProcessStreams(struct inparams *UserParams, /* User parameters */
        struct calcparams *CalcParams)   /* Calculated Parameters */
    {
    int i,
        vidsect, /* num of video sect to spit out before sending audio sect*/
        vidcnt=0,        /* total number of video sectors sent */
        audcnt=0,        /* total audiosectors sent */
        counter;
    float ftmp;
    float currentratio;  /* current relationship of audio:video */
    float padsectcount = 0.0;
    float vidsectcount = 0.0;
    float audsectcount = 0.0;
    float sectcount = 0.0;
    float VBV = 0.0; /* buffer for video */
    float ABV = 0.0; /* buffer for audio */
    float SBV = 0.0; /* buffer for system */
```

```
    /* Just send out the first sectors for audio and video, these
    don't contain any real data they just initialize the decoder */

    if (UserParams->CDI_type)
        {
        cdi_vid_sect0(UserParams,CalcParams);
        sectcount++;
        CalcParams->SCR += UserParams->dsmpacksize/(UserParams->muxrate * 50)*CLOCKFREQ;


        cdi_aud_sect0(UserParams,CalcParams);
        sectcount++;
        CalcParams->SCR += UserParams->dsmpacksize/(UserParams->muxrate * 50)*CLOCKFREQ;
        }
    else
        {
        cdi_pad_sect0(UserParams,CalcParams);
        sectcount++;
        CalcParams->SCR += UserParams->dsmpacksize/(UserParams->muxrate * 50)*CLOCKFREQ;
        }

    while ( ((!feof(UserParams->video)) &&
            (!feof(UserParams->audio)) &&
            (UserParams->maxsectors--)) || UserParams->runforever )
        {
#ifdef DEBUG_BUFF
fprintf(stderr,"SCR: %10d   VBV %10.0f ABV %10.0f SBV %10.0f  ",CalcParams->SCR,VBV,ABV,SBV);
#endif

#ifdef DEBUG_BUFFVID
printf("%8.0f\n",VBV);
#endif
#ifdef DEBUG_BUFFAUD
printf("%8.0f\n",ABV);
#endif

        /* Start with a check for buffer problems */
        if (ABV > MAXAUDBUF) fprintf(stderr,"Audio buffer OVERflow (in) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (VBV > MAXVIDBUF) fprintf(stderr,"Video buffer OVERflow (in) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (SBV > MAXSYSBUF) fprintf(stderr,"System buffer OVERflow (in) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (ABV < MINAUDBUF) fprintf(stderr,"Audio buffer UNDERflow (in) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (VBV < MINVIDBUF) fprintf(stderr,"Video buffer UNDERflow (in) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (SBV < MINSYSBUF) fprintf(stderr,"System buffer UNDERflow (in) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);

        /* The control of whether to output a video audio or padding
        is controled by buffer fullness.  Ie if the buffer in question is almost
        empty better put another one out, The minimum is controlled
        by the amount each 1/75.0 of a second that needs to go out, the max
        is controled by current buffer fullness plus amount added (ie don't
        go over )  priority is given to Audio */

        if (MAXAUDBUF > (ABV + UserParams->audpacketsize))
            {
#ifdef DEBUG_BUFF
fprintf(stderr,"aud \n");
#endif
            cdi_aud_sect(UserParams,CalcParams);
            ABV += UserParams->audpacketsize;
            }
        else if (MAXVIDBUF > (VBV + UserParams->vidpacketsize))
            {
#ifdef DEBUG_BUFF
fprintf(stderr,"vid \n");
#endif
            cdi_vid_sect(UserParams,CalcParams);
            VBV += UserParams->vidpacketsize;
            }
        else
            {
```

```
#ifdef DEBUG_BUFF
fprintf(stderr,"pad \n");
#endif
            cdi_pad_sect(UserParams,CalcParams);
            }

        /* All cases add (full sector) to system buffer */
        SBV += UserParams->dsmpacksize;

        /*  Check the supplied time for when to output audio video
            start the decoders as required
            the amount to remove from each buffer is dependent on the
            bitrate of each channel and amount of pack/packet header
            stuff, I've done this stuff fixed to make life a bit
            easier
        */

        if (CalcParams->SCR >= UserParams->firstAudioPTS)
            ABV -= (UserParams->audiorate/8.0) *
                (UserParams->dsmpacksize/(UserParams->muxrate*50)) *
                (UserParams->audpacketsize/UserParams->auddatasize);

        if (CalcParams->SCR >= UserParams->firstVideoPTS)
            VBV -= (UserParams->videorate/8.0) *
                (UserParams->dsmpacksize/(UserParams->muxrate*50)) *
                (UserParams->vidpacketsize/UserParams->viddatasize);

        SBV -= UserParams->dsmpacksize;

        /* check again for buffer problems */
        if (ABV > MAXAUDBUF) fprintf(stderr,"Audio buffer OVERflow (out) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (VBV > MAXVIDBUF) fprintf(stderr,"Video buffer OVERflow (out) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (SBV > MAXSYSBUF) fprintf(stderr,"System buffer OVERflow (out) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (ABV < MINAUDBUF) fprintf(stderr,"Audio buffer UNDERflow (out) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (VBV < MINVIDBUF) fprintf(stderr,"Video buffer UNDERflow (out) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);
        if (SBV < MINSYSBUF) fprintf(stderr,"System buffer UNDERflow (out) ABV %8.0f VBV %8.0f
SBV %8.0f SCR %d!\n",ABV,VBV,SBV,CalcParams->SCR);

        CalcParams->SCR += UserParams->dsmpacksize/(UserParams->muxrate * 50)
            *CLOCKFREQ;
        }

    }/* end of process streams */



/*
{*NAME*}
    outputemptypack -- Output an empty pack.

{*SYNOPSIS*}
    static int outputemptypack(int totallength, \* lenght of a pack *\
        struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Send a padding PACK to the output.

{*ALGORITHM*}
    Output all the header information for this pack.
    Output padding packet of the required size.

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
```

```
{**}
*/
static int outputemptypack(int totallength, /* lenght of a pack */
    struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */


    /* SCR */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit, but
                                 since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);

    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);

    outputpadding(totallength-12,UserParams,CalcParams);
    }

/*
{*NAME*}
    outputpadding -- Output a CDI/ISO11172-1 compliant padding stream.

{*SYNOPSIS*}
    static int outputpadding(int totallength, \* lenght of a padding stream *\
        struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    This routine will output a padding PACKET.

{*ALGORITHM*}
    Output packet header stuff.
    Output padding bytes necessary to fill the packet.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int outputpadding(int totallength,     /* lenght of a padding stream */
        struct inparams *UserParams,
        struct calcparams *CalcParams)
    {
    int tmp,
        i;
    char    padblk[4096];

#ifdef DEBUG
fprintf(stdout,"\npadding\n");
#endif


    /* output a padding packet of the exact length specified */
    outbits(PACKETSTARTCODE,0,NULL,24,23);
    outbits(PADDINGSTREAMID,0,NULL,8,7);
```

```
    /* The length is calculated as the amount remaining to send after
    the length field... the total so far is 24bits + 8bits + 16bits = 6 bytes */
    tmp = totallength - 6;
    outbits(tmp,0,NULL,16,15);

    /* According to the White book 3.2.2 need to make sure that the data
    bytes is on even boundary... so I'll throw a stuffing byte in
    if  not padding even number of bytes */

    if (tmp%2 == 0)
        outbits(STUFFINGBYTE,0,NULL,8,7);

    /* now the else part of the packet */
    outbits(0x0FL,0,NULL,8,7);

    /* now the actual data that is part of a padding packet */
    /* this is a loop to fill out the remaining bytes, so far in this
    packet we have sent 7 bytes
    for(i=0;i<(totallength-7);i++)
        outbits(PADDINGBYTE,0,NULL,8,7);
    For now I'm going to use some memory routines so that things
    are cleaner in debugging and print outs.  In real time systems
    it is probably better to just have a static block of data to pull
    from */

    if (tmp%2 == 0)
        {
        memset(padblk,PADDINGBYTE,(totallength-8));
        outbits(0,1,padblk,(totallength-8)*8,7);
        }
    else
        {
        memset(padblk,PADDINGBYTE,(totallength-7));
        outbits(0,1,padblk,(totallength-7)*8,7);
        }

    }

/*
{*NAME*}
    GetParams -- CDI ISO11172-1 retrieve/calculate input parameters.

{*SYNOPSIS*}
    static int GetParams(int argc, char *argv[],
        struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Create or get all the necessary parameters used in this program.

{*ALGORITHM*}
    Get the user parameters.
    Calculate the internal variables from the supplied user parameters.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int GetParams(int argc, char *argv[],
    struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    int ret = 0;
    float tmp;
```

```
    ReadUserParams(argc, argv, UserParams);

    /* This is the frame rate/sec *90000 */
    /* Each PTS gets incremented by this amount, it is based on the frame rate */
    CalcParams->videoPTSconst =
        (int)((float)1/(float)UserParams->videoframesec * CLOCKFREQ);

    /* This is the audio PTS constant, it is based on the number of samples
    per frame (derived from the standard as 1152 bytes) */
    CalcParams->audioPTSconst = AUDIOSAMPLESPERAU/UserParams->audiosamplerate * CLOCKFREQ;


    /* Now the reorder delay necessary for video frames of type I/P.
    This is necessary since ordering of the frames in the encoded
    form is different from decoding order.  Since B frames are displayed
    as soon as they are recieved this value is actually the number of
    frames between successvie P/I frames * the rate per frame (in
    other words the M * 1/framerate * CLOCKFREQ) */
    CalcParams->videoPTSdelay  =
        CalcParams->videoPTSconst * UserParams->video_M;

    /* the clock starts at 0 (easier on the math for other things) */
    CalcParams->SCR = 0;


    /* Set the audio and video decoder start times... note that
    video starts one frame period before its PTS */

    CalcParams->videoPTS = UserParams->firstVideoPTS - CalcParams->videoPTSconst;
    CalcParams->audioPTS = UserParams->firstAudioPTS ;


#ifdef DEBUG
fprintf(stderr,"videoPTS %d\n, videoPTSdelay %d\n, audioPTS %d\n",
CalcParams->videoPTS, CalcParams->videoPTSdelay, CalcParams->audioPTS);
#endif

    return(ret);
    }

/*
{*NAME*}
    ReadUserParams -- CDI ISO1172-1 encoder, read user provided parameters.

{*SYNOPSIS*}
    static int ReadUserParams(int argc, char *argv[],
        struct inparams *UserParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Read the user parameters from the supplied param file (command line option).
    A sample param file follows

        # Sample param file for system encoder
        audio.mpg    # Name of the input audio file
        video.mpg    # Name of the input video file
        sys.mpg          # Name of the output system file
        e0               # hex value for video id, normally e0 (dec 224)
        c0               # hex value for audio id, normally c0 (dec 192)
        3                # MPEG video M value, ie size between P frames
        1152000.0        # Video bit rate
        224000.0     # Audio bit rate
        0                # bool flag, if set, encoder continues even on EOF of aud/vid
        500              # max number of sectors to create
        3528         # muxrate, can be different from totaldeliveryrate/50, why?
        15000            # First Audio pts in 90khz clock units
        21000            # First Video pts in 90khz clock units
        0                # Fixed bitrate flag 1->set, 0->variable bitrate stream
        1                # CSPS flag
        1                # Audio lock flag 1 or 0
        1                # Video lock flag 1 or 0
        44100.0          # Audio sample rate
        29.97            # Video frame rate
        2324         # Number of bytes per pack (CDI = 2324)
```

```
        2352        # Number of bytes per pack + any DSM related stuff (CDI = 2352)
        2312        # Number of bytes per video packet (CDI=2312)
        2294        # Number of data bytes per video packet (CDI=2294)
        2312        # Number of bytes per audio packet (CDI=2312)
        2279        # Number of data bytes per audio packet (CDI=2279, don't include the 20
bytes of nulls here!)
        1               # Number of packets per pack (not implemented, fixed to 1, CDI=1)
        1               # (0|1) If set then output CDI type headers and audio (20byte) padding
        1               # (0|1) If set then include VCD type nulls with audio packs

{*ALGORITHM*}
    Read all the parameters into the UserParam structure.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int ReadUserParams(int argc, char *argv[],
    struct inparams *UserParams)
    {
    char line[200];
    char    stmp[100];
    char    ctmp;
    int itmp;
    float   ftmp;
    FILE *paramfil;


    /* the first param is the name of the param file */
    if ((argc != 2)||(access(argv[1],R_OK)))
        {
        fprintf(stderr,"usage: cdisys paramfile\n");
        exit(-1);
        }


    if ( (paramfil = fopen(argv[1],"r")) == NULL)
        {
        fprintf(stderr,"could not open param file?\n");
        exit(-1);
        }

    /* skip the first line */
    fgets(line,MAXLINEBUF,paramfil);

    /* read the audio file name */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%s",stmp);
    UserParams->audio = fopen(stmp,"rb");
    fprintf(stderr,"%15s audio file\n",UserParams->audio);

    /* read the video file name */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%s",stmp);
    UserParams->video = fopen(stmp,"rb");
    fprintf(stderr,"%15s video file\n",UserParams->video);

    /* read the output system file name */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%s",stmp);
    outfile = fopen(stmp,"wb");
    fprintf(stderr,"%15s output file\n",outfile);

    /* read the value for the video id */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%x",&itmp);
    UserParams->videostreamid = (unsigned char)itmp;
    fprintf(stderr,"%15x video stream id\n",UserParams->videostreamid);

    /* read the value for the audio id */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%x",&itmp);
    UserParams->audiostreamid = (unsigned char)itmp;
```

```
    fprintf(stderr,"%15x audio stream id\n",UserParams->audiostreamid);

    /* read the M value for distance to P frames */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->video_M = itmp;  /* number of frames between successive
    fprintf(stderr,"%15d video M distance (distance between P frames)\n",UserParams->video_M);

    /* read teh video bit rate */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->videorate =  ftmp;
    fprintf(stderr,"%15f video bit rate\n",UserParams->videorate);

    /* read the audio bit rate */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->audiorate = ftmp;
    fprintf(stderr,"%15f audio bit rate\n",UserParams->audiorate);

    /* read bool which determines how the encoder functinos on EOF */
    /* if set the encoder never stops, useless right... well not */
    /* if this ever gets put into a broadcast environment */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->runforever = itmp;
    fprintf(stderr,"%15d Run forever? (1 implies true)\n",UserParams->runforever);

    /* read max number of sectors to read */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->maxsectors = itmp;
    fprintf(stderr,"%15d Maximum number of packs/sectors to generate\n",UserParams-
>maxsectors);

    /*read the mux rate, for now this can differ from the system bit rate */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->muxrate = itmp;
    fprintf(stderr,"%15d Muxrate\n",UserParams->muxrate);

    /* Read in start time for first audio PTS */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->firstAudioPTS = itmp;
    fprintf(stderr,"%15d First Audio PTS, ie time to start up audio decoder\n",UserParams-
>firstAudioPTS);

    /* Read in start time for first video PTS */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->firstVideoPTS = itmp;
    fprintf(stderr,"%15d First Video PTS\n",UserParams->firstVideoPTS);

    /* Read Fixed_flag ie fixed bitrate or variable bit rate */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->fixed_flag = itmp;
    fprintf(stderr,"%15d Fixed bitrate operation\n",UserParams->fixed_flag);

    /* Read CSPS_flag */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->CSPS_flag = itmp;
    fprintf(stderr,"%15d CSPS flag\n",UserParams->CSPS_flag);

    /* Read audio lock flag */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->audio_lock = itmp;
    fprintf(stderr,"%15d audio lock flag\n",UserParams->audio_lock);

    /* Read video lock flag */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->video_lock = itmp;
    fprintf(stderr,"%15d video lock flag\n",UserParams->video_lock);

    /* Read audio sample rate */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->audiosamplerate = ftmp; /* 44100.0; */
    fprintf(stderr,"%15f audio sample rate\n",UserParams->audiosamplerate);

    /* Read video frames a second */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->videoframesec = ftmp; /* 29.97      number of video frames a second */
    fprintf(stderr,"%15f Video frame rate\n",UserParams->videoframesec);
```

```
    /* pack size in bytes */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->packsize = ftmp;
    fprintf(stderr,"%15f Number of bytes in a pack\n",UserParams->packsize);

    /* DSM pack size in bytes */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->dsmpacksize = ftmp;
    fprintf(stderr,"%15f Number of bytes needed in the DSM to store a pack\n",UserParams-
>dsmpacksize);

    /* video packet size in bytes */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->vidpacketsize = ftmp;
    fprintf(stderr,"%15f Size of video packet in bytes\n",UserParams->vidpacketsize);

    /* video data packet size in bytes */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->viddatasize = ftmp;
    fprintf(stderr,"%15f Amount of video data in a give packet\n",UserParams->viddatasize);

    /* audio packet size in bytes */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->audpacketsize = ftmp;
    fprintf(stderr,"%15f Size of an audio packet\n",UserParams->audpacketsize);

    /* audio data bytes per packet */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%f",&ftmp);
    UserParams->auddatasize = ftmp;
    fprintf(stderr,"%15f Amount of audio data within a packet\n",UserParams->auddatasize);

    /* Read packets per pack */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->packetsperpack = itmp;
    fprintf(stderr,"%15d Number of packets per pack\n",UserParams->packetsperpack);

    /* Read the CDI flag, if set then output CDI type system streams */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->CDI_type = itmp;
    fprintf(stderr,"%15d If 1 then output CDI type system headers\n",UserParams->CDI_type);

    /* Read the param that determines if to pad audio packs with 20 nulls */
    /* this is a Video CD thing */
    fgets(line,MAXLINEBUF,paramfil);sscanf(line,"%d",&itmp);
    UserParams->VCD_nulls = itmp;
    fprintf(stderr,"%15d If 1 output 20 NULLS after audio packs (VCD spec)\n",UserParams-
>VCD_nulls);

    fclose(paramfil);
    }


/*
{*NAME*}
    getpicturetype -- Returns I B or P frame for an MPEG encoded picture.

{*SYNOPSIS*}
    int getpicturetype(char *ptr) \* pointer to picture start code *\

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Given a pointer to  picture startcode determine the type of picture
    that is contained in the following data (I|B|P).  This is
    needed in determining how much to increment the PTS/DTS values
    in later calculations.


{*ALGORITHM*}
    Pointer is to the beginning of a startcode, use the spec to see
    where the bits for picture type fall.

{*SEE ALSO*}

{*BUGS*}
```

```
{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int getpicturetype(char *ptr) /* pointer to picture start code */
    {
    /* ptr point to a PICTURE START CODE */
    int picttype;
    char tmp;

    /* the picture type should be found as follows */
    /* (32bits for start code ) (10 bits for temporal_reference)
    (3 bits for picture type ) */

    /* WARNING THIS CODE ASSUMES 8 BIT BYTES..., of if longer chars,
    that only bits 0..7 have data. */
    tmp = ptr[5] & 0x38; /* 00111000, mask out unwanted bits */
    picttype = tmp;
    return(picttype);
    }

/*
{*NAME*}
    exitroutine -- CDISYS exit routine (die die die die as in failed).

{*SYNOPSIS*}
    static int exitroutine()

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    General death routine, if need to exit in the code.

{*ALGORITHM*}
    Just die with some lovely message.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int exitroutine()
    {
    fprintf(stderr,"According to ISO/ITC standards of diplomacy,\n");
    fprintf(stderr,"it has become necessary for this program to fail.\n");
    fprintf(stderr,"I'm sorry, please take any problems you have encountered to your\n");
    fprintf(stderr,"respective representative.\n");
    exit(-1);   /*die die die die die die*/
    }



/*
{*NAME*}
    cdi_pad_sect0 -- Routine to output first sector of system stream

{*SYNOPSIS*}
    int cdi_pad_sect0(struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg
```

```
{*DESCRIPTION*}
    This routine outputs the first pack.  Its sole purpose is to output
    a systems header as defined by ISO11172-1.  Whereas
    vid_sect0 and aud_sect0 put out seperate audio and video system headers
    this one puts out a single header with audio and video defined
    within it.


{*ALGORITHM*}
    Out the pack header
    Out the sytems header defining both audio and video
    Output a padding packet to fill in the remaining.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/

int cdi_pad_sect0(struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    /* Output CDI compliant mpeg padding sector with SYSTEM header */
    /* this one has the audio and video system id's defined */
#ifdef DEBUG
fprintf(stdout,"\ncdi_pad_sect0\n");
#endif

    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */


    /* SCR */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit, but
                                 since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);

    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);

    /* SYSTEM header as stated in the white book figure IV.10 */
    outbits(SYSTEMHEADERSTARTCODE,0,NULL,32,31);
    outbits(12L,0,NULL,16,15);       /* system header length's always fixed */
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(UserParams->muxrate,0,NULL,22,21);      /*ratebound */
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(1L,0,NULL,6,5);  /*audiobound */
    outbits(UserParams->fixed_flag,0,NULL,1,0);             /* fixed_flag */
    outbits(UserParams->CSPS_flag,0,NULL,1,0);              /* CSPS */
    outbits(UserParams->audio_lock,0,NULL,1,0);             /* audio_lock */
    outbits(UserParams->video_lock,0,NULL,1,0);             /* video_lock */

    outbits(MARKERBIT,0,NULL,1,0);
    outbits(1L,0,NULL,5,4);      /*video bound */
    outbits(0xFFL,0,NULL,8,7);   /* reserved byte */

    /* the std bounds for audio */
    outbits(UserParams->audiostreamid,0,NULL,8,7);  /* audio stream id */
    outbits(0x03L,0,NULL,2,1);
    outbits(0x00L,0,NULL,1,0);   /* STD buffer scale, indicates scale of video */
    outbits(32L,0,NULL,13,12);   /* CSPS is set this in constrained to 4k */
```

```
    /* the std bounds for video */
    outbits(UserParams->videostreamid,0,NULL,8,7); /* video stream id */
    outbits(0x03L,0,NULL,2,1);
    outbits(0x01L,0,NULL,1,0);   /* STD buffer scale, indicates scale of video */
    outbits(46L,0,NULL,13,12);   /* since CSPS is set this in constrained to 46*1024 */

    /* Padding packet of 2297 bytes to fill out the rest of the sector */
    outputpadding(UserParams->viddatasize, UserParams, CalcParams);
    }


/*
{*NAME*}
    cdi_vid_sect0 -- Output a proper CD-I video sector 0.

{*SYNOPSIS*}
    int cdi_vid_sect0(struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Output a system header as defined by the Video CD spec, this only
    defines the video streams.

{*ALGORITHM*}
    Output a pack header
    Output a system header defining only the video stream
    Output a padding stream to fill the rest of the packet.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int cdi_vid_sect0(struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    /* Output CDI compliant mpeg video sector with SYSTEM header */
#ifdef DEBUG
fprintf(stdout,"\ncdi_vid_sect0\n");
#endif

    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */


    /* SCR */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit, but
                                 since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);

    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);

    /* SYSTEM header as stated in the white book figure IV.10 */
    outbits(SYSTEMHEADERSTARTCODE,0,NULL,32,31);
    outbits(9L,0,NULL,16,15);        /* system header length's always fixed */
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(UserParams->muxrate,0,NULL,22,21);      /*ratebound */
    outbits(MARKERBIT,0,NULL,1,0);
```

```
    outbits(0L,0,NULL,6,5);  /*audiobound */
    outbits(UserParams->fixed_flag,0,NULL,1,0);             /* fixed_flag */
    outbits(UserParams->CSPS_flag,0,NULL,1,0);              /* CSPS */
    outbits(UserParams->audio_lock,0,NULL,1,0);             /* audio_lock */
    outbits(UserParams->video_lock,0,NULL,1,0);             /* video_lock */

    outbits(MARKERBIT,0,NULL,1,0);
    outbits(1L,0,NULL,5,4);      /*video bound */
    outbits(0xFFL,0,NULL,8,7);   /* reserved byte */

    /* the std bounds for video */
    outbits(UserParams->videostreamid,0,NULL,8,7); /* video stream id */
    outbits(0x03L,0,NULL,2,1);
    outbits(0x01L,0,NULL,1,0);   /* STD buffer scale, indicates scale of video */
    outbits(46L,0,NULL,13,12);   /* since CSPS is set this in constrained
                                 to 46*1024 */

    /* Padding packet of 2297 bytes to fill out the rest of the sector */
    outputpadding(2297L, UserParams, CalcParams);
    }

/*
{*NAME*}
    cdi_vid_sect -- Output a proper CD-I sector (used after cdi_vid_sect0)

{*SYNOPSIS*}
    int cdi_vid_sect(struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Output a pack that contains video data within its packet.

{*ALGORITHM*}
    Pack header
    Out a video packet

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int cdi_vid_sect(struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    /* Output CDI compliant mpeg video sector (without the SYSTEM header) */
#ifdef DEBUG
fprintf(stdout,"\ncdi_vid_sect\n");
#endif

    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */

#ifdef DEBUG
printf("\nSCR %u\n",CalcParams->SCR);
#endif

    /* SCR */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit,
                                 but since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);
```

```
    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);

    /* Now the packet for video */
    cdivideopacket(UserParams,CalcParams);
    }

/*
{*NAME*}
    cdivideopacket -- Output the CD-I packet associated to a sector.

{*SYNOPSIS*}
    static int cdivideopacket(struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Routine to setup output for outputting a cdi video compliant packet.


{*ALGORITHM*}
    Read the data from the video stream
    Based on the amount read output the video packet taking care of EOF.
    If EOF output a end code

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int cdivideopacket(struct inparams *UserParams,
        struct calcparams *CalcParams)
    {
    /* WARNING THIS MODULE IS VERY CLOSELY COUPLED TO outvideopacket, sorry */
    unsigned char buf[4048]; /* just allocated enough for now, avoid dynamic
                                allocation for potential real time app */
    unsigned char buf2[4];      /* used for finding the AU at the boundary of a packet */
    int bytecount,
        numtostuff,     /* number of stuffing bytes required */
        bytesread,      /* number of bytes read from input */
        tmp,
        i;

#ifdef DEBUG
fprintf(stdout,"\nvideopacket\n");
#endif


    /* Try to read in enough data for one VIDEO packet.  If I can't then
    add padding packets to fill out this pack with another padding packet.
    (This implies that partial video packets need another padding packet
    right next to it) */
    if (!feof(UserParams->video)&&
        (bytesread = fread(buf,1,UserParams->viddatasize,UserParams->video)) == UserParams-
>viddatasize)
        {
        outvideopacket(UserParams->vidpacketsize,bytesread,UserParams,CalcParams,buf);
        }
    else
        {
        /* Have reached the EOF of the video input... */

        /* Here the same problem arises for video as audio... the
        white book specification is bonkers on the end of stream.
```

```
bytes remaining  What to do
2291-2294            Drop the extra data and treat as if we have
                     a video packet that is 2290 bytes
                     (2290 + 18 bytes = 2308)

2290             Since want to output 2290 + 18 bytes =2308... do it.

                     Now the rest is tough, need to make sure that
                     don't use more then 16 stuffing bytes.  The
                     amount of stuffing used is based on the AU
                     type found in the data.  The smallest
                     even byte padding stream is 8 bytes!
                     Currently the worse case is that the amount
                     of stuffing bytes = 11, this leaves 5 to
                     play with.  (outvideopacket could be modified
                     to leave 7 stuffing bytes avail if the
                     STDbuffer/size are coded in...).  So from
                     this the following table arises

2285-2289            Just like 2290. Since there are enough stuffing
                     bytes avail to fill out the video packet.

2284             Send out a vid packet that totals 2284 + 16 = 2300
                     (this says that the outvideopacket code will
                     trim off the 2 bytes that is normally pads for the
                     STDbuffer/size field).  An 8 byte padding packet
                     can then be sent!

2277-2283            Same as 2284, there are again enough stuffing
                     bytes to prevent the 16 byte stuff max.

0-2276               Okay the only trick here is to make sure
                     every packet stays even byte aligned.  Again
                     a maximum of 5 bytes for stuffing is allowed.
                     ( the modulo 2 just adds one to odd bytesread)
                     Vidsize = bytesread + bytesread%2 + 18
                     paddsize = 2308 - Vidsize

*/
/* for now I'll keep this hardcoded, don't know how to tie them
to the user provided parmeters... in any case this is only
handling a proper EOF on the stream. */
if (bytesread > 2290)
    bytesread = 2290; /* drop the uncodeable bytes... this may
                         cause problems */

if (bytesread >= 2285)
    {
    outvideopacket(2308,bytesread,UserParams,CalcParams,buf);
    }
else
    {
    if (bytesread >= 2277)
        {
        outvideopacket(2300,bytesread,UserParams,CalcParams,buf);
        outputpadding(8,UserParams,CalcParams);
        }
    else
        {
        int vidsize, paddsize;
        vidsize = bytesread + bytesread%2 + 18;
        paddsize = 2308 - vidsize;
        outvideopacket(vidsize,bytesread,UserParams,CalcParams,buf);
        outputpadding(paddsize,UserParams,CalcParams);
        }
    }

/* Output an ISO-end code */
if (!UserParams->runforever)
    outbits(ISOENDCODE,0,NULL,32,31);
else
    outbits(0L,0,NULL,32,31);

}
```

```
    }/* end cdivideopacket */

/*
{*NAME*}
    outvideopacket -- Actual output routine for CDI video packets

{*SYNOPSIS*}
    static int outvideopacket(int desiredsize,  \* size of packet to output *\
        int bytesread, \* bytes read in input buffer *\
        struct inparams *UserParams,
        struct calcparams *CalcParams,
        char *buf) \* input buffer associated with bytesread *\

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    The actual work routine for outputting a video packet.

{*ALGORITHM*}
    Output the header stuff.
    Count the number of AU within the video data read
    Determine the picture type of the first AU within the packet,
        this is used to determine the amount to increment the PTS/DTS
        values, along with wether or not to include them in the packet
        (rules defined within the spec).
    Output any necessary padding and PTS/DTS values
    Output the video data.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int outvideopacket(int desiredsize,  /* size of packet to output */
    int bytesread, /* bytes read in input buffer */
    struct inparams *UserParams,
    struct calcparams *CalcParams,
    char *buf) /* input buffer associated with bytesread */
    {
    /*  WARNING THIS MODULE IS VERY CLOSE COUPLES WITH cdivideopacket, sorry */
    static int firstpicture = 1; /* if set then have not read the
                                     first picture */
    char *AUptr; /* pointer to a found access unit */
    int tmp,
        bytecount,
        numtostuff,     /* number of stuffing bytes required */
        picttype,
        picsinbuf,      /* controls how much to update timestamps */
        i;
    longdelay;  /* number of clock periods to hold a AU before presenting
                    to user, only valid for I and P frames */

    outbits(PACKETSTARTCODE,0,NULL,24,23);
    outbits(UserParams->videostreamid,0,NULL,8,7);

    /* the packet length is fixed in all cases, stuffing bytes are use to
    accomplish this.  The lenght is that following this field (so -6)
    from total  */
    tmp = desiredsize - 6;
    outbits(tmp, 0,NULL,16,15);

    /* Look into the read in buffer, if there is an AU within this */
    /* buffer then we need to output a PTS value that corresponds to it */
    /* Also need to add the necessary byte stuffing */

    /* this used to be a call to strstrn... */

    AUptr = countPICTS(UserParams,CalcParams,
```

```
        buf,bytesread,VIDEOAU,SIZEOFVIDEOAU,&picsinbuf);

    if (AUptr != NULL)
        picttype = getpicturetype(AUptr);
    else
        {
        picttype = NOTPICT;  /* not a picture */
        }


    if ( (picttype == IFRAME)||
        (picttype == BFRAME) ||
        (picttype == PFRAME) )
        {
        /* Now determine what type of picture this is, if I or P then */
        /* need to encode both PTS and DTS */
        /* otherwise only need to one of them (PTS).*/
        /* AUptr points to the beginning of this thing so */
        switch(picttype)
            {
            case IFRAME: /* I frame */
#ifdef DEBUG
printf("\nIFRAME\n");
#endif
            case PFRAME: /* P frame */
#ifdef DEBUG
printf("\nPFRAME\n");
#endif
                /* First output the stuffing bytes */
                /* since both PTS and DTS go out (10 bytes) */
                /* the header is always 18 bytes */
                /* start + id = 3 bytes + length +5(pts) +5(dts) =16 */
                /* So need to stuff desiredsize - (16 + bytesread) */

                tmp = desiredsize - (16 +2+ bytesread);
                if (tmp > 16)
                    fprintf(stderr,"Video stuffing more then 16 bytes!\n");
                for(i=0;i<tmp;i++)
                    {
                    outbits(STUFFINGBYTE,0,NULL,8,7);
                    }

                /* STD buffer scale and size  */
                outbits(0x01L,0,NULL,2,1);
                outbits(0x01L,0,NULL,1,0);
                outbits(46L,0,NULL,13,12);

                /* Now the PTS and DTS values */
                if (!firstpicture)
                    {
                    delay = CalcParams->videoPTS +CalcParams->videoPTSdelay;
                    }
                else
                    {
                    delay = CalcParams->videoPTS + CalcParams->videoPTSconst;
                    }
#ifdef DEBUG
/*                SCR     PTS     DTS */
if (getpicturetype(AUptr) == IFRAME)
    printf("\nI\t%10lu\t%10lu\t%10lu\n",
        CalcParams->SCR,delay,CalcParams->videoPTS);
else
    printf("\nP\t%10lu\t%10lu\t%10lu\n",
        CalcParams->SCR,delay,CalcParams->videoPTS);
#endif

                if ((delay < CalcParams->SCR)||(CalcParams->videoPTS < CalcParams->SCR))
                    {
                    /* Sanity check type stuff, if this happens well its probably a
                    software bug or math error...? */
                    if (delay < CalcParams->SCR)
                        fprintf(stderr,"Oops SCR %d already greater then video PTS %d (%d)\n",
CalcParams->SCR,delay,CalcParams->SCR-delay);
                    if (CalcParams->videoPTS < CalcParams->SCR)
                        fprintf(stderr,"Oops SCR %d already greater then video DTS %d (%d)\n",
CalcParams->SCR,CalcParams->videoPTS,CalcParams->SCR - CalcParams->videoPTS);
```

```
                    }

                outbits(0x03L,0,NULL,4,3);   /* output the first markers */
                outbits(0x00L,0,NULL,1,0);   /* this is actually the first
                    PTS bit, but since I don't have 33 bit ints... */
                outbits(delay,0,NULL,2,31);
                outbits(MARKERBIT,0,NULL,1,0);
                outbits(delay,0,NULL,15,29);
                outbits(MARKERBIT,0,NULL,1,0);
                outbits(delay,0,NULL,15,14);
                outbits(MARKERBIT,0,NULL,1,0);

                /* The DTS values */
                outbits(0x01L,0,NULL,4,3);   /* output the first markers */
                outbits(0x00L,0,NULL,1,0);   /* this is actually the first
                    PTS bit, but since I don't have 33 bit ints... */
                outbits(CalcParams->videoPTS,0,NULL,2,31);
                outbits(MARKERBIT,0,NULL,1,0);
                outbits(CalcParams->videoPTS,0,NULL,15,29);
                outbits(MARKERBIT,0,NULL,1,0);
                outbits(CalcParams->videoPTS,0,NULL,15,14);
                outbits(MARKERBIT,0,NULL,1,0);

                /* Now the actual data */
                outbits(0L,1,buf,bytesread*8,7);

                break;

            case BFRAME: /* B frame */
#ifdef DEBUG
printf("\nBFRAME\n");
#endif
                /* First output the stuffing bytes */
                /* since only PTS is sent out (5 bytes) */
                /* okay send out the stuffing bytes */
                /* Here the header still need to be 18 bytes */
                /* so 3 + 1 + 2 + 5 -> add 7 bytes to stuffing */
                tmp = desiredsize - (bytesread + 11 +2);
                if (tmp > 16)
                    fprintf(stderr,"Video stuffing more then 16 bytes!\n");
                for(i=0;i<tmp;i++)
                    {
                    outbits(STUFFINGBYTE,0,NULL,8,7);
                    }

                /* STD buffer scale and size */
                outbits(0x01L,0,NULL,2,1);
                outbits(0x01L,0,NULL,1,0);
                outbits(46L,0,NULL,13,12);

#ifdef DEBUG
/*                SCR     PTS     DTS */
printf("\nB\t%10lu\t%10lu\t%10lu\n",
    CalcParams->SCR,CalcParams->videoPTS,CalcParams->videoPTS);
#endif
                if (CalcParams->videoPTS < CalcParams->SCR)
                    {
                    fprintf(stderr,"Oops SCR %d already greater then video PTS %d (%d)\n",
CalcParams->SCR,CalcParams->videoPTS,CalcParams->SCR - CalcParams->videoPTS);
                    }

                /* PTS = DTS since its a B picture */
                outbits(0x02L,0,NULL,4,3);   /* output the first markers */
                outbits(0x00L,0,NULL,1,0);   /* this is actually the first
                    PTS bit, but since I don't have 33 bit ints... */
                outbits(CalcParams->videoPTS,0,NULL,2,31);
                outbits(MARKERBIT,0,NULL,1,0);
                outbits(CalcParams->videoPTS,0,NULL,15,29);
                outbits(MARKERBIT,0,NULL,1,0);
                outbits(CalcParams->videoPTS,0,NULL,15,14);
                outbits(MARKERBIT,0,NULL,1,0);

                /* Now the actual data */
                outbits(0L,1,buf,bytesread*8,7);

                break;
```

```
            default:
                for (i=-10;i<50;i++)
                    fprintf(stderr,"%2.2x ",0x00FF & AUptr[i]);
                fprintf(stderr,"this does not look like a picture !!!\n");
                exitroutine();

            }
        firstpicture = 0;    /* Then have read the very first AU/picture */

        }
    else
        {
#ifdef DEBUG
printf("\nNO AU\n");
#endif
        /* No AU found within packet just output the marker and the data */
        /* Again the header must be 18 bytes */
        /* so this implies add 11 stuffing bytes */
        tmp = desiredsize - (bytesread + 7 +2);
        if (tmp > 16)
            fprintf(stderr,"Video code stuffing more then 16 bytes!\n");
        for(i=0;i<tmp;i++)
            {
            outbits(STUFFINGBYTE,0,NULL,8,7);
            }

        /* STD buffer scale and size  */
        outbits(0x01L,0,NULL,2,1);
        outbits(0x01L,0,NULL,1,0);
        outbits(46L,0,NULL,13,12);

        outbits(0x0FL,0,NULL,8,7);   /* else clause in mpeg
                                         spec (not PTS or DTS) */
        outbits(0L,1,buf,bytesread*8,7);
        }

    /* Update the next PTS value based on how many AU's are within this
    current buffer */
    CalcParams->videoPTS = CalcParams->videoPTS +
        CalcParams->videoPTSconst* picsinbuf;
#ifdef DEBUG
printf("Video AU's is %d\n",picsinbuf);
#endif
    }

/*
{*NAME*}
    cdi_aud_sect0 -- Output a CDI audio sector 0.

{*SYNOPSIS*}
    int cdi_aud_sect0(struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Output the first sector of a VIDEO CD compliant mpeg system stream.
    VCD has the oddity of wanting a seperate system header for both
    audio and video, this one takes care of the audio system header.

{*ALGORITHM*}
    Output the pack header
    Output the system header as defined for audio in the VCD spec
    Output a padding stream to fill in the rest of the sector.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
```

```
    All rights reserved.
{**}
*/
int cdi_aud_sect0(struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    /* Output CDI compliant mpeg audio sector with SYSTEM header */

    int i;

#ifdef DEBUG
    printf("\ncdi_aud_sect0\n");
#endif

    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */


    /* SCR value */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit, but
                         since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);

    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);

    /* SYSTEM header as stated in the white book figure IV.10 */
    outbits(SYSTEMHEADERSTARTCODE,0,NULL,32,31);
    outbits(9L,0,NULL,16,15);        /* system header length's always fixed,
                                     since only have 1 only and 1 video) */
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(UserParams->muxrate,0,NULL,22,21);      /* rate bound */
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(1L,0,NULL,6,5);          /* audiobound */
    outbits(UserParams->fixed_flag,0,NULL,1,0);             /* fixed_flag */
    outbits(UserParams->CSPS_flag,0,NULL,1,0);              /* CSPS */
    outbits(UserParams->audio_lock,0,NULL,1,0);             /* audio_lock */
    outbits(UserParams->video_lock,0,NULL,1,0);             /* video_lock */

    outbits(MARKERBIT,0,NULL,1,0);
    outbits(0L,0,NULL,5,4);      /* video bound */
    outbits(0xFFL,0,NULL,8,7);   /* reserved byte */

    /* the std bounds for audio */
    outbits(UserParams->audiostreamid,0,NULL,8,7);  /* audio stream id */
    outbits(0x03L,0,NULL,2,1);
    outbits(0x00L,0,NULL,1,0);   /* STD buffer scale, indicates scale of video */
    outbits(32L,0,NULL,13,12);   /* CSPS is set this in constrained to 4k */

    /* Padding packet of 2277 bytes to fill out the rest of the sector */
    outputpadding(2277L, UserParams, CalcParams);

    /* Now the zeros field, whatever the hell they are... they sure
    aren't MPEG!  Twenty of them */
    if (UserParams->VCD_nulls)
        for (i=0;i<20;i++)
            outbits(0x00L,0,NULL,8,7);
    }

/*
{*NAME*}
    cdi_aud_sect -- Output CDI compliant audio sectors, other then first one.

{*SYNOPSIS*}
    int cdi_aud_sect(struct inparams *UserParams,
        struct calcparams *CalcParams)

{*CLASS*}
    c:mpeg
```

```
{*DESCRIPTION*}
    Create a systems pack that contains audio data.

{*ALGORITHM*}
    Output a pack header
    Go and output the audio data
    If VCD type stuff pad the end of the thing with 20 NULLS.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int cdi_aud_sect(struct inparams *UserParams,
    struct calcparams *CalcParams)
    {
    /* Output CDI compliant mpeg audio sector (without SYSTEM header) */

    int i;

#ifdef DEBUG
    printf("\ncdi_aud_sect\n");
#endif


    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */


    /* SCR value */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit,
                                 but since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);

    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);


    /* Now the actual audio data */
    cdiaudiopacket( UserParams, CalcParams);

    /* Now the zeros field, whatever the hell they are... they sure
    aren't MPEG!  Twenty of them, might as well be a million...
    I believe this can just be ignored in all processing even
    the effect it has on the buffering? */

    if (UserParams->VCD_nulls)
        {
        for (i=0;i<20;i++)
            outbits(0x00L,0,NULL,8,7);
        }
    }

/*
{*NAME*}
    cdiaudiopacket -- Output a CDI compliant audio packet.

{*SYNOPSIS*}
    static int cdiaudiopacket(struct inparams *UserParams,
        struct calcparams *CalcParams)
```

```
{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    First level routine to actaully create a audiopacket.

{*ALGORITHM*}
    Read the data from the audio stream.
    Based on the amount read go and output it.
    If at EOF output a systems end code.

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int cdiaudiopacket(struct inparams *UserParams,
        struct calcparams *CalcParams)
    {
    unsigned char buf[4048]; /* just allocated enough for now, avoid
                             dynamic allocation for potential real time app */
    int bytecount,
        numtostuff,      /* number of stuffing bytes required */
        bytesread,       /* number of bytes read from input */
        tmp,
        i;

#ifdef DEBUG
fprintf(stdout,"\naudiopacket\n");
#endif


    /* Try to read in enough data for one audio packet.  If I can't then
    add padding packets to fill out this pack with another padding packet.
    (This implies that partial audio packets need another padding packet
    right next to it) */
    if (!feof(UserParams->audio)&&
        (bytesread = fread(buf,1,UserParams->auddatasize,UserParams->audio)) == UserParams-
>auddatasize)
        {
        if(UserParams->CDI_type)
            outaudiopacket(UserParams->audpacketsize-20,bytesread,UserParams,CalcParams,buf);
        else
            outaudiopacket(UserParams->audpacketsize,bytesread,UserParams,CalcParams,buf);

        }
    else
        {
        /*  So have read < 2279, the problem here is that an ISOEND
        code must be tagged to this 'partial packet', and we need
        even byte boundaries (white book 3.2.4).
        So here are some senario's:
            Bytesread       What to do
            2276-2278      Don't know, can't fit an end code.  And the
                             white book implies only that in sectors
                             in which an end code show up can I exceed
                             13 bytes for the packet headers (stuffing bytes)
                             For now I think I will just send 2275 bytes and then
                             and endcode... this probably will cut off any
                             audio endcode but at least there's an system
                             endcode that I'm tacking on.

            2275        Output the packet, then send the endcode
                             (2275+4 = 2279 + 13 = 2292)

            2259-2274        Output the packet as 2275, the output routine will
                             stuff bytes.  Then just send the endcode.  This
                             range takes into consideration the maximun stuffing
                             allowed by MPEG1
```

```
            0-2258          Audio packet size = bytesread +
                            13 + (15 - bytesread%16)
                            padding packet size = 2288 - audio packet size
                            This math will result in at most two packets,
                            one audio the other padding that are both even bytes
                            aligned and can then be output.  The remaining 4
                            bytes are used for the ISOendcode.  Oh, the only
                            reason that padding packets are done here is that
                            I can now do them since 2288-bytesread now gives
                            me enough space to create a padding packet
        */
        /* Again I will leave this alone for now.  Worse case is that
        I have mucked up the EOF of a stream (hardware hopefully will recover) */
        if (bytesread > 2275)
            bytesread = 2275;    /* This is the make shift thing that drops the
                        extra audio data since I can't do anything with it */

        if (bytesread >= 2259)
            {
            /* Just the audio packet is sent, the end code is followed
            directly by this packet */
            outaudiopacket(2288,bytesread,UserParams,CalcParams,buf);
            }
        else
            {
            /* Both an audio packet and a padding packet is sent, again
            both must align to even byte boundaries */
            int audsize, paddsize;
            audsize = bytesread + 13 + (15 - bytesread%16);
            paddsize = 2288 - audsize;
            outaudiopacket(audsize,bytesread,UserParams,CalcParams,buf);
            outputpadding(paddsize,UserParams,CalcParams);
            }

        /* Output an ISO-end code */
        if (!UserParams->runforever)
            outbits(ISOENDCODE,0,NULL,32,31);
        else
            /* this is moderately safe... since a decoder should ignore
            these leading things anyways */
            outbits(0L,0,NULL,32,31);

        }


    }



/*
{*NAME*}
    outaudiopacket -- Acutal output routine for CDI compliant audio packets

{*SYNOPSIS*}
    static int outaudiopacket(int desiredsize,  \* acutal size of packet *\
        int bytesread,   \* bytes available in input buffer, audio bytes *\
        struct inparams *UserParams,
        struct calcparams *CalcParams,
        char *buf)  \* acutal audio data read, associated with bytes read*\

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    Actual work routine for outputting a audio packet (ie one that
    contains audio data).

{*ALGORITHM*}
    Output the packet headers
    Determine the number of AU's within the audio data.
    Increment the PTS values for audio based on the number of AU's contained.
    Add any stuffing necessary to fill out the packet to the size required.
    Ouput the PTS values (audio has no DTS values).
    Output the data.
```

```
{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
static int outaudiopacket(int desiredsize,  /* acutal size of packet */
    int bytesread,   /* bytes available in input buffer, audio bytes */
    struct inparams *UserParams,
    struct calcparams *CalcParams,
    char *buf)  /* acutal audio data read, associated with bytes read*/
    {
    char *AUptr;
    int tmp,i;

    outbits(PACKETSTARTCODE,0,NULL,24,23);
    outbits(UserParams->audiostreamid,0,NULL,8,7);

    /* THE NORMAL case:
    In the audio packets that take up the whole sector the size is
    fixed to be 2292 bytes/packet - 6 = 2286, the 6 comes from the
    startcodelen + idlen + packetlenlen.  Actually I use the desired
    size to control the size  */

    tmp = desiredsize - 6;

    outbits( tmp, 0,NULL,16,15);

    /* Look into the read in buffer, if there is an AU within this */
    /* buffer then we need to output a PTS value that corresponds to it */
    /* Also need to add the necessary byte stuffing */
if (1)
        {
        /* Stuff bytes to make up the difference in the amount of data
        supplied and the amount to send out. WARNING there is a maximum
        allowed number of stuffing bytes allowed.  A serious warning message
        appears.  What to do about it I don't know.  But maybe try to
        touch up the size of the audio file, since this seems to
        be a screw up in the white book specification */
        tmp = desiredsize - (6+5+2+bytesread);
        if (tmp > 16)
            fprintf(stderr,
            "Audio code is stuffing more then 16 bytes!  MPEG1 violation!\n");
        for(i=0;i<tmp;i++)
            outbits(STUFFINGBYTE,0,NULL,8,7);

        /* STD buffer scale and size */
        outbits(0x01L,0,NULL,2,1);
        outbits(0x00L,0,NULL,1,0);
        outbits(32L,0,NULL,13,12);


#ifdef DEBUG
printf("\nA\t%10lu\t%10lu\t%10lu\n",
    CalcParams->SCR,CalcParams->audioPTS,CalcParams->audioPTS);
#endif

        if (CalcParams->audioPTS < CalcParams->SCR)
            {
            fprintf(stderr,"Oops SCR %d already greater then audio PTS %d (%d) \n",
CalcParams->SCR,CalcParams->audioPTS, CalcParams->SCR - CalcParams->audioPTS);
            }

        outbits(0x02L,0,NULL,4,3);   /* output the first markers */
        outbits(0x00L,0,NULL,1,0);   /* this is actually the first PTS bit,
                                     but since I don't have 33 bit ints... */
        outbits(CalcParams->audioPTS,0,NULL,2,31);
        outbits(MARKERBIT,0,NULL,1,0);
        outbits(CalcParams->audioPTS,0,NULL,15,29);
        outbits(MARKERBIT,0,NULL,1,0);
```

```
        outbits(CalcParams->audioPTS,0,NULL,15,14);
        outbits(MARKERBIT,0,NULL,1,0);

        /* Update the next audioPTS value */
        CalcParams->audioPTS = CalcParams->audioPTS +
            CalcParams->audioPTSconst *
            (countAudioAUs(UserParams,CalcParams,
                buf,bytesread,AUDIOAU,SIZEOFAUDIOAU));
#ifdef DEBUG
printf("Audio AU's is %d\n",
countAudioAUs(UserParams,CalcParams, buf,bytesread,AUDIOAU,SIZEOFAUDIOAU));
#endif
        }
    else
        {
        /* No AU found within buffer So don't need a PTS Value */

#ifdef DEBUG
printf("\nAUDIO no AU\n");
#endif

        /* Stuff bytes to make up the difference in the amount of data
        supplied and the amount to send out. Also take into consideration
        that PTS values are not sent
        WARNING there is a maximum
        allowed number of stuffing bytes allowed.  A serious
        warning message results... */
        tmp = desiredsize - (6+2+bytesread+1);
        if (tmp > 16)
            fprintf(stderr,
            "Audio code is stuffing more then 16 bytes!  MPEG1 violation!\n");
        for(i=0;i<tmp;i++)
            outbits(STUFFINGBYTE,0,NULL,8,7);

        /* okay send out the stuffing bytes */
        /* since each packet header must be 13 bytes and a PTS
        takes up 5 bytes ( startcode + id + len + stuff + 2 + 1 = 13 )
        => stuff = 4 */
        for(i=0;i<4;i++)
            {
            outbits(STUFFINGBYTE,0,NULL,8,7);
            }

        /* If odd packet length stuff to even */
        if ((desiredsize % 2) == 1)
            outbits(STUFFINGBYTE,0,NULL,8,7);

        /* STD buffer scale and size */
        outbits(0x01L,0,NULL,2,1);
        outbits(0x00L,0,NULL,1,0);    /*STD buffer scale */
        outbits(32L,0,NULL,13,12);    /*STD buffer size */

        /* output the 0000 1111 byte since no PTS values*/
        outbits(0x0FL,0,NULL,8,7);

        }

    /* Now can output the actual data */
    outbits(0L,1,buf,bytesread*8,7);
    }



/*
{*NAME*}
    countPICTS -- Given a buffer count the number of pictures within it.

{*SYNOPSIS*}
    char *countPICTS( struct inparams *UserParams,
                struct calcparams *CalcParams,
                char *buf,   \* Buffer to search *\
                int sizebuf, \* Size of buffer *\
                char *token, \* token to find in buf *\
                int sizetok,     \* size of token to search for *\
                int *numpics)    \* number of pictures within buffer *\
```

```
{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    The primary purpose of this routine is to determine how many
    access units occur within a given buffer.  This can then be
    used to determine the amount to increase the PTS/DTS values
    for later time stamps (ie if there are two pictures within
    this buffer then the next timestamp must be two pictures
    later...).

{*ALGORITHM*}
    Zip through the entire buffer, don't forget that according to the
    definition a AU occurs if the first byte is contained within the
    buffer (so need to do some lookahead). This would normally take
    only 3 extra bytes of lookahead but since for confirmation I
    need to check the picture type I really need 5 bytes (picture type
    fall within bytes 5 after a startcode).

{*SEE ALSO*}

{*BUGS*}
    This routine assumes a minimum size to allow one token to be present.
    At this point in time this is set to 200 bytes.  I do this to make
    life a bit easier on programming (ie quick hack).  Plus this should
    be reasonable assuming the normal MPEG-1 streams that are processed.

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
char *countPICTS( struct inparams *UserParams,
            struct calcparams *CalcParams,
            char *buf,   /* Buffer to search */
            int sizebuf, /* Size of buffer */
            char *token, /* token to find in buf */
            int sizetok,     /* size of token to search for */
            int *numpics)    /* number of pictures within buffer */
    {
    char *firstpict = NULL;
    char *ptr;
    int ret = 0,
        i,j,pictype;


    *numpics = 0;

    /* Add 3 bytes so that can detect boundary condition for buffer */
    /* Okay the code really adds 5, need the other 2 bytes so that
    I can determine the picture type, otherwise I will have notread
    the bytes containing the picture type... this only applies
    when the picture start code spans the buffer boundary, with 1 byte
    in the first and 3 bytes in the second, I don't modify the
    search range since still only need to search until the 3 bytes - tok size */
    if (!feof(UserParams->video)&&
        fread(&buf[sizebuf],1,5,UserParams->video) == 5)
        sizebuf += 3;    /* pass by value so can modify it here (temporarily) */

    for(i=0;i<sizebuf-sizetok;i++)
        {
        if (buf[i] == token[0])
            if (buf[i+1] == token[1])
                if (buf[i+2] == token[2])
                    if (buf[i+3] == token[3])
                        {
                        pictype = getpicturetype(&buf[i]);
                        if ((pictype == IFRAME) |
                            (pictype == PFRAME) |
                            (pictype == BFRAME))
                            {
                            (*numpics)++;
                            if (firstpict == NULL)
```

```
                                firstpict = &(buf[i]);
                            }
                        else
                            {
                            /* ERROR condition */
                            /* not conviced that this is an error bq 6/14*/
                            int k;
                            fprintf(stderr,"PICTURESTART code emulation\n");
                            fprintf(stderr,"%2.2x %2.2x %2.2x %2.2x %2.2x %2.2x\n\n",
                            0x00ff&buf[i],0x00ff&buf[i+1],0x00ff&buf[i+2],
                            0x00ff&buf[i+3],0x00ff&buf[i+4],0x00ff&buf[i+5]);

                            for(k=0;k<sizebuf;k++)
                                {
                                fprintf(stderr,"%2.2x ",0x00ff&buf[k]);
                                if (k%20 == 0)
                                    fprintf(stderr,"\n");
                                }
                            fprintf(stderr,"\n");
                            }
                        }
        }
    /* remove the boundary bytes from the search buffer */
    /* again remember actually read 5 extra bytes */
    if (!feof(UserParams->video)&&fseek(UserParams->video,-5,1)!=0)
        fprintf(stderr,"Error in fseeking back 5 bytes\n");

    return(firstpict);
    }


/*
{*NAME*}
    countAudioAUs -- Count audio AU's within a given audio buffer.

{*SYNOPSIS*}
    int countAudioAUs( struct inparams *UserParams,
        struct calcparams *CalcParams,
        unsigned char *buf,  \* Buffer to search *\
        int sizebuf, \* Size of buffer *\
        char *token, \* token to find in buf *\
        int sizetok)     \* size of token to search for *\

{*CLASS*}
    c:mpeg

{*DESCRIPTION*}
    This one does the same as countPICTS but for a audio buffer,
    the primary difference is lack of checking for the picture type.

{*ALGORITHM*}
    Similar to countPICTS

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    HEURIS Logic -- Brian Quandt

{*COPYRIGHT*}
    Copyright 1994 Heuris Logic.
    All rights reserved.
{**}
*/
int countAudioAUs( struct inparams *UserParams,
    struct calcparams *CalcParams,
    unsigned char *buf,  /* Buffer to search */
            int sizebuf, /* Size of buffer */
            char *token, /* token to find in buf */
            int sizetok)     /* size of token to search for */
    {
    unsigned char *ptr;
    int ret = 0,
        i,j,
        numAUs;
```

```
    numAUs = 0;

    /* Add 3 more bytes from the input stream to the search buffer this
    is to accomodate AU's that may span the boundary */
    if (!feof(UserParams->audio)&&
        fread(&buf[sizebuf],1,3,UserParams->audio) == 3)
        sizebuf += 3;    /* pass by value so can modify it here (temporarily) */


    for(i=0;i<sizebuf-sizetok;i++)
        {
        /* Okay so I ignore the size of the token and the acutal value,
        this needs to be changed to be modifiable via the proper
        defines, but now this was just a quick hack (as always) */
        /* the pattern that we need to match for an audio AU in the
        case of layer II is  1111 1111 1111 110x  layer I audio
        is of course allowed but not for this CDI type coding */
        if (buf[i] == 0xFF)
            if ((buf[i+1] & 0xFE) == 0xFC)
                    {
                    numAUs++;
                    }
        }

    /* remove the boundary bytes from the search buffer */
    if (!feof(UserParams->audio)&&fseek(UserParams->audio,-3,1)!=0)
        fprintf(stderr,"Error in fseeking back 3 bytes\n");

    return(numAUs);
    }


int cdi_pad_sect(struct inparams *UserParams,
    struct calcparams *CalcParams)
    {

#ifdef DEBUG
    printf("\ncdi_pad_sect\n");
#endif


    outbits(PACKSTARTCODE,0,NULL,32L,31L);          /* pack start */
    outbits(0x02L,0,NULL,4,3);                      /* 0010 */


    /* SCR value */
    outbits(0x00L,0,NULL,1,0);   /* this is actually the first SCR bit,
                                 but since I don't have 33 bit ints... */
    outbits(CalcParams->SCR,0,NULL,2,31);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,29);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(CalcParams->SCR,0,NULL,15,14);
    outbits(MARKERBIT,0,NULL,1,0);
    outbits(MARKERBIT,0,NULL,1,0);

    /* MUXRATE */
    outbits(UserParams->muxrate,0,NULL,22,21);      /* muxrate */
    outbits(MARKERBIT,0,NULL,1,0);

    /* I use video packet size here, just becuase I think I will then be CDI tolerable?? */
    outputpadding(UserParams->vidpacketsize,UserParams,CalcParams);
    }
```

A.2.3 cdisys.p

```
    int main(int argc, char *argv[]);

    int ProcessStreams(struct inparams *UserParams, /* User parameters */
        struct calcparams *CalcParams); /* Calculated Parameters */

    static int outputemptypack(int totallength, /* lenght of a pack */
```

```
        struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int outputpadding(int totallength, /* lenght of a padding stream */
        struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int GetParams(int argc, char *argv[],
        struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int ReadUserParams(int argc, char *argv[],
        struct inparams *UserParams);

    int getpicturetype(char *ptr); /* pointer to picture start code */

    static int exitroutine();

    int cdi_vid_sect0(struct inparams *UserParams,
        struct calcparams *CalcParams);

    int cdi_vid_sect(struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int cdivideopacket(struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int outvideopacket(int desiredsize,  /* size of packet to output */
        int bytesread, /* bytes read in input buffer */
        struct inparams *UserParams,
        struct calcparams *CalcParams,
        char *buf); /* input buffer associated with bytesread */

    int cdi_aud_sect0(struct inparams *UserParams,
        struct calcparams *CalcParams);

    int cdi_aud_sect(struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int cdiaudiopacket(struct inparams *UserParams,
        struct calcparams *CalcParams);

    static int outaudiopacket(int desiredsize,  /* acutal size of packet */
        int bytesread,     /* bytes available in input buffer, audio bytes */
        struct inparams *UserParams,
        struct calcparams *CalcParams,
        char *buf); /* acutal audio data read, associated with bytes read*/

    char *countPICTS( struct inparams *UserParams,
                struct calcparams *CalcParams,
                char *buf,    /* Buffer to search */
                int sizebuf,    /* Size of buffer */
                char *token,    /* token to find in buf */
                int sizetok,        /* size of token to search for */
                int *numpics); /* number of pictures within buffer */
```

A.2.4 makefile

```
#
#
# Generic Makefile -- architecture level
# This is a template makefile that is used in making Heuris Logic projects
#
#
#




#
# Compiler and linker options
#
CC = gcc
```

```
CFLAGS = -c -I.
#CFLAGS = -c -I. -DDEBUG_BUFFVID
#CFLAGS = -c -I. -DDEBUG_BUFFAUD
#CFLAGS = -c -I.
LINK = $(CC)
LDFLAGS = -o
#LIBS =


#
# Basic rules
#
.c.o:
    $(CC) $(CFLAGS) $<


#
# Definitions
#
MAKEBIN = gnumake
TRANSPORTFILE = cdisys.lha # Archive to create when making a distribution
ARCHIVER = lha # Archiver, lha or tar or whatever
ARCH_OPS = a # archiver options
MAKEFILE = Makefile.cdisys.portable #actual name of makefile in RCS
BINARYNAME = cdisys
PORTABLEDIR = ../portable
INSTALLDIR = /home/heuris/bin
INSTALLMODE = 755


#
# List of all source files, include files, pictures etc
#
CFILES =     cdisys.c \
             outbits.c\
             strstrn.c

INCFILES =   outbits.p \
             cdisys.p \
             strstrn.p \
             bits.h
OTHERFILES =
OBJS = $(CFILES:.c=.o)
DISTFILES =  $(CFILES) \
             $(INCFILES) \
             $(OTHERFILES)

all:
    @echo "$(MAKEBIN) [dist|clean|objects|depend|binary]"

depend: $(CFILES)
    mkdep $(CFLAGS) $(CFILES)

#
# Create the binary
#
binary: $(BINARYNAME)

$(BINARYNAME): objects
    $(LINK) $(LDFLAGS) $@ $(OBJS) $(LIBS)


#
# Update all the necessary objects
#
objects: $(OBJS)

#
# Commands to create distribution media so that code can be compiled elsewhere
#
dist: clean makedist

makedist: $(DISTFILES)
    $(ARCHIVER) $(ARCH_OPS) $(TRANSPORTFILE) $? Makefile

install:
```

```
    cp $(BINARYNAME) $(INSTALLDIR)
    chmod $(INSTALLMODE) $(INSTALLDIR)/$(BINARYNAME)
#
# Clean everthing up
#
clean:
    rm -i $(DISTFILES) $(TRANSPORTFILE) $(OBJS)


#####
# include dependencies
# If no support on system to auto gen these then create by hand.
#####
#include .depend
```

A.2.5 outbits.c

```
/*
{*NAME*}
    __FILE__ -- Routines for outputing bits.

{*SYNOPSIS*}
    __FILE__

{*CLASS*}
    files:generic

{*DESCRIPTION*}
    These are a collection of utilities for sending out bits.  The
    idea is to send a word/byte/string and have this software
    worry about actually putting the bits out.

    There are also marker routines that allow you to keep count
    of how many bits thus far sent, with the ability to zero the
    user defined counters.

{*ENVIRONMENT*}
    None

{*SYSTEM REQUIREMENTS*}
    None

{*COMPILATION INSTRUCTIONS*}
    gcc -c __FILE__
    If DEBUG is defined then the outbits routine will output HEX text.

{*RESOURCE CONTROL*}
    $Id: outbits.c,v 1.4 1994/02/28 17:49:13 quandt Exp $
    $Log: outbits.c,v $
 * Revision 1.4  1994/02/28  17:49:13  quandt
 * func:cont
 * File now complies to HEURIS prototype standards (still needs more docs
 * though)
 *

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}
    Brian Quandt

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/

#include <outbits.p>
#include <stdio.h>
#include <bits.h>


extern FILE *outfile;
```

```
/* these modules should proably go in to there own file */
static int bitcount=0;   /* this is total byte count of bytes sent out, cannot be altered
(zeroed) */


/*
{*NAME*}
    outbits -- Routine to send out bits (one byte at a time).

{*SYNOPSIS*}
    int outbits(int val,     \* If whichin is 0 then read thi int as input *\
            int whichin,     \* controls input from val or ptr *\
            char *ptr,       \* if whichin is !0 read this string as input *\
            int howmany,     \* how many bits to read from the input *\
            int startbit)    \* starting bit to start reading from *\

{*CLASS*}
    c:bytes

{*DESCRIPTION*}

{*ENVIRONMENT*}

{*SYSTEM REQUIREMENTS*}

{*ALGORITHM*}

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/
int outbits(int val,     /* If whichin is 0 then read thi int as input */
        int whichin,     /* controls input from val or ptr */
        char *ptr,       /* if whichin is !0 read this string as input */
        int howmany,     /* how many bits to read from the input */
        int startbit)    /* starting bit to start reading from */
    {
    /* If ptr is NULL then a 32 bit int was passed to this function
    howmany controls how many bits to read from this int, startbit
    determines the place in the int to start.

    whichin controls which input to use the 32bit int or the byte pointer

    For example
        a call with the following params
        (10,NULL,14,15)
        will take the number 10 as a 32 bit int 0x000A and start
        with bit number 15 and output 14 bits so the output is
        '00000000000010'

    If the number to output goes past the rightmost bit (2^0) then
    zero's are padded.  The same is true if you want to start to the
    left of the the left most (2^31).

    The char *ptr variable is used as an alternate way to send
    long 'strings' of things to send out.  If not NULL this
    takes precedence over the int 'val'. */


    int i,j;
    static char buf;
    static int bitct=0;  /* counter of how many currently in buffer */

    if ( whichin != 1)
        {
        i = 0;
        while (i<howmany)
            {
            while((bitct != 8)&&(i < howmany))
```

```
                {
                if (BITSET(val, (startbit-i)))
                    SETBIT(buf,7-bitct);
                else
                    UNSETBIT(buf,7-bitct);

                bitct++; i++;
                }
            if (bitct == 8)
                {
                /* output the byte */
#ifdef DEBUG
                fprintf(stdout,"%2.2x ",0x00FF & buf);
#else
                fwrite(&buf,1,1,outfile);
#endif
                bitct = 0;
                }
            }

        }
    else
        {
#ifdef DEBUG
        fprintf(stdout,"\n\t\tDATA %d bytes\n",howmany/8);
#else
        fwrite(ptr,(howmany/8),1,outfile);
#endif
        }

    /* Update our counters */
    bitcount = bitcount + howmany;   /* update the total bit count */

    }



/* routines for user counters */
/* The user must supply a place to hold temporary counts, the user
should not and does not need to do anything with this variable */

/* userbitcount is used interally to hold the state of the static
counter when markbits was first called */

/* WARNING these counters are 32 bit counters only, I have NOT coded them
to work around the 32 bit boundaries (this probably should be changed) */

/*
{*NAME*}
    markbits -- Return the current bit count (used to zero the counter).

{*SYNOPSIS*}
    int markbits(int *userbitcount ) \*  Read current bit count *\

{*CLASS*}
    c:bytes

{*DESCRIPTION*}

{*ENVIRONMENT*}

{*SYSTEM REQUIREMENTS*}

{*ALGORITHM*}

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/
```

```
int markbits(int *userbitcount ) /*  Read current bit count */
    {
    /* Set the counter to current number of bits output */
    *userbitcount = bitcount;
    }

/*
{*NAME*}
    getUserbitcount -- Return the number of bits since a markbit call.

{*SYNOPSIS*}
    int getUserbitcount(
        int *userbitcount) \* return bit count since markbit call *\

{*CLASS*}
    c:bytes

{*DESCRIPTION*}

{*ENVIRONMENT*}

{*SYSTEM REQUIREMENTS*}

{*ALGORITHM*}

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/
int getUserbitcount(int *userbitcount) /* return bit count since markbit call */
    {
    /* return the total number of bits sent to output since the
    last call to markbits */

    return (bitcount - *userbitcount);
    }

/*
{*NAME*}
    getbitcount -- Return total number of bits output since the beginning.

{*SYNOPSIS*}
    int getbitcount(void)

{*CLASS*}
    c:bytes

{*DESCRIPTION*}

{*ENVIRONMENT*}

{*SYSTEM REQUIREMENTS*}

{*ALGORITHM*}

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/
int getbitcount(void)
    {
    /* Return total number of bits sent to output */
```

```
    return bitcount;
    }
```

A.2.6 outbits.p

```
    int outbits(int val,     /* If whichin is 0 then read thi int as input */
            int whichin,     /* controls input from val or ptr */
            char *ptr,         /* if whichin is !0 read this string as input */
            int howmany,     /* how many bits to read from the input */
            int startbit); /* starting bit to start reading from */

    int markbits(int *userbitcount ); /*  Read current bit count */

    int getUserbitcount(
        int *userbitcount); /* return bit count since markbit call */

    int getbitcount(void);
```

A.2.7 strstrn.c

```
/*
{*NAME*}
    __FILE__ -- strstrn comparison routines

{*SYNOPSIS*}
    __FILE__

{*CLASS*}
    files:generic

{*DESCRIPTION*}
    String match functions for strings of known length with internal zeroes.

{*ENVIRONMENT*}

{*SYSTEM REQUIREMENTS*}

{*ALGORITHM*}

{*COMPILATION INSTRUCTIONS*}

{*RESOURCE CONTROL*}
    $Id: strstrn.c,v 1.3 1994/05/14 00:55:51 quandt Exp quandt $
    $Log: strstrn.c,v $
 * Revision 1.3  1994/05/14  00:55:51  quandt
 * func:cont
 *
 * Revision 1.2  1994/02/24  21:45:46  quandt
 * doc.change
 * Added the #include<strstrn.p> so that we don't get proto's mixed up
 *
 * Revision 1.1  1993/12/06  21:58:05  quandt
 * Initial revision
 *

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/

#include <stddef.h>
#include <strstrn.p>


/*
```

```
{*NAME*}
    strstrn -- find a string match (including possible zero chars)

{*SYNOPSIS*}
    char *strstrn(char *ptr1, \* source string *\
        int len1,    \* length of source string *\
        char *ptr2, \* string to match *\
        int len2) \* length of string to match *\

{*CLASS*}
    c:i/o

{*DESCRIPTION*}
    Finds a match in source string for match string, if it exists.  Searches
    up to last position in source that could match second string, allowing
    zero character values within given lengths of each string.

{*ENVIRONMENT*}

{*ALGORITHM*}

{*SEE ALSO*}

{*BUGS*}

{*CREDITS*}

{*COPYRIGHT*}
    Copyright 1993 Heuris Logic.
    All rights reserved.
{**}
*/
char *strstrn(char *ptr1,
    int len1,
    char *ptr2,
    int len2)
{
    int len,start;
    char *ret=NULL;

    /* avoid any work if second string can't possibly be in first */
    if (len2 <= len1)
    {
        for(start=0,len=0; (start <= (len1-len2))&&(len!=len2); start++)
        {
            ret = &ptr1[start];
            for (len = 0; (len<len2)&&(ret[len]==ptr2[len]); len++);
        }
        if (len!=len2)
            ret = NULL;
    }
    return(ret);
}
```

A.2.8 strstrn.p

```
char *strstrn(char *ptr1, /* source string */
        int len1,    /* length of source string */
        char *ptr2, /* string to match */
        int len2); /* length of string to match */
```

A.3 Decoder

A.3.1 makefile

```
CC = gcc
YACC = byacc
```

```
LEX = flex
LEXOPT = -8

CFLAGS = -I .

BINARYNAME = sysdec

OBJS = y.tab.o lex.yy.o

all: $(BINARYNAME)

$(BINARYNAME): $(OBJS)
	$(CC) -o $(BINARYNAME) $(OBJS) -ll

y.tab.c: mpeg1sys.yac
	$(YACC) $(YACCOPT) mpeg1sys.yac

lex.yy.c: mpeg1sys.lex
	$(LEX) $(LEXOPT) mpeg1sys.lex
```

A.3.2 mpeg1sys.lex

```
/* Lex definitions */

%{
#include <y.tab.h>
%}

ANY [\x00-\xFF]

%s packet_ids

%%

"\x00\x00\x01\xBA"{ANY}{8} {
                         /* PACK */

                         BEGIN(INITIAL);
                         /* Its a pack so suck in the data for the pack */
                         /* there is a fixed amount of data here, 8bytes */

                         return(PACK_START_CODE);
                         }

"\x00\x00\x01\xB9"      {
                         /* ISO_11172_END_CODE */

                         BEGIN(INITIAL);
                         return(ISO_11172_END_CODE);
                         }

"\x00\x00\x01\xBB"{ANY}{8} {
                         /* SYSTEM_HEADER */

                         /* headers are followed by 8 bytes of data then */
                         /* it can be followed by an optional number of */
                         /* 3 bytes identifying  the stream, scale bounds etc */
                         /* this is then terminated by either a packet */
                         /* another pack or iso_11172_end_code */
                         BEGIN(packet_ids);
                         return(SYSTEM_HEADER_START_CODE);
                         }

"\x00\x00\x01"          {
                         /* PACKET */

                         BEGIN(INITIAL);
                         return(PACKET_START_CODE_PREFIX);
                         }

<packet_ids>[\x80-\xFF]{ANY}{2}  {
                         /* SYSTEM HEADER packet id's */

                         /* You can tell if you have this section if the */
                         /* first bit is a 1 -> range 0x80..0xFF */
```

```
                        /* It will also be 8 bytes long */
                        return(SYS_HEAD_IDS);
                        }
.                       {
                        return(COMPRESSED_DATA);
                        }
%%
```

A.3.3 mpeg1sys.yak

```
%{
extern char * yytext;
extern int yyleng;
%}

%token PACK_START_CODE
%token ISO_11172_END_CODE
%token SYSTEM_HEADER_START_CODE
%token PACKET_START_CODE_PREFIX
%token SYS_HEAD_IDS
%token COMPRESSED_DATA

%%

start:  stuff

stuff:  /* empty */
        | stuff validsym
;

validsym:   PACK_START_CODE
                {
                int i;
                printf("*pack*\n");
                outyytext();
                }
            | ISO_11172_END_CODE {printf("*end code*\n"); outyytext();}
            | sys_header
            | a_packet
;

a_packet:   PACKET_START_CODE_PREFIX
                {
                printf("*packet*\n");
                outyytext();
                compdata = 0;    /* initialize data count to zero */
                }
            compressed_data
                {
                printf("*compressed data*\nbytes=%d\n",compdata);
                }
;

compressed_data: /* empty */
            | compressed_data COMPRESSED_DATA {compdata++;}
;

sys_header: SYSTEM_HEADER_START_CODE {printf("*sys head*\n");outyytext();} sysids
;

sysids:     /* empty */
            | sysids SYS_HEAD_IDS {printf("*sysids*\n");outyytext();}
;

%%
#include <stdio.h>

/* counter of compressed data, packet sizes? */
int compdata;

main()
    {
    return(yyparse());
    }
```

```
yyerror(char *s)
    {
    fprintf(stderr,"%s\n",s);
    }

outyytext()
    {
    int i;
    for(i=0;i<yyleng;i++)
        printf("%2.2x ", 0x00FF & yytext[i]);
    printf("\n");
    }
```

A.2 y.tab.h

```
#define PACK_START_CODE 257
#define ISO_11172_END_CODE 258
#define SYSTEM_HEADER_START_CODE 259
#define PACKET_START_CODE_PREFIX 260
#define SYS_HEAD_IDS 261
#define COMPRESSED_DATA 262
```

Annex B
(informative)

Video - code listings

B.1 Introduction

This annex contains the variable length code tables for macroblock addressing, macroblock type, macroblock pattern, motion vectors, and DCT coefficients.

Table B.1 List of encoder files

filename	description
acvlc.c	VLC generation for dct_coef_next table
bitcount.h	VLC table entry count
consts.h	constants used throughout encoder program
convert.c	picture format and sample-rate conversion routines
dct.c	forward and inverse Discrete Cosine Transform routines
decision.c	macroblock coding decision routines
encoder.c	main() encoder program
encoder.h	universal encoder constants
encoder.ini	user initilization script file
encoder.pro	collection of prototypes for all encoder modules
flc.h	fixed-length code (FLC) constants used throughput program
genbits.c	bitstream element generation routines for various layers
global.h	global variables
initial.c	initilization routines (picture buffer allocation, etc.)
makefile	example makefile for GNU gcc Unix and MS-DOS compilers
motion.c	motion estimation routines
mpeg1.h	data structures containing buffer pointers, and control parameters
perfdct.c	Zig-Zag, and quantization routines
procpict.c	picture-layer operations
procseq.c	sequence-layer operations
quantize.c	forward and inverse quantization, macroblock coding pattern.
ratectrl.c	Test Model rate control routines
README	informative file
readpict.c	read source picture from file
reconstr.c	macroblock prediction construction routines
stats.c	encoding statistics routines
transfer.c	block memory transfer operations
writebit.c	low-level write bit routines
writepic.c	write reconstructed picture to file

Table B.2 List of decoder files

filename	description
constr.c	construct macroblock predictions
consts.h	constants used throughout decoder program
convert.c	picture format and sample rate conversion routines
decode.c	main() decoder program
decode.h	control externals used in most files
decode.ini	user initilization script indicating picture format, verbose level, etc.

decode.pro	collection of all prototypes for all modules
flc.h	fixed-length code constants used throughout program
getbits.c	low-level bit fetching from file and coded data buffer maintanance
getcode.c	parse AC or DC coefficient from bitstream
global.h	global variables used throughout decoder program
idct.c	inverse DCT routine
initial.c	initlization routines based on sequence header parameters
iquant.c	inverse quantization routine
makefile	example makefile for GNU gcc Unix and MS-DOS compilers
mb.c	macroblock decoding and processing routines
perfidct.c	inverse Zig-Zag scaning and inverse quantization
picture.c	picture processing operations
quant.h	defualt quantization tables
README	informative file
sequence.c	sequence layer processing
slice.c	slice layer processing
transfer.c	block transfers from one picture buffer to another
types.h	data structures containing buffer pointers and control parameters
vlc.h	variable length code tables
writepic.c	write decoded and reconstructed picture to file

B.2 Encoder

B.2.1 acvlc.c

```
/* File:   acvlc.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
 *
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

 /*      $Log:   acvlc.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.2  93/06/15  15:36:44  oosa
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include "consts.h"
#include "mpeg1.h"
#include "bitcount.h"
#include "encoder.pro"

/* initialize dct_coeff_next tables for encoder */
void    Init_AC_VLC(void)
{
    extern BYTE     AC_length[][256];   /* code length */
    extern BYTE     AC_value[][256];    /* code value */
    int             run, level;

    /* Code length for FLC escape events */

    for (run = 0; run < 64; run++)          /* single escape */
        for (level = 0; level < 128; level++)
            AC_length[run][level] = 20;
    for (run = 0; run < 64; run++)          /* double escape */
        for (level = 128; level < 256; level++)
            AC_length[run][level] = 28;
```

```
    /* Code lengths for VLC events */

    AC_length[0][0] = 2;        /* EOB code */
    AC_length[0][1] = 3;
    AC_length[1][1] = 4;
    AC_length[2][1] = AC_length[0][2] = 5;
    AC_length[3][1] = AC_length[4][1] = AC_length[0][3] = 6;
    AC_length[5][1] = AC_length[1][2] = AC_length[6][1] = AC_length[7][1] = 7;
    AC_length[8][1] = AC_length[0][4] = AC_length[9][1] = AC_length[2][2] = 8;

    AC_length[10][1] = AC_length[0][5] = AC_length[1][3] = AC_length[3][2] =  AC_length[11][1]
    = AC_length[12][1] = AC_length[0][6] = AC_length[13][1] = 9;

    AC_length[4][2] = AC_length[14][1] = AC_length[15][1] = AC_length[1][4] =  AC_length[2][3]
    = AC_length[0][7] = AC_length[5][2] = AC_length[16][1] = 11;

    AC_length[17][1] = AC_length[6][2] = AC_length[0][8] = AC_length[3][3] =  AC_length[1][5]
    = AC_length[18][1] = AC_length[19][1] = AC_length[0][9] =  AC_length[20][1]   =
AC_length[21][1] = AC_length[7][2] = AC_length[2][4] =  AC_length[0][10]
            = AC_length[4][3] = AC_length[8][2] = AC_length[0][11] = 13;

    AC_length[22][1] = AC_length[23][1] = AC_length[24][1] = AC_length[25][1] =
AC_length[26][1] = AC_length[9][2] = AC_length[10][2] = AC_length[5][3] =  AC_length[3][4]
            = AC_length[2][5] = AC_length[1][6] = AC_length[1][7] =  AC_length[0][12]
            = AC_length[0][13] = AC_length[0][14] = AC_length[0][15] = 14;

    AC_length[0][16] = AC_length[0][17] = AC_length[0][18] = AC_length[0][19] =
AC_length[0][20]
        = AC_length[0][21] = AC_length[0][22] = AC_length[0][23] = AC_length[0][24]
        = AC_length[0][25] = AC_length[0][26] = AC_length[0][27] = AC_length[0][28]
        = AC_length[0][29] = AC_length[0][30] = AC_length[0][31] = 15;

    AC_length[0][32] = AC_length[0][33] = AC_length[0][34] = AC_length[0][35] =
AC_length[0][36]
        = AC_length[0][37] = AC_length[0][38] = AC_length[0][39] = AC_length[0][40]
        = AC_length[1][8] = AC_length[1][9] = AC_length[1][10] = AC_length[1][11]
        = AC_length[1][12] = AC_length[1][13] = AC_length[1][14] = 16;

    AC_length[1][15] = AC_length[1][16] = AC_length[1][17] = AC_length[1][18] =
AC_length[6][3]
        = AC_length[11][2] = AC_length[12][2] = AC_length[13][2] = AC_length[14][2]
        = AC_length[15][2] = AC_length[16][2] = AC_length[27][1] = AC_length[28][1]
        = AC_length[29][1] = AC_length[30][1] = AC_length[31][1] = 17;

    /* code word values for FLC escape events */
    for (level = 1; level < 256; level++)
    {
        AC_value[EscapeOffset][level] = level;    /* positive */
        AC_value[EscapeOffset + 1][level] =       /* negative */
            ~AC_value[EscapeOffset][level] + 1;
    }

    /* code word values for VLC events */
    AC_value[0][0] = 2;        /* EOB code */
    AC_value[0][1] = 6;
    AC_value[1][1] = 6;
    AC_value[2][1] = 10;
    AC_value[0][2] = 8;
    AC_value[3][1] = 14;
    AC_value[4][1] = 12;
    AC_value[0][3] = 10;
    AC_value[5][1] = 14;
    AC_value[1][2] = 12;
    AC_value[6][1] = 10;
    AC_value[7][1] = 8;
    AC_value[8][1] = 14;
    AC_value[0][4] = 12;
    AC_value[9][1] = 10;
    AC_value[2][2] = 8;
    AC_value[10][1] = 78;
    AC_value[0][5] = 76;
    AC_value[1][3] = 74;
    AC_value[3][2] = 72;
    AC_value[11][1] = 70;
    AC_value[12][1] = 68;
    AC_value[0][6] = 66;
```

```
AC_value[13][1] =64;
AC_value[4][2] = 30;
AC_value[14][1] = 28;
AC_value[15][1] = 26;
AC_value[1][4] = 24;
AC_value[2][3] = 22;
AC_value[0][7] = 20;
AC_value[5][2] = 18;
AC_value[16][1] = 16;
AC_value[17][1] = 62;
AC_value[6][2] = 60;
AC_value[0][8] = 58;
AC_value[3][3] = 56;
AC_value[1][5] = 54;
AC_value[18][1] = 52;
AC_value[19][1] = 50;
AC_value[0][9] = 48;
AC_value[20][1] = 46;
AC_value[21][1] = 44;
AC_value[7][2] = 42;
AC_value[2][4] = 40;
AC_value[0][10] = 38;
AC_value[4][3] = 36;
AC_value[8][2] = 34;
AC_value[0][11] = 32;
AC_value[22][1] = 62;
AC_value[23][1] = 60;
AC_value[24][1] = 58;
AC_value[25][1] = 56;
AC_value[26][1] = 54;
AC_value[9][2] = 34;
AC_value[10][2] = 32;
AC_value[5][3] = 36;
AC_value[3][4] = 38;
AC_value[2][5] = 40;
AC_value[1][6] = 44;
AC_value[1][7] = 42;
AC_value[0][12] = 52;
AC_value[0][13] = 50;
AC_value[0][14] = 48;
AC_value[0][15] = 46;
AC_value[0][16] = 62;
AC_value[0][17] = 60;
AC_value[0][18] = 58;
AC_value[0][19] = 56;
AC_value[0][20] = 54;
AC_value[0][21] = 52;
AC_value[0][22] = 50;
AC_value[0][23] = 48;
AC_value[0][24] = 46;
AC_value[0][25] = 44;
AC_value[0][26] = 42;
AC_value[0][27] = 40;
AC_value[0][28] = 38;
AC_value[0][29] = 36;
AC_value[0][30] = 34;
AC_value[0][31] = 32;
AC_value[0][32] = 48;
AC_value[0][33] = 46;
AC_value[0][34] = 44;
AC_value[0][35] = 42;
AC_value[0][36] = 40;
AC_value[0][37] = 38;
AC_value[0][38] = 36;
AC_value[0][39] = 34;
AC_value[0][40] = 32;
AC_value[1][8] = 62;
AC_value[1][9] = 60;
AC_value[1][10] = 58;
AC_value[1][11] = 56;
AC_value[1][12] = 54;
AC_value[1][13] = 52;
AC_value[1][14] = 50;
AC_value[1][15] = 38;
AC_value[1][16] = 36;
AC_value[1][17] = 34;
```

```
    AC_value[1][18] = 32;
    AC_value[6][3] = 40;
    AC_value[11][2] = 52;
    AC_value[12][2] = 50;
    AC_value[13][2] = 48;
    AC_value[14][2] = 46;
    AC_value[15][2] = 44;
    AC_value[16][2] = 42;
    AC_value[27][1] = 62;
    AC_value[28][1] = 60;
    AC_value[29][1] = 58;
    AC_value[30][1] = 56;
    AC_value[31][1] = 54;
}
```

B.2.2 bitcount.h

```
/* File:   bitcount.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
  By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   bitcount.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:52  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

#define cpbBitsLen              64
#define DCPredSizeLen           9
#define DCPredSizeCLen          9
#define MBABitsLen              34
#define MBTypeILen              2
#define MBTypePLen              8
#define MBTypeBLen              12
#define VlcCodeMagnitudeLen     33

#define kACVlc_Escape20         0x004000
#define kACVlc_Escape28         0x400000
#define kACVlc_Escape           1
```

```
#define kNumEscapeBits          6
#define EscapeOffset            62
#define kACVlcFirst             2       /* dct_coeff_first bit count */
#define kACVlcFirstP            2       /* dct_coeff_first pattern */
```

B.2.3 consts.h

```
/*  File:   consts.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   consts.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

#define kMaxCharBufLen  80

#define FALSE 0
#define TRUE  1

#define BYTE                        unsigned char
#define RND(i,x)                    i = ((x)>0.0 ? floor((x)+0.5) : ceil((x)-0.5))
#define TRUNC(i,x)                  i = ((x)>0.0 ? floor((x)) : ceil((x))

#define kMBWidth                    16      /*  width of Macro Block in pel  */
#define kMBHeight                   16      /*  height of Macro Block in pel */

/* picture dimension routines need to be moved into initial.c */

#define FORWARD         0       /*  for 4:2:2 to 4:2:0 conversion */
#define INVERSE         1       /*  for 4:2:0 to 4:2:2 conversion */
#define FIELD_1         1       /*  sometimes called field 0 */
#define FIELD_2         2       /*  sometimes called field 1 */
```

```
#define kIPicture       1       /*      I picture type          */
#define kPPicture       2       /*      P picture type          */
#define kBPicture       3       /*      B picture type  */
#define kBPicture1      3       /*      1st B picture type      */
#define kBPicture2      4       /*      2nd B picture type      */

#define kMBLen          (kMBWidth * kMBHeight)  /*  # of pels in MB  */
#define kDCTSize        8                       /*  width and height of DCT Block in pel */
#define kDCTLen         (kDCTSize * kDCTSize)   /*  # of pels in DCT Block  */
#define kBWidth         8                       /*  width of Chrom Block in pel */
#define kBHeight        8                       /*  height of Chrom Block in pel */
#define kBLen           (kBWidth * kBHeight)    /*  # of pels in block */

#define kMCType                 0x0002  /*  motion compensated  macro block type */
#define kIntraType              0x0004  /*  intra macro block type */
#define kCodedType              0x0010  /*  coded macro block type */
#define kForwardMV              0x0020  /*  forward mv  */
#define kBackwardMV             0x0040  /*  backward mv  */
#define kModifiedMQ             0x0080  /*  modified quantizer  */

#define kMBWidthDCT             (kMBWidth - kDCTSize)
```

B.2.4 convert.c

```
/* File: convert.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   convert.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:55  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
/*#include <string.h>*/
```

```
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

/* transfer function constants */
static  int hf0[7] ={-29, 0, 88, 138, 88, 0, -29};
#define sf0             256
#define sf0Half         128     /*  sf0 / 2  */
#define Nf0L            -3
#define Nf0H            3
#define Nf0Z            3     /*implies that 138 multiplies Oth line of fl0 */

static int hf1[4] ={1, 7, 7, 1};
#define sf1             16
#define sf1Half         8       /* sf1 / 2 */
#define Nf1L            -2
#define Nf1H            1
#define Nf1Z            2   /* implies that 2st 7 multiplies Oth line of fl1 */

void    convert422to420(
    int             fieldNum,
    BYTE            *chromPicture,
    int             chromWidth,
    int             chromHeight,
    int             direction,
    tParameter      *parameter)
{
    int             i;
    int             j;
    static int      *a, *b;
    void            filter();

    if(!a)
    {
        a = Iarray(parameter->luminHeight);
        b = Iarray(parameter->luminHeight);
    }

    switch(direction)
    {
        case FORWARD:

            for(j = 0; j < chromWidth; j++)
            {
                for(i = 0; i < chromHeight; i++)
                {
                    *(a + i) = *(chromPicture + i * chromWidth + j);
                }

                filter(fieldNum, chromHeight, a, b, FORWARD, parameter);

                for(i = 0; i < chromHeight / 2; i++)
                {
                    *(chromPicture + i * chromWidth +j) = *(b + i);
                }
            }

            break;

        case INVERSE:

            for(j = 0; j < chromWidth; j++)
            {
                for(i = 0; i < chromHeight / 2; i++)
                {
                    *(a + i) = *(chromPicture + i * chromWidth + j);
                }

                filter(fieldNum, chromHeight / 2, a, b, INVERSE,parameter);

                for(i = 0; i < chromHeight; i++)
                {
                    *(chromPicture + i * chromWidth + j) = *(b + i);
                }
            }
```

```
            break;
    }
}

void    filter(
    int     fieldNum,
    int     nn,
    int     *FF,
    int     *GG,
    int     direction,
        tParameter    *parameter)
{
    static int *F, *G;
    int i, k, l, n, M, L, sgn;
    int     tmp;

    if (!F)
    {
        F = Iarray(parameter->luminHeight);
        G = Iarray(parameter->luminHeight);
    }

    switch(direction)
    {
        case FORWARD:

            n = nn / 2;
            M = nn;

            for(i = 0; i < M; i++)
            {
                F[i] = FF[i];
                GG[i] = 0;       /* OK since dimentioned same */
            }

            switch(fieldNum)
            {
                case FIELD_1:

                    for(i = 0; i < n; i++)
                    {
                        k = 2 * i;

                        G[i] = 0;
                        G[n+i] = 0;

                        for(l = Nf0L; l < (Nf0H + 1); l++)
                        {
                            L = k - l;

                            if(L < 0)
                                L += M;
                            else if (L > (M - 1))
                                L -= M;

                            G[i] += hf0[l + Nf0Z] * F[L];
                        }

                        sgn = (G[i] < 0) ? -1 : 1;

                        if (G[i] < 0)
                            GG[i] = 0;
                        else
                        {
                            tmp = (G[i] + sf0Half) / sf0;

                            GG[i] = (tmp > 255) ? 255 : tmp;
                        }
                    }

                    break;  /* end Field 0 */

                case FIELD_2:

                    for(i = 0; i < n; i++)
                    {
```

```
                k = 2 * i;
                G[i] = 0;
                G[n+i] = 0;

                for(l = Nf1L; l < (Nf1H + 1); l++)
                {
                    L = k - l;
                    if(L < 0)
                        L += M;
                    else if(L > (M - 1))
                        L -= M;
                    G[i] += hf1[l + Nf1Z] * F[L];
                }

                sgn = (G[i] < 0) ? -1 : 1;

                GG[i] = (short)(sgn * ((sgn * G[i] + sf1Half) / sf1));
            }  /* end of Field 1 Forward direction */

            break;
    }

    break;

case INVERSE:

    n = nn;
    M = 2 * n;

    for(i = 0; i < M; i++)
    {
        F[i] = FF[i];
        G[i] = 0;
    }

    switch(fieldNum)
    {
        case FIELD_1:

            F[n] = F[n - 1];  /* for 'interpolation' of last point */

            for(i = 0; i < n; i++)
            {
                k = 2 * i;
                GG[k] = (short)F[i];
                G[k + 1] = F[i] + F[i + 1];

                /* normalize and round */

                sgn = (G[k + 1] < 0) ? -1 : 1;
                GG[k + 1] = (short)(sgn * ((sgn * G[k + 1] + 1) / 2));
            }

            break;

        case FIELD_2:

            F[n] = F[0];  /* for 'wrap filter wrap around' */

            for(i = 0; i < n; i++)
            {
                k = 2 * i;
                G[k] = 3 * F[i] + F[i + 1];
                G[k + 1] = F[i] + 3 * F[i + 1];
            }

            /* normalize and round */

            for(i = 0; i < 2 * n; i++)
            {
                sgn = (G[i] < 0) ? -1 : 1;
                GG[i] = (short)(sgn * ((sgn * G[i] + 2) / 4));
            }

            break;
    } /* end INVERSE */
```

```
        break;
    }

    return;
}

void    convert420to422(
    BYTE    *chromPicture,
    int     chromWidth,
    int     chromHeight)
{
    int     i;
    int     j;
    int     *a;
    int     *b;

    a = (int *) malloc(chromHeight * sizeof(int));
    b = (int *) malloc(chromHeight * sizeof(int));

    for(j = 0; j < chromWidth; j++)
    {
        for(i = 0; i < chromHeight / 2; i++)
            *(a + i) = *(chromPicture + i * chromWidth + j);

        filter2(chromHeight / 2, a, b);

        for(i = 0; i < chromHeight; i++)
            *(chromPicture + i * chromWidth + j) = *(b + i);
    }

    free(a);
    free(b);
}

void    filter2(
    int     n,
    int     *F,
    int     *G)
{
    int i, k, M;
    int     tmp;


    M = 2 * n;

    F[n] = F[n - 1];  /* for 'interpolation' of last point */

    for(i = 0; i < n; i++)
    {
        k = 2 * i;
        G[k] = F[i];

        tmp = F[i] + F[i + 1];

        /* normalize and round */
        G[k + 1] = (tmp >= 0) ? ((tmp + 1) / 2) : -((-tmp + 1) / 2);
    }

    return;
}

void convert420to444(
    BYTE    *chrom420,
    BYTE    *chrom444,
    int     chromWidth,
    int     chromHeight)
{
    int     chromWidth2, i, j;
    BYTE    *src, *dst;

    /* bilinear interpolation filter */

    chromWidth2 = 2 * chromWidth;

    /* horizontal direction */
```

```
    for (j = 0; j < chromHeight; j++)
    {
        src = chrom420 + j * chromWidth;
        dst = chrom444 + j * chromWidth2;
        dst[0] = src[0];
        for (i = 1; i < chromWidth; i++)
        {
            dst[2*i-1] = (3 * src[i-1] + src[i] + 2) >> 2;
            dst[2*i]   = (src[i-1] + 3 * src[i] + 2) >> 2;
        }
        dst[chromWidth2-1] = src[chromWidth-1];
    }

    /* vertical direction, (in-place, bottom-up) */

    for (i = 0; i < chromWidth2; i++)
    {
        src = chrom444 + i + (chromHeight-2) * chromWidth2;
        dst = chrom444 + i + (2*(chromHeight-2)+1) * chromWidth2;
        dst[2*chromWidth2] = src[chromWidth2];
        for (j = chromHeight - 2; j >= 0; j--)
        {
            dst[chromWidth2] = (src[0] + 3 * src[chromWidth2] + 2) >> 2;
            dst[0] = (3 * src[0] + src[chromWidth2] + 2) >> 2;
            dst -= 2 * chromWidth2;
            src -= chromWidth2;
        }
    }
}
```

B.2.5 dct.c

```
/* file: dct.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
```


By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).


```
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   dct.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:28:00  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
```

```
 */

#include <math.h>
#include "consts.h"

#define PI      3.141592654

double          Forward_Cosine_Matrix[kDCTSize][kDCTSize];
double          Inverse_Cosine_Matrix[kDCTSize][kDCTSize];


/************************************************************************
*
* Module name: Init_DCT
*
* Initializes the DCT scale and twiddle factor matrix
*
************************************************************************/

void Init_DCT(void)
{
    extern double   Forward_Cosine_Matrix[][kDCTSize];
    extern double   Inverse_Cosine_Matrix[][kDCTSize];

    /* Local variables */

    double  normalization_vector[kDCTSize];
    short freq, time;

    /* Execute */

    /* DC term has different normalization scale than AC terms */
    normalization_vector[0] = sqrt((double)(1.0 / kDCTSize));

    /* Compute normalization scales for fundamentals and harmonics */
    for(freq = 1; freq < kDCTSize; freq++)
        normalization_vector[freq] = sqrt((double)(2.0 / kDCTSize));

    for(freq = 0; freq < kDCTSize; freq++)
    {
        for(time = 0; time < kDCTSize; time++)
        {
            Forward_Cosine_Matrix[freq][time] = normalization_vector[freq]
                * cos((double)((2*time +1) * freq * PI)/(2*kDCTSize));

            Inverse_Cosine_Matrix[freq][time] = normalization_vector[time]
                * cos((double)((2*freq +1) * time * PI)/(2*kDCTSize));
        }
    }
}



/* forward DCT */

void dct8x8(
    float   sample_block[],     /* input block -- spatial domain */
    float   coefficient_block[])      /* output block -- frequency domain*/
{
    extern double   Forward_Cosine_Matrix[][kDCTSize];  /* cosine matrix */

    /* Local variables */

    short   row, col, freq, time, index, index1;
    float   intermediate[kDCTLen];

    /*      Transform rows  */

    for (row = 0, index = 0; row < kDCTSize; row++, index += kDCTSize)
    {
        for(freq = 0; freq < kDCTSize; freq++)
        {
            intermediate[index + freq] = 0;

            for(time = 0; time < kDCTSize; time++)
```

```
            {
                intermediate[index + freq] += sample_block[index + time]
                    * Forward_Cosine_Matrix[freq][time];
            }
        }
    }

    /*  transform colums.  Transpose operation is integrated into
        the indexing of intermediate matrix */

    for (col = 0; col < kDCTSize; col++)
    {
        for(freq = 0, index = col; freq < kDCTSize; freq++, index += kDCTSize)
        {
            coefficient_block[index] = 0;

            for(time = 0, index1 = col; time < kDCTSize; time++, index1 += kDCTSize)
            {
                coefficient_block[index] += intermediate[index1]
                    * Forward_Cosine_Matrix[freq][time];
            }
        }
    }
}

/**********************************************************************
*
* Module name: IDCT
*
* Calculates an inverse discrete cosine transform
*
**********************************************************************/

void idct8x8(
    float   coefficient_block[],
    float   recon_block[])
{
    extern double   Inverse_Cosine_Matrix[][kDCTSize];

    /* Local variables */

    short   row, col, freq, time, index, index1;
    float   intermediate[kDCTLen];

    /* Execute */

    for(row = 0, index = 0; row < kDCTSize; row++, index += kDCTSize)
    {
        for(time = 0; time < kDCTSize; time++)
        {
            intermediate[index + time] = 0;

            for(freq = 0; freq < kDCTSize; freq++)
            {
                intermediate[index + time] += coefficient_block[index + freq]
                    * Inverse_Cosine_Matrix[time][freq];
            }
        }
    }

    for(col = 0; col < kDCTSize; col++)
    {
        for(time = 0, index = col; time < kDCTSize; time++, index += kDCTSize)
        {
            recon_block[index] = 0;

            for(freq = 0, index1 = col; freq < kDCTSize; freq++, index1 += kDCTSize)
            {
                recon_block[index] += intermediate[index1]
                    * Inverse_Cosine_Matrix[time][freq];
            }
        }
    }
}
```

B.2.6 decision.c

```
/* File:   decision.c  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   decision.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:59  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"


/* construct intra macorblock */
int     ConstructI(
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    BYTE    *current,
    BYTE    *currentU,
    BYTE    *currentV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter)
{
    int     index1, index2;
    BYTE    *constY2, *constU2, *constV2;
    BYTE    *currentIndex, *currentIndexU, *currentIndexV;
    int     *diffIndex, *diffIndexU, *diffIndexV;

    /*      reconstruct with intra  */
```

```
    currentIndex = current;
    diffIndex = diff;
    constY2 = constY;

    /*      lumin   */

    for (index1 = 0; index1 < kMBHeight; index1++,
            constY2 += parameter->luminWidthMB)
    {
        for (index2 = 0; index2 < kMBWidth; index2++,
                constY2++, currentIndex++, diffIndex++)
        {
            *constY2 = *currentIndex;
            *diffIndex = *currentIndex;
        }
    }

    /*      chrom   */

    currentIndexU = currentU;
    currentIndexV = currentV;
    diffIndexU =    diffU;
    diffIndexV =    diffV;
    constU2 =       constU;
    constV2 =       constV;


    /* need to change kBHeight to something more clear */
    for (index1 = 0; index1 < kBHeight; index1++,
            constU2 += parameter->chromWidthB,
                constV2 += parameter->chromWidthB)
    {
        for (index2 = 0; index2 < kBWidth; index2++,
                constU2++, constV2++, currentIndexU++, currentIndexV++,
                diffIndexU++, diffIndexV++)
        {
            *constU2 =      *currentIndexU;
            *constV2 =      *currentIndexV;
            *diffIndexU =   *currentIndexU;
            *diffIndexV =   *currentIndexV;
        }
    }

    return(kIntraType);             /*      intra   */
}


/* Make P picture macroblock type decision */

int     Construct(
    BYTE    predictY[],
    BYTE    predictU[],
    BYTE    predictV[],
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    BYTE    current[],
    BYTE    currentU[],
    BYTE    currentV[],
    short   MV[],
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter     *parameter)
{
    int     mbType = 0;

    /*  bit:
     *  1 - MC(1)/No MC(0)
     *  2 - Intra(1)/Inter(0)
     *  5 - ForwardMV
     */

    int     index1, index2;
    int     SEMC, SENoMC;
```

```
BYTE    *constY2, *constU2, *constV2;
BYTE    *currentIndex, *currentIndexU, *currentIndexV;
BYTE    *frameIndex, *frameIndexU, *frameIndexV;
int     *diffIndex, *diffIndexU, *diffIndexV;

/* frame SE (Square Error) */

SEMC = CalSE(current, predictY, MV, parameter);

currentIndex = current;
frameIndex = predictY;

/* frame SE no MC  */

SENoMC = 0;

for (index1 = 0; index1 < kMBHeight; index1++,
        frameIndex += parameter->luminWidthMB)
{
    for (index2 = 0; index2 < kMBWidth; index2++,
            currentIndex++, frameIndex++)
    {
        SENoMC += (*currentIndex - *frameIndex) *
                (*currentIndex - *frameIndex);
    }
}

/* MC/no MC check  */

if (SENoMC > SEMC)
{
    mbType |= kMCType;      /* MC */
    mbType |= kForwardMV;   /* forward MV */

    /* reconstruct MB  */


    FrameConstr(predictY, predictU, predictV, constY, constU,
        constV,MV, current, currentU, currentV, diff, diffU,
        diffV, parameter);
}
else    /* No MC */
{
    constY2 = constY;
    constU2 = constU;
    constV2 = constV;

    frameIndex = predictY;
    currentIndex = current;
    diffIndex = diff;

    /* lumin */

    for (index1 = 0; index1 < kMBHeight; index1++,
            constY2 += parameter->luminWidthMB,
                frameIndex += parameter->luminWidthMB)
    {
        for (index2 = 0; index2 < kMBWidth; index2++,
                constY2++, frameIndex++, currentIndex++,
                diffIndex++)
        {
            *constY2 = *frameIndex;
            *diffIndex = *currentIndex - *constY2;
        }
    }

    /*      chrom   */

    frameIndexU =   predictU;
    frameIndexV =   predictV;
    currentIndexU = currentU;
    currentIndexV = currentV;
    diffIndexU =    diffU;
    diffIndexV =    diffV;

    for (index1 = 0; index1 < kBHeight; index1++,
```

```
                constU2 += parameter->chromWidthB,
                constV2 += parameter->chromWidthB,
                frameIndexU += parameter->chromWidthB,
                frameIndexV += parameter->chromWidthB)
        {
            for (index2 = 0; index2 < kBWidth; index2++,
                    constU2++, constV2++, frameIndexU++, frameIndexV++,
                    currentIndexU++, currentIndexV++,
                    diffIndexU++, diffIndexV++)
            {
                *constU2 = *frameIndexU;
                *constV2 = *frameIndexV;
                *diffIndexU = *currentIndexU - *constU2;
                *diffIndexV = *currentIndexV - *constV2;
            }
        }
    }

    if(IntraCheck(current, currentU, currentV, constY, constU, constV,
            diff, diffU, diffV, parameter))
    {
        mbType = kIntraType;
    }

    return (mbType);
}


/* Make intra/non-intra decision */
int     IntraCheck(
    BYTE    current[],
    BYTE    currentU[],
    BYTE    currentV[],
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter)
{
    BYTE    *currentIndex;
    int     var, varor, sum, *diffIndex;
    short   index1;

    /* intra/inter coding check */

    currentIndex = current;
    diffIndex = diff;
    var = 0;
    varor = 0;
    sum = 0;

    for (index1 = 0; index1 < kMBLen; index1++, currentIndex++, diffIndex++)
    {
        sum += *currentIndex;
        varor += *currentIndex * *currentIndex;
        var += *diffIndex * *diffIndex;
    }

    var /= 256;             /*  var/256  */
    varor /= 256;           /*  varor/256  */
    sum /= 256;             /*  sum/256  */
    varor -= (sum * sum);   /*  varor/256 - sum/256 * sum/256  */

    if ((var > 64) && (var > varor))
    {
        /* reconstruct with intra  */

        return (ConstructI (constY, constU, constV, current,
            currentU, currentV, diff, diffU, diffV, parameter));
    }

    return (0);     /*  non intra */
}
```

```
/* make B picture macroblock decision */
int     ConstructB(
    BYTE    predictY[],
    BYTE    predictU[],
    BYTE    predictV[],
    BYTE    predictY2[],
    BYTE    predictU2[],
    BYTE    predictV2[],
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    BYTE    current[],
    BYTE    currentU[],
    BYTE    currentV[],
    short   MV[],
    short   MVB[],
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter)
{
    int     mbType = kMCType;
    int     SEMC, FSEMC, BSEMC, ISEMC;


    FSEMC = CalSE(current, predictY, MV, parameter);
    BSEMC = CalSE(current, predictY2, MVB, parameter);
    ISEMC = ICalSE(current, predictY, predictY2, MV, MVB, parameter);

    if ((FSEMC <= BSEMC) && (FSEMC <= ISEMC))
    {
        SEMC = 0;       /*      forward         */
    }
    else if ((BSEMC < FSEMC) && (BSEMC <= ISEMC))
    {
        SEMC = 1;       /*      backward        */
    }
    else
    {
        SEMC = 2;       /*      interpolate     */
    }

    switch (SEMC)
    {
        case 0:         /*      forward         */

            mbType |= kForwardMV;

            FrameConstr(predictY, predictU, predictV, constY,
                    constU, constV, MV, current, currentU,
                    currentV, diff, diffU, diffV, parameter);
            break;

        case 1:         /*      backward        */

            mbType |= kBackwardMV;

            FrameConstr(predictY2, predictU2, predictV2, constY,
                    constU, constV, MVB, current, currentU,
                    currentV, diff, diffU, diffV, parameter);
            break;

        case 2:         /*      interpolate     */

            mbType |= kForwardMV;
            mbType |= kBackwardMV;

            FrameIConstr(predictY, predictU, predictV, predictY2,
                    predictU2, predictV2, constY, constU,
                    constV, MV, MVB, current, currentU,
                    currentV, diff, diffU, diffV, parameter);
            break;
    }

    /*      intra/inter coding check        */
```

```
    if(IntraCheck(current, currentU, currentV, constY, constU, constV,
            diff, diffU, diffV, parameter))
    {
        mbType = kIntraType;
    }

    return (mbType);
}


/* Calculate Squared Error between two macroblocks */

int     CalSE(
    BYTE    *current,
    BYTE    *predictY,
    short   MV[],
    tParameter      *parameter)
{
    BYTE    *frameIndex,*currentIndex;
    int     offset, diffByte;
    int     SEFrame;
    short   f0off2, f0off3, f0off4, index1, index2, mvx, mvy;

    /*      calculate SEs   */

    mvx = MV[0] / 2;
    mvy = MV[1] / 2;
    offset = (mvy * parameter->luminWidth) + mvx;
    frameIndex = (BYTE *) (predictY + offset);

    GetLoc(MV, &f0off2, &f0off3, &f0off4, parameter->luminWidth);

    currentIndex = current;

    /*      frame SE (Square Error) */

    SEFrame = 0;

    for (index1 = 0; index1 < kMBHeight; index1++,
            frameIndex += parameter->luminWidthMB)
    {
        for (index2 = 0; index2 < kMBWidth; index2++,
                currentIndex++, frameIndex++)
        {
            /*      add 2 for integer rounding      */

            diffByte = ((int) *frameIndex +
                    (int) *(frameIndex + f0off2) +
                    (int) *(frameIndex + f0off3) +
                    (int) *(frameIndex + f0off4) + 2) / 4;
            diffByte = *currentIndex - diffByte;

            SEFrame += diffByte * diffByte;
        }
    }

    return(SEFrame);
}


/* Calculate Interpolated (Bi-directional) Square Error */
int     ICalSE(
    BYTE    *current,
    BYTE    *predictY,
    BYTE    *predictYB,
    short   MV[],
    short   MVB[],
    tParameter      *parameter)
{
    BYTE    *frameIndex, *frameIndexB, *currentIndex;
    short   index1, index2, mvx, mvy;
    short   f0off2, f0off3, f0off4;
    short   f0off2B, f0off3B, f0off4B;
    int     offset, diffByte, SEFrame;

    /*      calculate SEs   */
```

```
    mvx = MV[0] / 2;
    mvy = MV[1] / 2;
    offset = (mvy * parameter->luminWidth) + mvx;
    frameIndex = (BYTE *) (predictY + offset);

    GetLoc(MV, &f0off2, &f0off3, &f0off4, parameter->luminWidth);

    mvx = MVB[0] / 2;
    mvy = MVB[1] / 2;
    offset = (mvy * parameter->luminWidth) + mvx;
    frameIndexB = (BYTE *) (predictYB + offset);

    GetLoc(MVB, &f0off2B, &f0off3B, &f0off4B, parameter->luminWidth);

    currentIndex = current;

    /*      frame SE (Square Error) */

    SEFrame = 0;

    for (index1 = 0; index1 < kMBHeight; index1++,
            frameIndex += parameter->luminWidthMB,
                frameIndexB += parameter->luminWidthMB)
    {
        for (index2 = 0; index2 < kMBWidth; index2++,
                currentIndex++, frameIndex++, frameIndexB++)
        {
            /*      add 4 for integer rounding      */

            diffByte = ((int) *frameIndex +
                    (int) *(frameIndex + f0off2) +
                    (int) *(frameIndex + f0off3) +
                    (int) *(frameIndex + f0off4) +
                    (int) *frameIndexB +
                    (int) *(frameIndexB + f0off2B) +
                    (int) *(frameIndexB + f0off3B) +
                    (int) *(frameIndexB + f0off4B) + 4) / 8;
            diffByte = *currentIndex - diffByte;

            SEFrame += diffByte * diffByte;
        }
    }

    return (SEFrame);
}
```

B.2.7 encoder.c

```
/*   File:   encoder.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
```

```
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   encoder.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.3      93/08/20  21:15:53  hanamura
 * MPEG1 encoder minor revision
 *
 * Revision 1.2  93/06/15  15:29:13  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "consts.h"
#include "mpeg1.h"
#include "flc.h"
#include "encoder.pro"

int main(
    int     argc,
    char    *argv[])
{
    tParameter      parameter;
    tData           data;
    tRateControl    rateControl;
    int ret;

fprintf(stderr,"Starting program\n");
    if (argc > 1)   /*      get parameter data      */
        Initial(&parameter, &data, &rateControl, argv[1]);
    else
        Initial(&parameter, &data, &rateControl, "encoder.ini");

fprintf(stderr,"Initilized program\n");
    /* initialize user parameters */
    ret = InitParameter(&parameter);

    /* initialize picture buffers */
    ret = InitData(&data, &parameter);

        ret = Print_Parameters(&data, &parameter, &rateControl);

    ProcSequence(&parameter, &data, &rateControl);
/* add error exit condition to the above  */

    return 0;
}
```

B.2.8 encoder.h

```
/*  File:   encoder.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

 * These software programs are available to the user without any license fee or
```

```
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   encoder.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 */

#ifdef GLOBAL
#define EXTERN
#else
#define EXTERN extern
#endif

/*
EXTERN int HorizontalSize, VerticalSize, MbWidth, MbHeight;
EXTERN FILE *TraceFile;
EXTERN int TraceLevel;

float aspectRatioTable[16] = {0.0, 1.0000, 0.6735, 0.7031, 0.7615, 0.8055,
        0.8437, 0.8935, 0.9157, 0.9815, 1.0255, 1.0695, 1.0950, 1.1575,
        1.2015, 0};
*/

double pictureRateTable[9] = {0.0, 23.976, 24.0, 25.0, 29.97, 30.0, 50.0, 59.94, 60.0};

/*
static char  *aspectTab[16] = {"forbidden", "1.0000 (VGA etc.)", "0.6735",
                                "0.7031 (16:9, 625line)", "0.7615", "0.8055",
                                "0.8437 (16:9, 525line)", "0.8935",
                                "0.9157 (CCIR601, 625line)", "0.9815", "1.0255", "1.0695",
                                "1.0950 (CCIR601, 525line)", "1.1575", "1.2015", "reserved"};

static char *rateTab[9] = {"forbidden", "23.976", "24", "25", "29.97",
                                "30", "50", "59.94", "60"};
*/
```

B.2.9 encoder.ini

```
r3_             /* prefix of source pictures file name */
rr3             /* reconstructed pictures file name prefix */
3               /* source picture file format: 0=SIF, 1=TGA, 2=PPM 3=YUV */
1               /* recon picture file format:  0=SIF, 1=TGA, 2=PPM 3=YUV */
status5         /* file name where status data is to be written */
test5.mpg        /* file name for coded bitstream */
0               /* first picture in sequence */
3               /* last picture in sequence */
352             /* source horizontal size (luminance samples) */
240             /* source vertical size  (luminance samples) */
12              /* pel aspect ratio code (see ISO/IEC 11172-2  2.4.3.2 ) */
5               /* frame rate code (see ISO/IEC 11172-2  2.4.3.2) */
1200.0          /* bit rate in multiples of kilobits/sec */
```

```
7.0             /* maximum vbv delay in picture period units */
327680          /* maxiumum vbv buffer size in units of bits */
1               /* constrained parameter flag */
15              /* Group of picture size / distance between I pictures */
3               /* distance between P pictures (M factor) */
7               /* maximum f code (motion vector / search distance) */
1               /* half pel enable ? */
1               /* use motion vector files ?  */
```

B.2.10 encoder.pro

```
void    Init_AC_VLC(
    void);
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

void    Write_Sequence_Header(
    tParameter      *parameter,
    tRateControl    *rateControl);

short   Write_GOP_Header(
    tParameter      *parameter,
    tRateControl    *rateControl);

short   Write_Picture_Header(
    short           picture_coding_type,
    tRateControl    *rateControl,
    short           forward_f_code,
    short           backward_f_code,
    int             temporal_reference);

short   Write_Slice_Header(
    short           picture_coding_type,
    tRateControl    *rateControl);

int     Write_MB(
    short           picture_coding_type,
    int             macrblock_type,
    short           coded_block_pattern,
    tRateControl    *rateControl,
    tParameter      *parameter,
    int             MB_row,
    int             MB_col,
    short           forwardMV[],
    short           backwardMV[],
    int             y_coef[],
    int             cb_coef[],
    int             cr_coef[],
    int             quantizer_scale,
    short           fRange,
    short           forward_f_code,
    short           bRange,
    short           backward_f_code,
    int             *new_macroblock_type);

void    Write_MV(
    int             *diff_vector,
    short           numDiffVectors,
    short           f_code,
    short           range,
    int             *MB_bit_count,
    tParameter      *parameter,
    short           picture_coding_type);

void    Write_Intra_Coefficients(
    int             *y_coef,
    int             *cr_coef,
    int             *cb_coef,
    int             *dct_dc_past_y,
    int             *dct_dc_past_cb,
```

```
    int             *dct_dc_past_cr,
    int             *MB_bit_count);

void    Write_Non_Intra_Coefficients(
    int             *y_coef,
    int             *cb_coef,
    int             *cr_coef,
    int             *MB_bit_count,
    short           coded_block_pattern);

int     Write_AC_VLC(
    int             diff,
    short           *run,
    int             *MB_bit_count,
    short           first);

short   CheckMbType(
    int             prevMbType,
    int             mbType,
    short           codedBPattern);

int     CalSE(
    BYTE            *current,
    BYTE            *predictY,
    short           MV[],
    tParameter      *parameter);

int     Construct(
    BYTE            predictY[],
    BYTE            predictU[],
    BYTE            predictV[],
    BYTE            *constY,
    BYTE            *constU,
    BYTE            *constV,
    BYTE            current[],
    BYTE            currentU[],
    BYTE            currentV[],
    short           MV[],
    int             *diff,
    int             *diffU,
    int             *diffV,
    tParameter      *parameter);

int     IntraCheck(
    BYTE            current[],
    BYTE            currentU[],
    BYTE            currentV[],
    BYTE            *constY,
    BYTE            *constU,
    BYTE            *constV,
    int             *diff,
    int             *diffU,
    int             *diffV,
    tParameter      *parameter);

int     ConstructB(
    BYTE    predictY[],
    BYTE    predictU[],
    BYTE    predictV[],
    BYTE    predictY2[],
    BYTE    predictU2[],
    BYTE    predictV2[],
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    BYTE    current[],
    BYTE    currentU[],
    BYTE    currentV[],
    short   MV[],
    short   MVB[],
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter);

int     ConstructI(
```

```
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    BYTE    *current,
    BYTE    *currentU,
    BYTE    *currentV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter);

int     ICalSE(
    BYTE    *current,
    BYTE    *predictY,
    BYTE    *predictYB,
    short   MV[],
    short   MVB[],
    tParameter      *parameter);

void    PreFrame(
    short           pictureType,
    tParameter      *parameter,
    tRateControl    *rateControl,
    short           forward_f_code,
    short           backward_f_code);

void    PostFrame(
    short           pictureType,
    tRateControl    *rateControl,
    tParameter      *parameter);

void    DoPreMBRC(
    short           pictureType,
    tRateControl    *rateControl,
    BYTE            *currentMB,
    int             *MQuant);

void    PerformDct(
    int             mbType,
    BYTE            currentMB[],
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             mbIndex);

void    dct8x8(
    float   frame[],                        /* picture store */
    float   trframe[]);                     /* dct transformed picture */

void    writeBits(
    unsigned int    bits,
    short           numBits,
    short           startCode);

void    Write_Intra_Coefficients(
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             *DCPredictionY,
    int             *DCPredictionU,
    int             *DCPredictionV,
    int             *MB_bit_count);

/*
short   GetMotionVector(
    BYTE    *reconst,
    BYTE    *original,
    BYTE    *current,
    short   bufMV[][2],
    short   searchDistance,
    int     halfPel,
    short   fileMV,
    char    *fName,
    tParameter      *parameter);
*/
```

```
unsigned int    HalfPel(
    BYTE                    *original,
    BYTE                    *current,
    int                     *hor,
    int                     *ver,
    int                     bufRow,
    int                     bufCol,
    unsigned int            minF,
    short                   offset,
    tParameter              *parameter);


short   GetMotionVector(
    BYTE    *reconst,
    BYTE    *original,
    BYTE    *current,
    short   bufMV[][2],
    int     searchDistance,
    int     halfPel,
    int     fileMV,
    char    *fName,
    tParameter *parameter);



/*
unsigned int    HalfPel(
    BYTE            *original,
    BYTE            *current,
    short           mv[],
    int             bufRow,
    int             bufCol,
    unsigned int    minF,
    short           offset,
    short           mvF[],
        tParameter      *parameter);

*/


void    ProcessMB(
    tData           *data,
    tParameter      *parameter,
    tRateControl    *rateControl,
    BYTE            *bufY,
    BYTE            *bufU,
    BYTE            *bufV,
    BYTE            *bufYR,
    BYTE            *bufUR,
    BYTE            *bufVR,
    short           pictureType,
    int             pictureNum,
    short           forwardMV[][2],
    short           backwardMV[][2],
    int             mbType[],
    short           codedBPattern[],
    int             MQuant[]);

int     QuantizeI(
    short           *codedBPattern,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             MQuant);

int     Quantize(
    short           *codedBPattern,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             MQuant);

int     calcVbvDelay(
    short           pictureType,
    tRateControl    *rateControl,
    tParameter      *parameter);
```

```
void    checkUpdateVbv(
    short           pictureType,
    int             pictureNum,
    tRateControl    *rateControl,
    tParameter      *parameter);

void    GetLoc(
    short   MV[],
    short   *f0off2,
    short   *f0off3,
    short   *f0off4,
    short   width);

void    FrameConstr(
    BYTE    *predictY,
    BYTE    *predictU,
    BYTE    *predictV,
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    short   MV[],
    BYTE    *current,
    BYTE    *currentU,
    BYTE    *currentV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter);

void    FrameIConstr(
    BYTE    *predictY,
    BYTE    *predictU,
    BYTE    *predictV,
    BYTE    *predictYB,
    BYTE    *predictUB,
    BYTE    *predictVB,
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    short   MV[],
    short   MVB[],
    BYTE    *current,
    BYTE    *currentU,
    BYTE    *currentV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter);


short   ReadPicture(
    int             aPictureNum,
    BYTE            *luminPicture,
    BYTE            *chromUPicture,
    BYTE            *chromVPicture,
    tParameter      *aParameter);


short   WritePicture(
    int             aPictureNum,
    short           aPictureType,
    BYTE            *luminPictureR,
    BYTE            *chromUPictureR,
    BYTE            *chromVPictureR,
    BYTE            *luminPicture,
    BYTE            *chromUPicture,
    BYTE            *chromVPicture,
    tParameter      *aParameter);


void    convert422to420(
    int             fieldNum,
    BYTE            *chromPicture,
    int             chromWidth,
    int             chromHeight,
```

```
    int             direction,
    tParameter      *parameter);


void    filter(
    int     fieldNum,
    int     nn,
    int     *FF,
    int     *GG,
    int     direction,
    tParameter  *parameter);


void    PerformIDct(
    int             mbType,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV);


void    idct8x8(
    float   trframe[],      /*      dct transformed picture */
    float   itrframe[]);    /*      picture store   */


void    Initial(
    tParameter              *aParameter,
    tData                   *aData,
    tRateControl            *aRateControl,
    char                    *parameterFName);


BYTE    *Barray(
    int     size);


short *Sarray(
    int     size);

int *Iarray(
    int     size);


void    IQuantizeI(
    short           codedBPattern,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             MQuant);

void    IQuantize(
    short           codedBPattern,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             MQuant);


void    printDiff(
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV);


void    statistic(
    int             pictureNum,
    short           pictureType,
    tParameter      *parameter,
    BYTE            *original,
    BYTE            *originalU,
    BYTE            *originalV,
    BYTE            *reconst,
    BYTE            *reconstU,
    BYTE            *reconstV,
    int             numLuminPixels,
    int             numChromPixels);
```

```
void    snr(
    BYTE    *ycod,
    BYTE    *ysrc,
    double  *ysnr,
    BYTE    *ucod,
    BYTE    *usrc,
    double  *usnr,
    BYTE    *vcod,
    BYTE    *vsrc,
    double  *vsnr,
    int     numLuminPixels,
    int     numChromPixels);


void    CopyToBuf(
    int     bufIndex,
    int     bufIndexC,
    BYTE    bufY[],
    BYTE    bufU[],
    BYTE    bufV[],
    BYTE    *currentMB,
    BYTE    *currentMBU,
    BYTE    *currentMBV,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    tParameter      *parameter);


void    CopyFromBuf(
    short           pictureType,
    int             mbType,
    short           codedBPattern,
    int             mbRow,
    int             mbCol,
    BYTE            *bufYR,
    BYTE            *bufUR,
    BYTE            *bufVR,
    int             currentMBDiff[],
    int             currentMBDiffU[],
    int             currentMBDiffV[],
    int             mbIndex,
    tParameter      *parameter);




int FullSearch(
    int                     *hor,
    int                     *ver,
    short                   searchDistance,
    int                     halfPel,
    BYTE                    *reconst,
        BYTE                    *current_picture,
    BYTE                    *original,
    BYTE                    *current,
    int                     bufRow,
    int                     bufCol,
    tParameter              *parameter);

int MAE (BYTE           *currentIndex1,
     BYTE           *predictIndex,
     tParameter     *parameter);

void    ProcSequence(
    tParameter      *parameter,
    tData           *data,
    tRateControl    *rateControl);

void    ProcPicture(
    tData           *data,
    tParameter      *parameter,
    tRateControl    *rateControl,
```

```
    BYTE            *bufY,          /* source picture buffer */
    BYTE            *bufU,
    BYTE            *bufV,
    BYTE            *bufYR,         /* reconstruction picture buffer */
    BYTE            *bufUR,
    BYTE            *bufVR,
    int             pictureType,
    int             pictureIndex,   /* P=0,B1=1,B2=2,... */
    short           forwardMV[][2], /* motion vector field */
    short           backwardMV[][2],
    int             mbType[],
    short           codedBPattern[],
    int             MQuant[]);

int InitParameter(tParameter *parameter);

int InitData(tData *aData, tParameter *aParameter);

void Init_DCT(void);


short   Read_SIF(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
        tParameter      *parameter);

short   Read_TGA(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter);

short   Read_PPM(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter);

short   Read_YUV(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter);

int     Write_SIF(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter);

int     Write_TGA(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter);

int     Write_PPM(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter);

void    yuvtorgb(
    int             y_recon,
    int             cb_recon,
    int             cr_recon,
    int             *pr,
    int             *pg,
```

```
    int             *pb);

int     Write_YUV(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter);

void    convert420to422(
    BYTE    *chromPicture,
    int             chromWidth,
    int             chromHeight);

void    filter2(
    int     n,
    int     *F,
    int     *G);

void convert420to444(
    BYTE    *chrom420,
    BYTE    *chrom444,
    int             chromWidth,
    int             chromHeight);
```

B.2.11 flc.h

```
/* File: flc.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   flc.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.3      93/08/20  21:15:53  hanamura
 * MPEG1 encoder minor revision
 *
 * Revision 1.2  93/06/15  14:59:40  oosa
 * *** empty log message ***
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
```

```
 */

/*      sequence flc    */
#define bSequenceHeaderCode             32      /*      bslbf   */
#define bHorizontalSize                 12      /*      uimsbf  */
#define bVerticalSize                   12      /*      uimsbf  */
#define bPelAspectRatio                 4       /*      uimsbf  */
#define bPictureRate                    4       /*      uimsbf  */
#define bBitRate                        18      /*      uimsbf  */
#define bMarkerBit                      1       /*      "1      */
#define bVbvBufferSize                  10      /*      uimsbf  */
#define bConstrainedParameterFlag       1
#define bLoadIntraQuantizerMatrix       1
#define bIntraQuantizer                 8       /*      uimsbf  */
#define bLoadNonIntraQuantizerMatrix    1
#define bNonIntraQuantizer              8       /*      uimsbf  */
#define bSequenceEndCode                32      /*      bslbf   */

#define cSequenceHeaderCode             0x000001B3      /*      bslbf   */
#define cHorizontalSize                 kLuminWidth     /*      uimsbf  */
#define cVerticalSize                   kLuminHeight    /*      uimsbf  */
#define cPelAspectRatio                 12              /*      uimsbf  */
#define cPictureRate                    4               /*      uimsbf  */
#define cBitRate                        400             /*      uimsbf  */
#define cMarkerBit                      1               /*      "1"     */
#define cVbvBufferSize                  1               /*      NEED    */
#define cConstrainedParameterFlag       1               /*      NEED    */
#define cLoadIntraQuantizerMatrix       0               /* default (0), else (1)  NEED */
#define cLoadNonIntraQuantizerMatrix    0               /* default (0), else (1)  NEED */
#define cSequenceEndCode                0x000001B7      /* bslbf   */

/*      group of picture flc    */

#define bGOPHeaderCode                  32      /*      bslbf   */
#define bTimeCode                       25      /*      NEED    */
#define bClosedGOP                      1       /*      NEED    */
#define bBrokenLink                     1       /*      NEED    */

#define cGOPHeaderCode                  0x000001B8   /*      bslbf   */
#define cTimeCode                       25      /*      NEED    */
#define cClosedGOP                      0       /*      NEED    */
#define cBrokenLink                     0       /*      NEED    */


/*      picture flc     */

#define bPictureStartCode               32      /*      bslbf   */
#define bTemporalReference              10      /*      uimsbf  */
#define bPictureCodingType              3       /*      uimsbf  */
#define bVbvDelay                       16      /*      uimsbf  */
#define bFullPelForwardVector           1
#define bForwardFCode                   3       /*      uimsbf  */
#define bFullPelBackwardVector          1
#define bBackwardFCode                  3       /*      uimsbf  */
#define bExtraBitPicture                1       /*      uimsbf  */

#define cPictureStartCode               0x00000100   /*      bslbf   */
#define cTemporalReference              1       /*      NEED    */
#define cVbvDelay                       1       /*      uimsbf  */
#define cFullPelForwardVector           0
#define cForwardFCode                   3       /*      uimsbf  */
#define cFullPelBackwardVector          0
#define cBackwardFCode                  3       /*      uimsbf  */
#define cExtraBitPicture                0       /*      uimsbf  */

/*      slice flc       */

#define bSliceStartCode                 32      /*      bslbf   */
#define bQuantizerScale                 5       /*      uimsbf  */
#define bExtraBitSlice                  1       /*      uimsbf  */

#define cSliceStartCode                 0x00000101  /*      bslbf   */
#define cQuantizerScale                 5       /*      NEED - uimsbf   */
#define cExtraBitSlice                  0       /*      uimsbf  */

/*      MB flc  */
```

```
#define bFrameMotionType                2       /*      uimsbf  */
#define bFieldMotionType                2       /*      uimsbf  */
#define bEndOfMacroBlock                1       /*      "1"    */
```

B.2.12 genbits.c

```
/*  file: genbits.c */

 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   genbits.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.2  93/06/15  15:36:31  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "flc.h"
#include "bitcount.h"
#include "global.h"
#include "encoder.pro"


/* write sequence header and update its cost to ratecontrol */
void    Write_Sequence_Header(
    tParameter *parameter,
    tRateControl *rateControl)

{
    extern BYTE tQuantIntra[], tQuant[];

    int     index,
        index1;
```

```
    rateControl->headerBits += bSequenceHeaderCode;
    writeBits(cSequenceHeaderCode, bSequenceHeaderCode, 1);

    rateControl->headerBits += bHorizontalSize;
    writeBits(parameter->horizontal_size, bHorizontalSize, 0);

    rateControl->headerBits += bVerticalSize;
    writeBits(parameter->vertical_size, bVerticalSize, 0);

    rateControl->headerBits += bPelAspectRatio;
    writeBits(parameter->pel_aspect_ratio, bPelAspectRatio, 0);

    rateControl->headerBits += bPictureRate;
    writeBits(parameter->picture_rate, bPictureRate, 0);

    rateControl->headerBits += bBitRate;

    /* convert internal type to bitstream representation */
    index1 = parameter->bit_rate;
    index = index1 / cBitRate;
    if (index1 % cBitRate)
        index++;
    writeBits(index, bBitRate, 0);

    rateControl->headerBits += bMarkerBit;
    writeBits(cMarkerBit, bMarkerBit, 0);

    /* convert internal buffer size representation to bitstream type */
    rateControl->headerBits += bVbvBufferSize;
    index = parameter->vbv_buffer_size / 16384 ; /* 16384 = 16 * 1024 */
    writeBits(index, bVbvBufferSize, 0);

    rateControl->headerBits += bConstrainedParameterFlag;
    writeBits(parameter->constrained_parameters_flag, bConstrainedParameterFlag, 0);

    rateControl->headerBits += bLoadIntraQuantizerMatrix;
    writeBits(parameter->load_intra_quantizer_matrix, bLoadIntraQuantizerMatrix, 0);

    if(parameter->load_intra_quantizer_matrix) {
        /*      loadIntraQuantizerMatrix = TRUE */

        rateControl->headerBits += 64 * bIntraQuantizer;
        for (index = 0; index < 64; index++)
            writeBits(tQuantIntra[index], bIntraQuantizer, 0);
    }

    rateControl->headerBits += bLoadNonIntraQuantizerMatrix;
    writeBits(parameter->load_non_intra_quantizer_matrix, bLoadNonIntraQuantizerMatrix, 0);

    if(parameter->load_non_intra_quantizer_matrix) {
        /*      loadNonIntraQuantizerMatrix = TRUE      */

        rateControl->headerBits += 64 * bNonIntraQuantizer;
        for (index = 0; index < 64; index++)
            writeBits(tQuant[index], bNonIntraQuantizer, 0);
    }
}


/* write GOP header and update its cost to ratecontrol */
short   Write_GOP_Header(
        tParameter      *parameter,
        tRateControl    *rateControl)
{
    int             pic, sec, min, hrs;
        int  fps;
    short           numBits;

    numBits = rateControl->Si;

    rateControl->Si += bGOPHeaderCode;
    writeBits(cGOPHeaderCode, bGOPHeaderCode, 1);

    /* we need special handling for 29.97 and 23.XXX cases */
```

```
fprintf(stderr,"Now calculating GOP time code\n");
    /* this line may need to be changed to the look-up value */
    fps = (int) parameter->float_picture_rate;

    pic = parameter->pictureNum % fps;
    sec = (parameter->pictureNum / fps) % 60;
    min = (parameter->pictureNum / (60 * fps)) % 60;
    hrs = (parameter->pictureNum / (3600 * fps)) % 60;

    parameter->time_code = (long) (parameter->drop_frame_flag << 24) |
          (hrs << 19) | (min << 13) | (1 << 12) | (sec << 6) | (pic);

    /* probably better to have several writebit calls... that way
    we don't have to worry about Big vs. Little Endian */

    rateControl->Si += bTimeCode;
    writeBits(parameter->time_code, bTimeCode, 0);

printf(stderr,"GOP time code = %ld\n", parameter->time_code);

    rateControl->Si += bClosedGOP;
    writeBits(parameter->closed_GOP, bClosedGOP, 0);
    rateControl->Si += bBrokenLink;

    /* need to change broken link flag */
    writeBits(parameter->broken_link, bBrokenLink, 0);

    numBits = rateControl->Si - numBits;

    return (numBits);
}

/* write picture header and update its cost to rate control */
short   Write_Picture_Header(
    short           picture_coding_type,
    tRateControl    *rateControl,
    short           forward_f_code,
    short           backward_f_code,
    int             temporal_reference)   /* display picture number */
{
    short   numBits;

    switch(picture_coding_type)
    {
        case kBPicture:

            numBits = rateControl->Sb;

            rateControl->Sb += bPictureStartCode;
            writeBits(cPictureStartCode, bPictureStartCode, 1);
            rateControl->Sb += bTemporalReference;

            writeBits(temporal_reference%1024, bTemporalReference, 0);

            rateControl->Sb += bPictureCodingType;
            writeBits(kBPicture, bPictureCodingType, 0);
            rateControl->Sb += bVbvDelay;
            if(rateControl->vbv_delay>=0){
                writeBits(rateControl->vbv_delay, bVbvDelay, 0);
                }
            else{
                /* VBV underflow */
                writeBits(0, bVbvDelay, 0);
                }
            rateControl->Sb += bFullPelForwardVector;
            writeBits(cFullPelForwardVector, bFullPelForwardVector, 0);
            rateControl->Sb += bForwardFCode;
            writeBits(forward_f_code, bForwardFCode, 0);
            rateControl->Sb += bFullPelBackwardVector;
            writeBits(cFullPelBackwardVector, bFullPelBackwardVector, 0);
            rateControl->Sb += bBackwardFCode;
            writeBits(backward_f_code, bBackwardFCode, 0);
            rateControl->Sb += bExtraBitPicture;
            writeBits(cExtraBitPicture, bExtraBitPicture, 0);

            numBits = rateControl->Sb - numBits;
```

```
            break;

        case kPPicture:

            numBits = rateControl->Sp;

            rateControl->Sp += bPictureStartCode;
            writeBits(cPictureStartCode, bPictureStartCode, 1);
            rateControl->Sp += bTemporalReference;
            writeBits(temporal_reference%1024, bTemporalReference, 0);
            rateControl->Sp += bPictureCodingType;
            writeBits(kPPicture, bPictureCodingType, 0);
            rateControl->Sp += bVbvDelay;
            if(rateControl->vbv_delay>=0){
                writeBits(rateControl->vbv_delay, bVbvDelay, 0);
            }
            else{
                /* VBV underflow */
                writeBits(0, bVbvDelay, 0);
            }
            rateControl->Sp += bFullPelForwardVector;
            writeBits(cFullPelForwardVector, bFullPelForwardVector, 0);
            rateControl->Sp += bForwardFCode;
            writeBits(forward_f_code, bForwardFCode, 0);
            rateControl->Sp += bExtraBitPicture;
            writeBits(cExtraBitPicture, bExtraBitPicture, 0);

            numBits = rateControl->Sp - numBits;

            break;

        case kIPicture:

            numBits = rateControl->Si;

            rateControl->Si += bPictureStartCode;
            writeBits(cPictureStartCode, bPictureStartCode, 1);
            rateControl->Si += bTemporalReference;
            writeBits(temporal_reference%1024, bTemporalReference, 0);
            rateControl->Si += bPictureCodingType;
            writeBits(kIPicture, bPictureCodingType, 0);
            rateControl->Si += bVbvDelay;
            if(rateControl->vbv_delay>=0){
                writeBits(rateControl->vbv_delay, bVbvDelay, 0);
            }
            else{
                /* VBV underflow */
                writeBits(0, bVbvDelay, 0);
            }
            rateControl->Si += bExtraBitPicture;
            writeBits(cExtraBitPicture, bExtraBitPicture, 0);

            numBits = rateControl->Si - numBits;

            break;
    }

    return (numBits);
}


short   Write_Slice_Header(
    short           picture_coding_type,
    tRateControl    *rateControl)
{
    short   numBits;

    rateControl->macroblock_address_increment = 1;

    switch (picture_coding_type)
    {
        case kBPicture:

            numBits = rateControl->Sb;
```

```
            rateControl->Sb += bSliceStartCode;
            rateControl->Sb += bQuantizerScale;
            rateControl->Sb += bExtraBitSlice;

            numBits = rateControl->Sb - numBits;

            break;

        case kPPicture:

            numBits = rateControl->Sp;

            rateControl->Sp += bSliceStartCode;
            rateControl->Sp += bQuantizerScale;
            rateControl->Sp += bExtraBitSlice;

            numBits = rateControl->Sp - numBits;

            break;

        case kIPicture:

            numBits = rateControl->Si;

            rateControl->Si += bSliceStartCode;
            rateControl->Si += bQuantizerScale;
            rateControl->Si += bExtraBitSlice;

            numBits = rateControl->Si - numBits;

            break;
    }

    return (numBits);
}


int     Write_MB(
    short           picture_coding_type,
    int             macroblock_type,
    short           coded_block_pattern,
    tRateControl    *rateControl,
    tParameter      *parameter,
    int             MB_row,         /* current position with picture */
    int             MB_col,         /* current position within picture */
    short           forwardMV[],
    short           backwardMV[],
    int             y_coef[],
    int             cb_coef[],
    int             cr_coef[],
    int             quantizer_scale,
    short           fRange,
    short           forward_f_code,
    short           bRange,
    short           backward_f_code,
    int             *new_macroblock_type)
{
    static int      previous_macroblock_type;
    static int      previous_quantizer_scale;
    static short    PMVFor[2];
    static short    PMVBack[2];
    int             diff_vector[4], diff_vectorB[4];
    int             MB_bit_count = 0;
    static int      dct_dc_y_past, dct_dc_cb_past, dct_dc_cr_past;
    short           modified_quantizer_scale = 0;
    short           same;

    if (! MB_row && MB_col == 0)      /*      new frame       */
        previous_macroblock_type = 0;

    if (! MB_col)
    {
        /*      new slice       */

        dct_dc_y_past = 128;
        dct_dc_cb_past = 128;
```

```
        dct_dc_cr_past = 128;
    }

    /*      determine number of bits required to code current MB    */

    switch (picture_coding_type)
    {
        case kIPicture:

            MB_bit_count += 1;    /*      MBA - 1 bit                             */
            writeBits(1, 1, 0);

            /*      code MB type - I frame  */

            if (macroblock_type & kModifiedMQ)
            {
                MB_bit_count += MB_typeI_length[1];
                writeBits(MB_typeI_value[1], MB_typeI_length[1], 0);
                MB_bit_count += 5;
                writeBits(quantizer_scale, 5, 0);
                *new_macroblock_type = MB_typeI_switch[1];
            }
            else
            {
                MB_bit_count += MB_typeI_length[0];
                writeBits(MB_typeI_value[0], MB_typeI_length[0], 0);
                *new_macroblock_type = MB_typeI_switch[0];
            }

            /*      code blocks     */

            Write_Intra_Coefficients(y_coef, cb_coef, cr_coef,
                    &dct_dc_y_past, &dct_dc_cb_past, &dct_dc_cr_past,
                    &MB_bit_count);

            rateControl->dji += MB_bit_count - rateControl->Tij;
            rateControl->Si += MB_bit_count;
            return(MB_bit_count);

        case kPPicture:

            if ((coded_block_pattern) || (! MB_col) ||
                (MB_col == parameter->numMBsPerSlice - 1)
                    || (macroblock_type & kMCType))
            {
                if ((! MB_col) || (macroblock_type & kIntraType))
                {
                    PMVFor[0] = 0;
                    PMVFor[1] = 0;
                }

                while (rateControl->macroblock_address_increment > 33)
                {
                    MB_bit_count += MBA_length[0];
                    writeBits(MBA_value[0], MBA_length[0], 0);
                    rateControl->macroblock_address_increment -= 33;
                }

                MB_bit_count += MBA_length[rateControl->macroblock_address_increment];
                writeBits(MBA_value[rateControl->macroblock_address_increment],
                        MBA_length[rateControl->macroblock_address_increment], 0);

                rateControl->macroblock_address_increment = 1;

                /*      code mb type - P frame  */

                if (macroblock_type & kIntraType)
                {
                    if (macroblock_type & kModifiedMQ)
                    {
                        /*      mb type - intra w/ mod MQ       */

                        MB_bit_count += MB_typeP_length[6];
                        writeBits(MB_typeP_value[6], MB_typeP_length[6], 0);
                        modified_quantizer_scale = 1;
                        *new_macroblock_type = MB_typeP_switch[6];
```

```
                    }
                    else
                    {
                        MB_bit_count += MB_typeP_length[3];   /*      mb type - intra
*/
                        writeBits(MB_typeP_value[3], MB_typeP_length[3], 0);
                        *new_macroblock_type = MB_typeP_switch[3];
                    }
                }
                else
                {
                    if (macroblock_type & kMCType)
                    {
                        if (coded_block_pattern)
                        {
                            if (macroblock_type & kModifiedMQ)
                            {
                                /*      mb type - MC,coded,mod MQ       */

                                MB_bit_count += MB_typeP_length[4];
                                writeBits(MB_typeP_value[4], MB_typeP_length[4], 0);
                                modified_quantizer_scale = 1;
                                *new_macroblock_type = MB_typeP_switch[4];
                            }
                            else    /*      no mod MQ       */
                            {
                                /*      mb type - MC,coded      */

                                MB_bit_count += MB_typeP_length[0];
                                writeBits(MB_typeP_value[0], MB_typeP_length[0], 0);
                                *new_macroblock_type = MB_typeP_switch[0];
                            }
                        }
                        else    /*      not coded       */
                        {
                            /*      mb type - MC                    */

                            MB_bit_count += MB_typeP_length[2];
                            writeBits(MB_typeP_value[2], MB_typeP_length[2], 0);
                            *new_macroblock_type = MB_typeP_switch[2];
                            if (MB_col)
                                quantizer_scale = previous_quantizer_scale;
                        }
                    }
                    else    /*      no MC   */
                    {
                        if (coded_block_pattern)
                        {
                            if (macroblock_type & kModifiedMQ)
                            {
                                /*      mb type - coded,mod MQ  */

                                MB_bit_count += MB_typeP_length[5];
                                writeBits(MB_typeP_value[5], MB_typeP_length[5], 0);
                                modified_quantizer_scale = 1;
                                *new_macroblock_type = MB_typeP_switch[5];
                            }
                            else
                            {
                                /*      mb type - coded                 */

                                MB_bit_count += MB_typeP_length[1];
                                writeBits(MB_typeP_value[1], MB_typeP_length[1], 0);
                                *new_macroblock_type = MB_typeP_switch[1];
                            }

                            PMVFor[0] = 0;
                            PMVFor[1] = 0;
                        }
                        else
                        {
                            /*      mb type - no MC, not coded */
                            /*
                                first or last MB in slice, not skipped
                                -> MC with MV=0, not coded
                            */
```

```
/*  3/21/94: the following two lines were added by Mr. Hirofumi Nisikawa
         (Mitsubishi Electric Corp) and Kinya Oosa (Nippon Steel).  */

                             forwardMV[0] = 0;
                             forwardMV[1] = 0;

        macroblock_type |= kMCType;
                             MB_bit_count += MB_typeP_length[2];
                             writeBits(MB_typeP_value[2], MB_typeP_length[2], 0);
                             *new_macroblock_type = MB_typeP_switch[2];
                             if (MB_col)
                                 quantizer_scale = previous_quantizer_scale;
                         }
                     }
                 }
            }
            else    /*      skipped */
            {
                PMVFor[0] = 0;
                PMVFor[1] = 0;

                rateControl->macroblock_address_increment++;
                rateControl->djp += MB_bit_count - rateControl->Tpj;
                rateControl->Sp += MB_bit_count;
                return(MB_bit_count);
            }

            previous_quantizer_scale = quantizer_scale;

            if ((macroblock_type & kIntraType) &&
                    (! (previous_macroblock_type & kIntraType)))
            {
                dct_dc_y_past = 128;
                dct_dc_cb_past = 128;
                dct_dc_cr_past = 128;
            }

            break;

        case kBPicture:

            /*      check if mvs are the same       */

            same = FALSE;

            if ((macroblock_type & kMCType) && (previous_macroblock_type & kMCType))
            {
                /*      check mvs are the same  */

                if ((macroblock_type & kForwardMV) && (macroblock_type & kBackwardMV))
                {
                    if ((PMVFor[0] == forwardMV[0]) &&
                            (PMVFor[1] == forwardMV[1]) &&
                            (PMVBack[0] == backwardMV[0]) &&
                            (PMVBack[1] == backwardMV[1]))
                        same = TRUE;
                }
                else if (macroblock_type & kForwardMV)
                {
                    if ((PMVFor[0] == forwardMV[0]) &&
                            (PMVFor[1] == forwardMV[1]))
                        same = TRUE;
                }
                else    /*      macroblock_type & kBackwardMV    */
                {
                    if ((PMVBack[0] == backwardMV[0]) &&
                            (PMVBack[1] == backwardMV[1]))
                        same = TRUE;
                }
            }

            if (same)       /*      check if macroblock_type the same        */
                same = CheckMbType(previous_macroblock_type, macroblock_type,
coded_block_pattern);
```

```
if ((! same) || (! MB_col) || (MB_col == (parameter->numMBsPerSlice - 1)) ||
        coded_block_pattern)
{
    if ((! MB_col) || (macroblock_type & kIntraType))
    {
        PMVFor[0] = 0;
        PMVFor[1] = 0;
        PMVBack[0] = 0;
        PMVBack[1] = 0;
    }

    while (rateControl->macroblock_address_increment > 33)
    {
        MB_bit_count += MBA_length[0];
        writeBits(MBA_value[0], MBA_length[0], 0);
        rateControl->macroblock_address_increment -= 33;
    }

    MB_bit_count += MBA_length[rateControl->macroblock_address_increment];
    writeBits(MBA_value[rateControl->macroblock_address_increment],
            MBA_length[rateControl->macroblock_address_increment], 0);

    rateControl->macroblock_address_increment = 1;

    /*      code mb type - B frame  */

    if (macroblock_type & kIntraType)
    {
        if (macroblock_type & kModifiedMQ)
        {
            /*      mb type - intra w/ mod MQ       */

            MB_bit_count += MB_typeB_length[10];
            writeBits(MB_typeB_value[10], MB_typeB_length[10], 0);
            modified_quantizer_scale = 1;
            *new_macroblock_type = MB_typeB_switch[10];
        }
        else    /*      no mod MQ       */
        {
            /*      mb type - intra                         */

            MB_bit_count += MB_typeB_length[6];
            writeBits(MB_typeB_value[6], MB_typeB_length[6], 0);
            *new_macroblock_type = MB_typeB_switch[6];
        }
    }
    else    /*      non intra       */
    {
        if (macroblock_type & kForwardMV)
        {
            if (macroblock_type & kBackwardMV)
            {
                /*      interpolated    */

                if (coded_block_pattern)
                {
                    if (macroblock_type & kModifiedMQ)
                    {
                        /*      interp,coded,mod MQ     */

                        MB_bit_count += MB_typeB_length[7];
                        writeBits(MB_typeB_value[7], MB_typeB_length[7], 0);
                        modified_quantizer_scale = 1;
                        *new_macroblock_type = MB_typeB_switch[7];
                    }
                    else    /*      no MQ   */
                    {
                        /*      interp,coded    */

                        MB_bit_count += MB_typeB_length[1];
                        writeBits(MB_typeB_value[1], MB_typeB_length[1], 0);
                        *new_macroblock_type = MB_typeB_switch[1];
                    }
                }
                else    /*      not coded       */
```

```
                    {
                        /*      interp          */

                        MB_bit_count += MB_typeB_length[0];
                        writeBits(MB_typeB_value[0], MB_typeB_length[0], 0);
                        *new_macroblock_type = MB_typeB_switch[0];
                    }
                }
                else    /*      forward only    */
                {
                    if (coded_block_pattern)
                    {
                        if (macroblock_type & kModifiedMQ)
                        {
                            /*      for,coded,mod MQ        */

                            MB_bit_count += MB_typeB_length[8];
                            writeBits(MB_typeB_value[8], MB_typeB_length[8], 0);
                            modified_quantizer_scale = 1;
                            *new_macroblock_type = MB_typeB_switch[8];
                        }
                        else    /*      no MQ   */
                        {
                            /*      for,coded       */

                            MB_bit_count += MB_typeB_length[5];
                            writeBits(MB_typeB_value[5], MB_typeB_length[5], 0);
                            *new_macroblock_type = MB_typeB_switch[5];
                        }
                    }
                    else    /*      not coded       */
                    {
                        /*      for             */

                        MB_bit_count += MB_typeB_length[4];
                        writeBits(MB_typeB_value[4], MB_typeB_length[4], 0);
                        *new_macroblock_type = MB_typeB_switch[4];
                    }
                }
            }
            else    /*      (macroblock_type & kBackwardMV)  */
            {
                if (coded_block_pattern)
                {
                    if (macroblock_type & kModifiedMQ)
                    {
                        /*      back,coded,mod MQ       */

                        MB_bit_count += MB_typeB_length[9];
                        writeBits(MB_typeB_value[9], MB_typeB_length[9], 0);
                        modified_quantizer_scale = 1;
                        *new_macroblock_type = MB_typeB_switch[9];
                    }
                    else    /*      no MQ   */
                    {
                        /*      back,coded      */

                        MB_bit_count += MB_typeB_length[3];
                        writeBits(MB_typeB_value[3], MB_typeB_length[3], 0);
                        *new_macroblock_type = MB_typeB_switch[3];
                    }
                }
                else    /*      not coded       */
                {
                    /*      back            */

                    MB_bit_count += MB_typeB_length[2];
                    writeBits(MB_typeB_value[2], MB_typeB_length[2], 0);
                    *new_macroblock_type = MB_typeB_switch[2];
                }
            }
        }
    }
    else    /*      skipped */
    {
        /*      current MVs == prev MVs */
```

```
                rateControl->macroblock_address_increment++;
                rateControl->djb += MB_bit_count - rateControl->Tbj;
                rateControl->Sb += MB_bit_count;
                return(MB_bit_count);
            }

            if ((macroblock_type & kIntraType) &&
                    (! (previous_macroblock_type & kIntraType)))
            {
                dct_dc_y_past = 128;
                dct_dc_cb_past = 128;
                dct_dc_cr_past = 128;
            }

            break;
    }

    previous_macroblock_type = *new_macroblock_type;

    if (modified_quantizer_scale)
    {
        MB_bit_count += 5;            /*      new MQ  */
        writeBits(quantizer_scale, 5, 0);
    }

    if ((! (macroblock_type & kIntraType)) && (macroblock_type & kMCType))
    {
        switch (picture_coding_type)
        {
            case kBPicture:

                /*      find diff_vector        */

                if (macroblock_type & kForwardMV)
                {
                    diff_vector[0] = forwardMV[0] - PMVFor[0];
                    diff_vector[1] = forwardMV[1] - PMVFor[1];

                    PMVFor[0] = forwardMV[0];
                    PMVFor[1] = forwardMV[1];

                    /*      code foward mv  */

                    Write_MV(diff_vector, 2, forward_f_code,
                            fRange, &MB_bit_count, parameter, picture_coding_type);
                }

                if (macroblock_type & kBackwardMV)
                {
                    diff_vectorB[0] = backwardMV[0] - PMVBack[0];
                    diff_vectorB[1] = backwardMV[1] - PMVBack[1];

                    PMVBack[0] = backwardMV[0];
                    PMVBack[1] = backwardMV[1];

                    /*      code backward mv        */

                    Write_MV(diff_vectorB, 2, backward_f_code,
                            bRange, &MB_bit_count, parameter, picture_coding_type);
                }

                break;

            case kPPicture:

                /*      find diff_vector        */

                diff_vector[0] = forwardMV[0] - PMVFor[0];
                diff_vector[1] = forwardMV[1] - PMVFor[1];

                PMVFor[0] = forwardMV[0];
                PMVFor[1] = forwardMV[1];

                /*      code foward mv  */
```

```
                Write_MV(diff_vector, 2, forward_f_code,
                        fRange, &MB_bit_count, parameter, picture_coding_type);

                break;
        }
    }

    if ((! (macroblock_type & kIntraType)) && (macroblock_type & kCodedType))
    {
        /*      code CBP        */

        MB_bit_count += CBP_length[coded_block_pattern];
        writeBits(CBP_value[coded_block_pattern], CBP_length[coded_block_pattern], 0);
    }

    /*      code coeffs     */

    if (macroblock_type & kIntraType)
        Write_Intra_Coefficients(y_coef, cb_coef, cr_coef,
                &dct_dc_y_past, &dct_dc_cb_past, &dct_dc_cr_past, &MB_bit_count);
    else if (coded_block_pattern)
        Write_Non_Intra_Coefficients(y_coef, cb_coef, cr_coef,
                &MB_bit_count, coded_block_pattern);

    switch (picture_coding_type)
    {
        case kPPicture:

            rateControl->djp += MB_bit_count - rateControl->Tpj;
            rateControl->Sp += MB_bit_count;

            break;

        case kBPicture:

            rateControl->djb += MB_bit_count - rateControl->Tbj;
            rateControl->Sb += MB_bit_count;

            break;
    }

    return(MB_bit_count);
}

/* generate and write macroblock motion vector data to bitstream */

void    Write_MV(
    int             *diff_vector,
    short           numDiffVectors,
    short           f_code,
    short           range,
    int             *MB_bit_count,
    tParameter      *parameter,
    short           picture_coding_type)
{
    int             scale_factor, residual, vlc_code_magnitude;
    short   index;

    scale_factor = 1 << (f_code - 1);

    for (index = 0; index < numDiffVectors; index++)
    {
        if (diff_vector[index] < -range)
            diff_vector[index] += 2 * range;
        else if (diff_vector[index] > (range - 1))
            diff_vector[index] -= 2 * range;

        if (diff_vector[index] == 0)
        {
            residual = 0;
            vlc_code_magnitude = 0;
        }
        else
        {
            residual = (abs(diff_vector[index]) - 1) % scale_factor;
            vlc_code_magnitude =
```

```
                    (abs(diff_vector[index]) - residual) / scale_factor;

            if (scale_factor != 1)
                vlc_code_magnitude++;
        }

        /*      encode vlc_code_magnitude and sign of diff_vector[index]        */

        *MB_bit_count += MV_length[vlc_code_magnitude];

        if (diff_vector[index] < 0)
            writeBits(MV_value[vlc_code_magnitude] + 1,
                    MV_length[vlc_code_magnitude], 0);
        else
            writeBits(MV_value[vlc_code_magnitude],
                    MV_length[vlc_code_magnitude], 0);

        if (f_code != 1 && vlc_code_magnitude != 0)
        {
            *MB_bit_count += f_code - 1;  /*      encode residual */
            writeBits(residual, f_code - 1, 0);
        }
    }
}


void    Write_Intra_Coefficients(
    int             *y_coef,
    int             *cb_coef,
    int             *cr_coef,
    int             *dct_dc_y_past,
    int             *dct_dc_cb_past,
    int             *dct_dc_cr_past,
    int             *MB_bit_count)
{
    unsigned int    tempInt;
    short                   size, line, pix, row, col, offset, currentIndex, run;
    int                             diff_DC;

    offset = kMBWidth - kDCTSize;

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex = line * kMBWidth + pix;

            /*      code DC coefficient     */

            diff_DC = abs(y_coef[currentIndex] - *dct_dc_y_past);

            /*      determine size  */

            size = 0;

            if (diff_DC != 0)
            {
                while (diff_DC >= (1 << size))
                    size++;

                *MB_bit_count += DC_Y_length[size];            /*      VLC code
*/
                writeBits(DC_Y_value[size], DC_Y_length[size], 0);

                diff_DC = y_coef[currentIndex] - *dct_dc_y_past;

                if (diff_DC < 0)
                {
                    tempInt = -diff_DC;
                    tempInt = ~tempInt;
                    tempInt = (tempInt << (32 - size)) >> (32 - size);
                }
                else
                    tempInt = diff_DC;
```

```
                *MB_bit_count += size;                                      /*
additional bits */
                writeBits(tempInt, size, 0);

                *dct_dc_y_past = y_coef[currentIndex];
            }
            else
            {
                *MB_bit_count += DC_Y_length[size];           /*      VLC code
*/
                writeBits(DC_Y_value[size], DC_Y_length[size], 0);
            }

            /*      code AC coeffs  */

            col = 1;
            currentIndex++;
            run = 0;

            for (row = 0; row < kDCTSize; row++,
                    currentIndex += offset)
            {
                for ( ; col < kDCTSize; col++, currentIndex++)
                {
                    if (y_coef[currentIndex])
                    {
                        Write_AC_VLC(y_coef[currentIndex],
                                &run, MB_bit_count, 0);
                    }
                    else    /*      zero    */
                    {
                        run++;
                    }
                }

                col = 0;
            }

            *MB_bit_count += AC_length[0][0];  /*  EOB code  */
            writeBits(AC_value[0][0], AC_length[0][0], 0);
        }
    }

    /*      chrom - U       */

    /*      code DC coefficient     */

    diff_DC = abs(cb_coef[0] - *dct_dc_cb_past);

    /*      determine size  */

    size = 0;

    while (diff_DC >= (1 << size))
        size++;

    *MB_bit_count += DC_C_length[size];   /*  VLC code  */
    writeBits(DC_C_value[size], DC_C_length[size], 0);

    diff_DC = cb_coef[0] - *dct_dc_cb_past;

    if (diff_DC < 0)
    {
        tempInt = -diff_DC;
        tempInt = ~tempInt;
        tempInt = (tempInt << (32 - size)) >> (32 - size);
    }
    else
        tempInt = diff_DC;

    *MB_bit_count += size;                                      /*      additional bits */
    writeBits(tempInt, size, 0);

    *dct_dc_cb_past = cb_coef[0];

    /*      code AC coeffs  */
```

```
col = 1;
currentIndex = 1;
run = 0;

for (row = 0; row < kDCTSize; row++)
{
    for ( ; col < kDCTSize; col++, currentIndex++)
    {
        if (cb_coef[currentIndex])
        {
            Write_AC_VLC(cb_coef[currentIndex],
                    &run, MB_bit_count, 0);
        }
        else    /*      zero    */
        {
            run++;
        }
    }

    col = 0;
}

*MB_bit_count += AC_length[0][0];         /*      EOB code        */
writeBits(AC_value[0][0], AC_length[0][0], 0);

/*      chrom - V       */

/*      code DC coefficient     */

diff_DC = abs(cr_coef[0] - *dct_dc_cr_past);

/*      determine size  */

size = 0;

while (diff_DC >= (1 << size))
    size++;

*MB_bit_count += DC_C_length[size];  /*  VLC code  */
writeBits(DC_C_value[size], DC_C_length[size], 0);

diff_DC = cr_coef[0] - *dct_dc_cr_past;

if (diff_DC < 0)
{
    tempInt = -diff_DC;
    tempInt = ~tempInt;
    tempInt = (tempInt << (32 - size)) >> (32 - size);
}
else
    tempInt = diff_DC;

*MB_bit_count += size;          /* additional bits */
writeBits(tempInt, size, 0);

*dct_dc_cr_past = cr_coef[0];

/*      code AC coeffs  */

col = 1;
currentIndex = 1;
run = 0;

for (row = 0; row < kDCTSize; row++)
{
    for ( ; col < kDCTSize; col++, currentIndex++)
    {
        if (cr_coef[currentIndex])
        {
            Write_AC_VLC(cr_coef[currentIndex],
                    &run, MB_bit_count, 0);
        }
        else    /*      zero    */
        {
            run++;
```

```
            }
        }

        col = 0;
    }

    *MB_bit_count += AC_length[0][0];         /*      EOB code        */
    writeBits(AC_value[0][0], AC_length[0][0], 0);
}


void    Write_Non_Intra_Coefficients(
    int     *y_coef,
    int     *cb_coef,
    int     *cr_coef,
    int     *MB_bit_count,
    short   coded_block_pattern)
{
    short   line, pix, row, col, offset, currentIndex, run, code, first;

    offset = kMBWidth - kDCTSize;

    code = 0x20;

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            if (! (coded_block_pattern & code))
            {
                code >>= 1;
                continue;
            }

            code >>= 1;

            currentIndex = line * kMBWidth + pix;

            run = 0;
            first = 1;

            for (row = 0; row < kDCTSize; row++,
                    currentIndex += offset)
            {
                for (col = 0; col < kDCTSize; col++, currentIndex++)
                {
                    if (y_coef[currentIndex])
                    {
                        Write_AC_VLC(y_coef[currentIndex],
                                &run, MB_bit_count, first);

                        first = 0;
                    }
                    else    /*      zero    */
                    {
                        run++;
                    }
                }
            }

            *MB_bit_count += AC_length[0][0];         /*      EOB code        */
            writeBits(AC_value[0][0], AC_length[0][0], 0);
        }
    }

    /*      chrom - U       */

    if (coded_block_pattern & 0x02)
    {
        run = 0;
        currentIndex = 0;
        first = 1;

        for (row = 0; row < kDCTSize; row++)
        {
            for (col = 0; col < kDCTSize; col++, currentIndex++)
```

```
            {
                if (cb_coef[currentIndex])
                {
                    Write_AC_VLC(cb_coef[currentIndex],
                            &run, MB_bit_count, first);
                    first = 0;
                }
                else    /*      zero    */
                {
                    run++;
                }
            }
        }

        *MB_bit_count += AC_length[0][0];         /*      EOB code        */
        writeBits(AC_value[0][0], AC_length[0][0], 0);
    }

    /*      chrominance - Cr       */

    if (coded_block_pattern & 0x01)
    {
        run = 0;
        currentIndex = 0;
        first = 1;

        for (row = 0; row < kDCTSize; row++)
        {
            for (col = 0; col < kDCTSize; col++, currentIndex++)
            {
                if (cr_coef[currentIndex])
                {
                    Write_AC_VLC(cr_coef[currentIndex],
                            &run, MB_bit_count, first);
                    first = 0;
                }
                else    /*      zero    */
                {
                    run++;
                }
            }
        }

        *MB_bit_count += AC_length[0][0];   /*      EOB code        */
        writeBits(AC_value[0][0], AC_length[0][0], 0);
    }
}


int     Write_AC_VLC(
    int     diff,
    short   *run,
    int     *MB_bit_count,
    short   first)
{
    unsigned int    mag, tempInt;
    short           signBit;
    int             numBits;

    numBits = *MB_bit_count;
    mag = abs(diff);
    signBit = (diff < 0) ? 1 : 0;

    if (first && (mag == 1) && (*run == 0))
    {
        writeBits(kACVlcFirstP + signBit, kACVlcFirst, 0);
        *MB_bit_count += kACVlcFirst;
    }
    else
    {
        *MB_bit_count += AC_length[*run][mag];

        if (AC_length[*run][mag] == 20)
        {
            tempInt = kACVlc_Escape20;
            tempInt |= (*run << 8);
```

```
            tempInt |= AC_value[EscapeOffset + signBit][mag];
            writeBits(tempInt, 20, 0);
        }
        else if (AC_length[*run][mag] == 28)
        {
            tempInt = kACVlc_Escape28;
            tempInt |= (*run << 16);

            if (signBit)
                tempInt |= 0x8000;

            tempInt |= AC_value[EscapeOffset + signBit][mag];
            writeBits(tempInt, 28, 0);
        }
        else
        {
            writeBits(AC_value[*run][mag] + signBit,
                AC_length[*run][mag], 0);
        }
    }

    *run = 0;

    return(*MB_bit_count - numBits);
}


short   CheckMbType(
    int     previous_macroblock_type,
    int     macroblock_type,
    short   coded_block_pattern)
{
    int     new_macroblock_type;

    /*      both previous_macroblock_type and macroblock_type are not intra        */

    if (macroblock_type & kForwardMV)
    {
        if (macroblock_type & kBackwardMV)
        {
            /*      interpolated    */

            if (coded_block_pattern)
            {
                if (macroblock_type & kModifiedMQ)
                {
                    /*      interp,coded,mod MQ     */

                    new_macroblock_type = MB_typeB_switch[7];
                }
                else    /*      no MQ   */
                {
                    /*      interp,coded    */

                    new_macroblock_type = MB_typeB_switch[1];
                }
            }
            else    /*      not coded       */
            {
                /*      interp          */

                new_macroblock_type = MB_typeB_switch[0];
            }
        }
        else    /*      forward only    */
        {
            if (coded_block_pattern)
            {
                if (macroblock_type & kModifiedMQ)
                {
                    /*      for,coded,mod MQ        */

                    new_macroblock_type = MB_typeB_switch[8];
                }
                else    /*      no MQ   */
                {
```

```
                    /*      for,coded       */

                    new_macroblock_type = MB_typeB_switch[5];
                }
            }
            else    /*      not coded       */
            {
                /*      for             */

                new_macroblock_type = MB_typeB_switch[4];
            }
        }
    }
    else if (macroblock_type & kBackwardMV)
    {
        if (coded_block_pattern)
        {
            if (macroblock_type & kModifiedMQ)
            {
                /*      back,coded,mod MQ       */

                new_macroblock_type = MB_typeB_switch[9];
            }
            else    /*      no MQ   */
            {
                /*      back,coded      */

                new_macroblock_type = MB_typeB_switch[3];
            }
        }
        else    /*      not coded       */
        {
            /*      back            */

            new_macroblock_type = MB_typeB_switch[2];
        }
    }
    else    /*      no motion compensation   */
    {
        /*      could be skipped        */

        new_macroblock_type = MB_typeB_switch[11];
    }

    if (new_macroblock_type == previous_macroblock_type)
        return (1);

    return (0);
}
```

B.2.13 global.h

```
/* File: global.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
```

```
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   global.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
*/


/* _length represents the length of the variable length code (VLC)
   _value represents the right justified numerical value of the VLC.
   For example, 00011, would have a length of 5 and a value of 3.

*/

/* Macroblock address increment VLC table (B.1) */
BYTE    MBA_length[MBABitsLen] = {
    11, 1, 3, 3, 4, 4, 5, 5, 7, 7, 8, 8, 8, 8, 8, 8,
       10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,
       11,11,11};

BYTE    MBA_value[MBABitsLen] = {
     8, 1, 3, 2, 3, 2, 3, 2, 7, 6,11,10, 9, 8, 7, 6,
       23,22,21,20,19,18,35,34,33,32,31,30,29,28,27,
       26,25,24};

/* Coded block pattern VLC table (B.3) */
BYTE    CBP_length[cpbBitsLen] = {0,
    5, 5, 6, 4, 7, 7, 8, 4, 7, 7,
    8, 5, 8, 8, 8, 4, 7, 7, 8, 5,
    8, 8, 8, 6, 8, 8, 9, 5, 8, 8,
    9, 4, 7, 7, 8, 6, 8, 8, 9, 5,
    8, 8, 8, 5, 8, 8, 9, 5, 8, 8,
    8, 5, 8, 8, 9, 5, 8, 8, 9, 3,
    5, 5, 6};

BYTE    CBP_value[cpbBitsLen] = {0,
    11,  9, 13, 13, 23, 19, 31, 12, 22, 18,
    30, 19, 27, 23, 19, 11, 21, 17, 29, 17,
    25, 21, 17, 15, 15, 13,  3, 15, 11,  7,
     7, 10, 20, 16, 28, 14, 14, 12,  2, 16,
    24, 20, 16, 14, 10,  6,  6, 18, 26, 22,
    18, 13,  9,  5,  5, 12,  8,  4,  4,  7,
    10,  8, 12};

/* DC predition error size for luminance (B.5a) */
BYTE    DC_Y_length[DCPredSizeLen]   = {3, 2, 2, 3, 3, 4, 5,  6,  7};
BYTE    DC_Y_value[DCPredSizeLen]  = {4, 0, 1, 5, 6,14,30, 62,126};

/* DC prediction error size for chrominance (B.5b) */
BYTE    DC_C_length[DCPredSizeCLen]  = {2, 2, 2, 3, 4, 5, 6,  7,  8};
BYTE    DC_C_value[DCPredSizeCLen] = {0, 1, 2, 6,14,30,62,126,254};

/* dct_coef_next:  AC run-level events.  Entries are initialized in
   a separate module */

BYTE    AC_length[64][256];
BYTE    AC_value[64][256];

/* motion vector VLCs:  table B.4 */
```

```
BYTE    MV_length[VlcCodeMagnitudeLen] = {
     1, 3, 4, 5, 7, 8, 8, 8, 10,10,10,11,11,11,11,11,11,
     3, 4, 5, 7, 8, 8, 8, 10,10,10,11,11,11,11,11,11};

BYTE    MV_value[VlcCodeMagnitudeLen] = {
     1, 2, 2, 2, 6,10, 8, 6, 22,20,18,34,32,30,28,26,24,
     3, 3, 3, 7,11, 9, 7, 23,21,19,35,33,31,29,27,25};

/* Intra picture macroblock type tables (B.2a)  */
BYTE    MB_typeI_length[MBTypeILen] = {1, 2};
BYTE    MB_typeI_value[MBTypeILen] = {1, 1};


BYTE    MB_typeI_switch[MBTypeILen] = {kIntraType, kIntraType | kModifiedMQ};

/* Predictive picture macroblock type table (B.2b) */
BYTE    MB_typeP_length[MBTypePLen] = {1, 2, 3, 5, 5, 5, 6, 7};
BYTE    MB_typeP_value[MBTypePLen] = {1, 1, 1, 3, 2, 1, 1, 1};

short   MB_typeP_switch[MBTypePLen] = {
    kForwardMV | kMCType | kCodedType,
    kCodedType,
        kForwardMV | kMCType,
        kIntraType,
    kForwardMV | kMCType | kCodedType | kModifiedMQ,
    kCodedType | kModifiedMQ,
        kIntraType | kModifiedMQ,
        0};

/* Bi-directional picture macroblock type table (B.2c) */
BYTE    MB_typeB_length[MBTypeBLen] = {2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 6, 7};
BYTE    MB_typeB_value[MBTypeBLen] = {2, 3, 2, 3, 2, 3, 3, 2, 3, 2, 1, 1};

short   MB_typeB_switch[MBTypeBLen] = {
    kForwardMV | kBackwardMV | kMCType,
    kForwardMV | kBackwardMV | kMCType | kCodedType,
    kBackwardMV | kMCType,
    kBackwardMV | kMCType | kCodedType,
    kForwardMV | kMCType,
    kForwardMV | kMCType | kCodedType,
    kIntraType,
    kForwardMV | kBackwardMV | kMCType | kCodedType | kModifiedMQ,
    kForwardMV | kMCType | kCodedType | kModifiedMQ,
    kBackwardMV | kMCType | kCodedType | kModifiedMQ,
    kIntraType | kModifiedMQ,
    0};
```

B.2.14 initial.c

```
/* File: initial.c */
/* This file reads in the user parameter script file (encoder.ini) */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
```

```
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   initial.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.1.1.1  93/03/29  11:27:55  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.h"
#include "encoder.pro"

void    Initial(
    tParameter      *parameter,
    tData           *Data,
    tRateControl    *rateControl,
    char            *parameterFName)

{
    extern FILE     *gBitStream;

    FILE    *fptr;
    int     i, j;
    char    lineBuf[kMaxCharBufLen];
    double  vbvBufferStart;


    /* get user input data */

    if ((fptr = fopen (parameterFName, "r")) == 0)
    {
        fprintf (stderr, "input parameter file '%s' not found\n", parameterFName);
        exit(-1);
    }

    /*  file data (string) */
    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", parameter->inputSequence);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", parameter->outputSequence);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->sourceFileFormat);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->reconFileFormat);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", parameter->statusFName);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", parameter->bitStreamFName);

    fgets (lineBuf, kMaxCharBufLen, fptr);
```

```
    sscanf (lineBuf, "%d", &parameter->firstPicture);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->lastPicture);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->horizontal_size);

fprintf(stderr,"hor = %d  ", parameter->horizontal_size);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->vertical_size);

fprintf(stderr,"ver = %d\n", parameter->vertical_size);

    /* read in pel aspect ratio */
    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->pel_aspect_ratio);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->picture_rate);
    if(parameter->picture_rate < 1 || parameter->picture_rate>8)
    {
      fprintf(stderr, "Invalid picture rate code (%d)\n", parameter->picture_rate);
      exit(-1);
    }
    parameter->float_picture_rate=pictureRateTable[parameter->picture_rate];

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%lf", &parameter->bit_rate);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%lf", &parameter->maxPictureDelay);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->maxVbvBufferSize);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->constrained_parameters_flag);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->N);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->M);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->searchDistance);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->halfPel);

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &parameter->fileMV);  /* 0 = FALSE */

    fclose (fptr);


    /* change bitRate in kbit/s to that in bit/s */
    parameter->bit_rate *= 1000.;

    /* convert picture rate from one of the 8 legal values indicated
    in the MPEG-1 specification (clause XXXX) to a floating point
    value */
    /* The amount of initial insertion to VBV buffer */
    /*  = half of bit per frame                      */

    vbvBufferStart = 0.5 * parameter->bit_rate / parameter->float_picture_rate;

    /* calculate vbv_buffer_size */ /* 16384 = 16*1024 */
    parameter->vbv_buffer_size =
        parameter->maxPictureDelay * parameter->bit_rate
        / parameter->float_picture_rate
        + vbvBufferStart;

    parameter->vbv_buffer_size =
```

```
        (parameter->vbv_buffer_size / 16384 + 1) * 16384;


    if(parameter->vbv_buffer_size > parameter->maxVbvBufferSize)
            parameter->vbv_buffer_size = parameter->maxVbvBufferSize;

    parameter->forward_f_code = Iarray(parameter->M);
    parameter->backward_f_code = Iarray(parameter->M);
    parameter->fRange = Iarray(parameter->M);
    parameter->bRange = Iarray(parameter->M);

    /*      range are multiplied by 2 for half pel  */

    /*      P Picture       */

    parameter->fRange[0] = parameter->searchDistance *
            parameter->M * 2;
    parameter->forward_f_code[0] = 1;
    while (parameter->fRange[0] > (8 << parameter->forward_f_code[0]))
        parameter->forward_f_code[0]++;
    parameter->fRange[0] = 8 << parameter->forward_f_code[0];
    parameter->backward_f_code[0] = 0;
    parameter->bRange[0] = 0;

    /*      B Pictures      */

    for (i = parameter->M - 1, j = 1; i > 0; i--, j++)
    {
        parameter->fRange[j] = parameter->searchDistance * j * 2;
        parameter->forward_f_code[j] = 1;
        while (parameter->fRange[j] > (8 << parameter->forward_f_code[j]))
            parameter->forward_f_code[j]++;
        parameter->fRange[j] = 8 << parameter->forward_f_code[j];
        parameter->bRange[j] = parameter->searchDistance * i * 2;
        parameter->backward_f_code[j] = 1;
        while (parameter->bRange[j] > (8 << parameter->backward_f_code[j]))
            parameter->backward_f_code[j]++;
        parameter->bRange[j] = 8 << parameter->backward_f_code[j];
    }

    Init_AC_VLC();

    if ((gBitStream = fopen (parameter->bitStreamFName, "wb")) == NULL)
    {
        printf ("error opening bitstream file '%s' ... Terminating ...\n",
                parameter->bitStreamFName);
        exit(-1);
    }

    rateControl->headerBits = 0;
    rateControl->totalGOPBits = 0;

    rateControl->Xi = 160.0 * parameter->bit_rate / 115.0;
    rateControl->Xp = 60.0 * parameter->bit_rate / 115.0;
    rateControl->Xb = 42.0 * parameter->bit_rate / 115.0;
    rateControl->R = 0;

    /*      Note: check number of frames in first GOP       */

    rateControl->G = (parameter->bit_rate /
            parameter->float_picture_rate) * parameter->N;
    rateControl->r = 2.0 * parameter->bit_rate / parameter->float_picture_rate;
    rateControl->d0i = 10.0 * rateControl->r / 31;
    rateControl->d0p = rateControl->d0i;                 /*      d0i * Kp, Kp = 1.0      */
    rateControl->d0b = rateControl->d0i * 1.4;   /*      d0i * Kb, Kb = 1.4      */
    rateControl->Tipb = parameter->bit_rate /
        (8 * parameter->float_picture_rate);
    rateControl->avg_act = 400.0;

    /* initial vbv_buffer filling for B_(-1) virtually */
    rateControl->vbvOccupy = parameter->vbv_buffer_size - vbvBufferStart -
        parameter->bit_rate / parameter->float_picture_rate;

    /*      print user input data into status file  */

    Init_DCT();
```

```
}

/* initialize bitstream and user parameters */

int InitParameter(tParameter *parameter)
{


    if (parameter->horizontal_size % 2)
        {
        printf("Error! horizontal picture size must be an even
            value due to 2:1 chroma decimation factor!\n");
        exit (-1);
        }

    /* coded picture: round up to nearest macroblock multiple */
    parameter->mb_width = (parameter->horizontal_size + 15) / 16;
    parameter->mb_height = (parameter->vertical_size + 15) / 16;

fprintf(stderr,"mb_width = %d, mb_height = %d\n", parameter->mb_width, parameter->mb_height);

    printf("coded picture is %d samples wider than source picture\n",
        parameter->mb_width*16 - parameter->horizontal_size);
    printf("coded picture is %d samples taller than source picture\n",
        parameter->mb_height*16 - parameter->vertical_size);

/* we need a distinction between coded and source picture size */

    /* coded picture dimensions */

    parameter->luminWidth = parameter->mb_width * kMBWidth;

    parameter->luminHeight = parameter->mb_height * kMBHeight;
    parameter->chromWidth = parameter->luminWidth / 2;
    /* assumes 4:2:0 */
    parameter->chromHeight = parameter->luminHeight / 2;

    parameter->numLuminPixels =
        parameter->luminWidth * parameter->luminHeight;

    parameter->numChromPixels =
        parameter->chromWidth * parameter->chromHeight;

    parameter->numSlices = parameter->mb_height;
    parameter->numMBsPerSlice = parameter->mb_width;
    parameter->numMacroBlocks = parameter->mb_height *
        parameter->mb_width;

    /* note:  these figures are based on source dimensions which
    could be up to 15 pels smaller than coded dimensions */

    parameter->luminHeightMB = parameter->luminHeight - kMBHeight;
    parameter->luminWidthMB  = parameter->luminWidth - kMBWidth;
    parameter->luminWidthDCT = parameter->luminWidth - kDCTSize;
    parameter->chromWidthB = parameter->chromWidth - kBWidth;
    parameter->chromWidthDCT = parameter->chromWidth - kDCTSize;

    parameter->mBLuminRowOffset = ((kMBHeight - 1) * parameter->luminWidth);
    parameter->mBChromRowOffset = ((kBHeight - 1) * parameter->chromWidth);

        parameter->load_intra_quantizer_matrix = 0;
        parameter->load_non_intra_quantizer_matrix = 0;

return (0);
}


/* this function should be called before sequence header is written
    to bitstream and after InitParameter() has been called */


int InitData(tData *Data, tParameter *parameter)
{
    int numLuminPixels, numChromPixels;
```

```
/* this can be updated to use the new tParameter elements */
    numLuminPixels = (16*parameter->mb_width) *
                (16*parameter->mb_height);
    numChromPixels = (8*parameter->mb_width) *
                (16*parameter->mb_height);


    /* Original picture array memory allocation */

    Data->luminPicture0 = Barray(numLuminPixels);
    Data->luminPicture1 = Barray(numLuminPixels);
    Data->luminPicture2 = Barray(numLuminPixels);

    Data->chromUPicture0 = Barray(numChromPixels);
    Data->chromUPicture1 = Barray(numChromPixels);
    Data->chromUPicture2 = Barray(numChromPixels);

    Data->chromVPicture0 = Barray(numChromPixels);
    Data->chromVPicture1 = Barray(numChromPixels);
    Data->chromVPicture2 = Barray(numChromPixels);

    /* Reconstructed picture array memory allocation */

    Data->luminPicture0R = Barray(numLuminPixels);
    Data->luminPicture1R = Barray(numLuminPixels);
    Data->luminPicture2R = Barray(numLuminPixels);

    Data->chromUPicture0R = Barray(numChromPixels);
    Data->chromUPicture1R = Barray(numChromPixels);
    Data->chromUPicture2R = Barray(numChromPixels);

    Data->chromVPicture0R = Barray(numChromPixels);
    Data->chromVPicture1R = Barray(numChromPixels);
    Data->chromVPicture2R = Barray(numChromPixels);

    /*      assign working pointers */

    Data->firstPredictY = Data->luminPicture0;
    Data->firstPredictU = Data->chromUPicture0;
    Data->firstPredictV = Data->chromVPicture0;
    Data->secondPredictY = Data->luminPicture1;
    Data->secondPredictU = Data->chromUPicture1;
    Data->secondPredictV = Data->chromVPicture1;
    Data->predictY = Data->luminPicture2;
    Data->predictU = Data->chromUPicture2;
    Data->predictV = Data->chromVPicture2;

    Data->firstPredictYR = Data->luminPicture0R;
    Data->firstPredictUR = Data->chromUPicture0R;
    Data->firstPredictVR = Data->chromVPicture0R;
    Data->secondPredictYR = Data->luminPicture1R;
    Data->secondPredictUR = Data->chromUPicture1R;
    Data->secondPredictVR = Data->chromVPicture1R;
    Data->predictYR = Data->luminPicture2R;
    Data->predictUR = Data->chromUPicture2R;
    Data->predictVR = Data->chromVPicture2R;

return(0);
}




BYTE    *Barray(
    int     size)
{
    BYTE    *a;

    a = (BYTE *) malloc((unsigned) size * sizeof(BYTE));

    if (!a)
    {
        printf("\nBarray: Failure in allocating arrary memory\n");
        exit(-1);
    }
```

```
    return a;
}

short *Sarray(
    int     size)
{
    short   *a;

    a = (short *) malloc((unsigned) size * sizeof(short));

    if (!a)
    {
        printf("\nSarray: Failure in allocating arrary memory\n");
        exit(-1);
    }

    return a;
}

int *Iarray(
    int     size)
{
    int   *a;

    a = (int *) malloc((unsigned) size * sizeof(int));

    if (!a)
    {
        printf("\nIarray: Failure in allocating arrary memory\n");
        exit(-1);
    }

    return a;
}


int Print_Parameters(
 tData *data,
 tParameter *parameter,
 tRateControl *rateControl)
{

    FILE    *fptr;

    if((fptr =fopen(parameter->statusFName, "w")) == NULL)
    {
        printf("\nopen error filename: %s\n", parameter->statusFName);
        exit(-1);
    }

   /* this section needs to be incorperated into a tracelevel routine */
    fprintf(fptr, "***** %s *****\n", parameter->statusFName);
    fprintf(fptr, "Input sequence : %s\n", parameter->inputSequence);
    fprintf(fptr, "Output sequence : %s\n", parameter->outputSequence);
    fprintf(fptr, "firstPicture = %d\tlastPicture = %d\n",
            parameter->firstPicture, parameter->lastPicture);
    fprintf(fptr, "Coding rate = %f kb/s\n", parameter->bit_rate/1000);
    fprintf(fptr, "VBV buffer size = %d bits\n", parameter->vbv_buffer_size);
    fprintf(fptr, "Picture rate = %f\tgroupOfPicture = %d\t",
            parameter->float_picture_rate, parameter->N);
    fprintf(fptr, "HalfPel = %d\n", parameter->halfPel);
    fprintf(fptr, "kLuminWidth = %d\tkLuminHeight = %d\tkChromWidth = %d\tkChromHeight =
%d\n", parameter->luminWidth, parameter->luminHeight, parameter->chromWidth, parameter-
>chromHeight);
    fprintf(fptr, "Xi = %.2f\tXp = %.2f\tXb = %.2f\n",
            rateControl->Xi, rateControl->Xp, rateControl->Xb);
    fprintf(fptr, "G = %d\tr = %.2f\tTipb = %d\n",
            rateControl->G, rateControl->r, rateControl->Tipb);
    fprintf(fptr, "d0i = %d\td0p = %d\td0b = %d\n",
            rateControl->d0i, rateControl->d0p, rateControl->d0b);
        fprintf(fptr, "M = %d, N = %d\n", parameter->M, parameter->N);
    fclose(fptr);

return (0);
}
```

B.2.15 makefile

```
# Makefile for ISO MPEG-1 Encoder.   Say "make pc" for IBM PC (MS-DOS)
# version using the GNU gcc djgpp compiler.

OBJ= encoder.o motion.o initial.o convert.o readpict.o decision.o perfidct.o \
perfdct.o dct.o transfer.o procpict.o stats.o reconstr.o \
ratectrl.o genbits.o writebit.o acvlc.o quantize.o writepic.o procseq.o

CFLAGS = -O2 -Wall

CC = gcc

encoder: $(OBJ)
    $(CC) $(CFLAGS) $(OBJ) -o encoder -lm

pc: encoder.exe

encoder.exe: encoder
    coff2exe encoder

# dependencies

dct.o: consts.h
motion.o: consts.h mpeg1.h encoder.pro
initial.o: consts.h mpeg1.h encoder.h encoder.pro
perfdct.o: consts.h mpeg1.h encoder.pro
perfidct.o: consts.h mpeg1.h encoder.pro
ratectrl.o: consts.h mpeg1.h flc.h encoder.pro
readpict.o: consts.h mpeg1.h encoder.pro
decision.o: consts.h mpeg1.h encoder.pro
quantize.o: consts.h mpeg1.h encoder.pro
stats.o: consts.h mpeg1.h encoder.pro
transfer.o: consts.h mpeg1.h encoder.pro
writebit.o: consts.h mpeg1.h encoder.pro
writepic.o: consts.h mpeg1.h encoder.pro
reconstr.o: consts.h mpeg1.h encoder.pro
procpict.o: consts.h mpeg1.h flc.h encoder.pro
encoder.o: consts.h mpeg1.h flc.h encoder.pro
convert.o: consts.h mpeg1.h encoder.pro
acvlc.o: consts.h mpeg1.h bitcount.h encoder.pro
procseq.o: consts.h mpeg1.h flc.h encoder.pro
genbits.o: consts.h mpeg1.h flc.h bitcount.h global.h encoder.pro
```

B.2.16 motion.c

```
/* File:  motion.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
```

```
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   motion.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:28:00  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
*/

/* Self-reminder: don't forget to update encoder.pro with new modules */

#include <stdio.h>
#include <stdlib.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

/* Exhaustive Minimum Absolute Difference block matching search */

short   GetMotionVector(
 BYTE  *recon_picture,      /* pointer to upper left corner of ref buffer */
 BYTE  *original_picture,   /* pointer to upper left corner of orig picture
                               co-incident with the current picture under the
                   encoding process */
 BYTE  *current_picture,     /* pointer to upper left corner of cur picture */
 short MV_array[][2],
 int   search_radius,
 int   half_pel_flag,                /* half pel flag */
 int   fileMV,
 char  *fName,
 tParameter      *parameter)
{
    FILE            *fptr;
/* The line below negates the usefulness of file MV  -- Chad  3/22/94 */
    int             noFile = 1;


    int     MB_row,         /* current macroblock row */
        MB_col,         /* current macroblock column */
        MB_origin_row,  /* current sample row */
        MB_origin_col,  /* current sample column */
        MB_number;      /* current macroblock number within picture */

    int     MV_count;       /* number of motion vectors, horizontal and
                   vertical, in picture */
    int     ret = 0;

    int     hor,            /* horizontal vector component */
        ver;            /* vertical vector component */

    BYTE    *current_MB;


    MV_count = parameter->numMacroBlocks * 2;

    if (fileMV)             /*      use file MV     */
    {
        if ((fptr = fopen (fName, "rb")) != NULL)
        {
            noFile = 0;

            fread(MV_array, sizeof(short), MV_count, fptr);
            fclose(fptr);
        }
    }
```

```
    if (noFile)
    {       /* find match for all macroblocks within target picture */

        for (MB_row = 0, MB_origin_row = 0, MB_number = 0;
                MB_row < parameter->numSlices; MB_row++,
                MB_origin_row += kMBHeight)
        {       /* compute pointer address of current MB */
            current_MB = (BYTE *) (current_picture + (MB_origin_row *parameter->luminWidth));

            /* macroblock row */
            for (MB_col = 0, MB_origin_col = 0; MB_col <
                    parameter->numMBsPerSlice; MB_col++,
                    MB_origin_col+= kMBWidth, MB_number++, current_MB
                    += kMBWidth)
            {

                ret = FullSearch(
                    &hor,
                    &ver,
                    search_radius,
                    half_pel_flag,
                    recon_picture,
                                   current_picture,
                    original_picture,

/* 3/21/94 A bug was reported by Mr. Hirofumi Nisikawa and Kinya Oosa.  Current
        has been changed to currrent1 */

                    current_MB,
                    MB_origin_row,
                    MB_origin_col,
                    parameter);


                MV_array[MB_number][0] = hor;
                MV_array[MB_number][1] = ver;

            }
            /* add tracelevel here */
            printf("Row %d of %d motion search completed.\n",
                MB_row, parameter->numSlices);
        }

        if ((fptr = fopen (fName, "wb")) != NULL)
        {
            fwrite(MV_array, sizeof(short), MV_count, fptr);
            fclose(fptr);
        }

        else return (-1);
    }

    return (ret);
}


int FullSearch(
    int                     *hor,
    int                     *ver,
    short                   search_radius,
    int                     half_pel_flag,
    BYTE                    *recon_picture,
        BYTE                    *current_picture,
    BYTE                    *original_picture,
    BYTE                    *current_MB,
    int                     MB_origin_row,
    int                     MB_origin_col,
    tParameter              *parameter)

/* need to update bufMV array to either be a passed pointer or a
return value */


    {
```

```
    int     AE_F = 0;           /* absolute error value */
    unsigned int    minF;

    BYTE    *prediction_MB;

    int     dis_radius,             /* distance along radius */
        search_quadrant,        /* sector */
        dis_diameter,           /* distance along diameter */
        diameter,               /* search diameter */
        dis_x_origin,           /* displacement from search origin */
        dis_y_origin,
        mxF,                    /* buffer location of x MV element */
        myF,
        mv[2];


    /* compute address of reference macroblock with no motion */
    prediction_MB = original_picture + (MB_origin_row * parameter->luminWidth) +
MB_origin_col;

    /* compute distortion of target center (no motion compensation) */
    AE_F = MAE(current_MB, prediction_MB, parameter);

    minF = AE_F;    /* default vector is zero vector */
    mv[0] = 0;
    mv[1] = 0;

    /* now perform spiral search around origin of current macroblock */
    for (dis_radius = 1; dis_radius <= search_radius; dis_radius++)
    {
        /* relative displacement along search axis */
        dis_x_origin = dis_radius;
        dis_y_origin = -dis_radius;     /* start in quandrant III */

        /* total search width is equal to twice search distance (f_code)
            like diameter is twice radius */
        diameter = dis_radius << 1;

        /* seach each of the four Cartesian co-ordinate sectors */
        for (search_quadrant = 0; search_quadrant < 4; search_quadrant++)
        {
            /* total linear search span */
            for (dis_diameter = 0; dis_diameter < diameter; dis_diameter++)
            {
                /* relative displacement from target center */
                myF = MB_origin_row + dis_y_origin;
                mxF = MB_origin_col + dis_x_origin;

/* need to change kLuminWidthMB into more readable constant */

                /* make sure search window boundaries are within legal picture boundaries */
                if ((mxF >= 0) && (mxF < (parameter->luminWidthMB + 1))
                    && (myF >= 0) &&
                        (myF < (parameter->luminHeightMB + 1)))
                {
                  /*      frame   */

                    /* compute prediction address pointer */
                    prediction_MB = original_picture +
                        (myF * parameter->luminWidth) + mxF;

                    AE_F = MAE(current_MB, prediction_MB, parameter);

                    /* update lowest distortion vector */
                    if (AE_F < minF)
                    {
                        minF = AE_F;
                        mv[0] = dis_x_origin;
                        mv[1] = dis_y_origin;
                    }
                }

                /* increment search displacement */
                switch (search_quadrant)
                {
                case 0:
```

```
                    dis_y_origin++;
                    break;
                case 1:
                    dis_x_origin--;
                    break;
                case 2:
                    dis_y_origin--;
                    break;
                case 3:
                    dis_x_origin++;
                    break;
                }
            }
        }
    }

    *hor = 2 * mv[0];          /*      frame MV half pel       */
    *ver = 2 * mv[1];          /*      frame MV half pel       */

    /*      half pel search */

    if (half_pel_flag)
    {
        minF = HalfPel(recon_picture, current_picture, hor, ver, MB_origin_row,
            MB_origin_col, minF, 1, parameter);
    }


return (0);
}


/* compute absolute error between prediction and target macroblock */

int MAE (BYTE           *current_MB,
     BYTE           *prediction_MB,
     tParameter     *parameter)
{
    int x, y;
    int absolute_error = 0;

    for (y = 0; y < kMBHeight; y++, current_MB += parameter->luminWidthMB,
        prediction_MB += parameter->luminWidthMB)
    {
        for (x = 0; x < kMBWidth; x++, current_MB++, prediction_MB++)
        {
            absolute_error += abs(*current_MB - *prediction_MB);
        }
    }

    return (absolute_error);
}


/* Half pel refinement as described in ISO/IEC 11172-2 Annex D.?.?.? */

/* implementation note:  simplify this routine using calls to MAE()  */
/* I know this routine can be made much more concise ! -- Chad */

unsigned int    HalfPel(
    BYTE                    *original_picture,
    BYTE                    *current_picture,
    int                     *hor,
    int                     *ver,
    int                     MB_origin_row,
    int                     MB_origin_col,
    unsigned int            minF,
    short                   offset,
    tParameter              *parameter)
{
    BYTE    *org, *org1, *curr;
    short   dx, dy;
        short   i,
                row,
                col;
```

```
        int             offset1,
                sum,
                diff,
                MB_origin_colA,
                MB_origin_rowA;


    MB_origin_colA = MB_origin_col + *hor / 2;
    MB_origin_rowA = MB_origin_row + *ver / 2;

    offset1 = offset * parameter->luminWidth - kMBWidth;
    dx = 0;
    dy = 0;

    /* 8 neighbour pel search on */
    for (i = 0; i < 8; i++)
    {
        switch (i)
        {
            case 0:         /*      upper left      */

                if ((MB_origin_rowA <= 0) || (MB_origin_colA <= 0))
                    break;

                org1 = (BYTE *) (original_picture + MB_origin_rowA *
                    parameter->luminWidth + MB_origin_colA -1);
                org = (BYTE *) (org1 - offset *
                    parameter->luminWidth);
                curr = (BYTE *) (current_picture + MB_origin_row *
                    parameter->luminWidth + MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, org1 += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, org1++, curr++)
                    {
                        /*      add 2 for integer rounding      */

                        diff = ((int) *org + (int) *(org + 1) +
                                (int) *org1 + (int) *(org1 + 1) + 2) / 4;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = -1;
                    dy = -1;
                }

                break;

            case 1:         /*      upper   */

                if (MB_origin_rowA <= 0)
                    break;

                org1 = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA);
                org = (BYTE *) (org1 - offset * parameter->luminWidth);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, org1 += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, org1++, curr++)
                    {
                        /*      add 1 for integer rounding      */

                        diff = ((int) *org + (int) *org1 + 1) / 2;

                        sum += abs (*curr - diff);
                    }
```

```
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = 0;
                    dy = -1;
                }

                break;

            case 2:         /*      upper right     */

                if ((MB_origin_rowA <= 0) || (MB_origin_colA >= (parameter->luminWidth -
kMBWidth - 1)))
                    break;

                org1 = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA);
                org = (BYTE *) (org1 - offset * parameter->luminWidth);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, org1 += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, org1++, curr++)
                    {
                        /*      add 2 for integer rounding      */

                        diff = ((int) *org + (int) *(org + 1) +
                                (int) *org1 + (int) *(org1 + 1) + 2) / 4;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = 1;
                    dy = -1;
                }

                break;

            case 3:         /*      left    */

                if (MB_origin_colA <= 0)
                    break;

                org = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA - 1);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, curr++)
                    {
                        /*      add 1 for integer rounding      */

                        diff = ((int) *org + (int) *(org + 1) + 1) / 2;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = -1;
                    dy = 0;
                }
```

```
                break;

            case 4:         /*      right   */

                if (MB_origin_colA >= (parameter->luminWidth - kMBWidth - 1))
                    break;

                org = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, curr++)
                    {
                        /*      add 1 for integer rounding      */

                        diff = ((int) *org + (int) *(org + 1) + 1) / 2;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = 1;
                    dy = 0;
                }

                break;

            case 5:         /*      lower left      */

                if ((MB_origin_rowA >= (parameter->luminHeight - kMBHeight - 1)) ||
                        (MB_origin_colA <= 0))
                    break;

                org = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA - 1);
                org1 = (BYTE *) (org + offset * parameter->luminWidth);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, org1 += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, org1++, curr++)
                    {
                        /*      add 2 for integer rounding      */

                        diff = ((int) *org + (int) *(org + 1) +
                                (int) *org1 + (int) *(org1 + 1) + 2) / 4;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = -1;
                    dy = 1;
                }

                break;

            case 6:         /*      lower   */

                if (MB_origin_rowA >= (parameter->luminHeight - kMBHeight - 1))
                    break;

                org = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA);
```

```
                org1 = (BYTE *) (org + offset * parameter->luminWidth);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, org1 += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, org1++, curr++)
                    {
                        /*      add 1 for integer rounding      */

                        diff = ((int) *org + (int) *org1 + 1) / 2;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = 0;
                    dy = 1;
                }

                break;

            case 7:         /*      lower right     */

                if ((MB_origin_rowA >= (parameter->luminHeight - kMBHeight - 1)) ||
                        (MB_origin_colA >= (parameter->luminWidth - kMBWidth - 1)))
                    break;

                org = (BYTE *) (original_picture + MB_origin_rowA * parameter->luminWidth +
MB_origin_colA);
                org1 = (BYTE *) (org + offset * parameter->luminWidth);
                curr = (BYTE *) (current_picture + MB_origin_row * parameter->luminWidth +
MB_origin_col);

                for (row = 0, sum = 0; row < kMBHeight; row += offset,
                        org += offset1, org1 += offset1, curr += offset1)
                {
                    for (col = 0; col < kMBWidth; col++, org++, org1++, curr++)
                    {
                        /*      add 2 for integer rounding      */

                        diff = ((int) *org + (int) *(org + 1) +
                                (int) *org1 + (int) *(org1 + 1) + 2) / 4;

                        sum += abs (*curr - diff);
                    }
                }

                if (sum < minF)
                {
                    minF = sum;
                    dx = 1;
                    dy = 1;
                }

                break;
        }
    }

    *hor += dx;
    *ver += dy;

    return (minF);
}
```

B.2.17 mpeg1.h

```
/* File: mpeg1.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
```

```
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   mpeg1.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.4  93/09/03  1:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

typedef struct t_Data           /*      picture data    */
{
    /*      working pointers        */
    BYTE *firstPredictY;    /*  1st prediction picture  */
    BYTE *firstPredictU;    /*  1st prediction picture  */
    BYTE *firstPredictV;    /*  1st prediction picture  */
    BYTE *secondPredictY;   /*  2nd prediction picture  */
    BYTE *secondPredictU;   /*  2nd prediction picture  */
    BYTE *secondPredictV;   /*  2nd prediction picture  */
    BYTE *predictY;         /*  B picture       */
    BYTE *predictU;         /*  B picture       */
    BYTE *predictV;         /*  B picture       */
    BYTE *predictY2;        /*  2nd B picture   */
    BYTE *predictU2;        /*  2nd B picture   */
    BYTE *predictV2;        /*  2nd B picture   */

    BYTE *firstPredictYR;   /*  reconstructed 1st prediction picture */
    BYTE *firstPredictUR;   /*  reconstructed 1st prediction picture */
    BYTE *firstPredictVR;   /*  reconstructed 1st prediction picture */
    BYTE *secondPredictYR;  /*  reconstructed 2nd prediction picture */
    BYTE *secondPredictUR;  /*  reconstructed 2nd prediction picture */
    BYTE *secondPredictVR;  /*  reconstructed 2nd prediction picture */
    BYTE *predictYR;        /*  reconstructed B picture */
    BYTE *predictUR;        /*  reconstructed B picture */
    BYTE *predictVR;        /*  reconstructed B picture */

    /*      storage arrays  */

    BYTE *luminPicture0;    /*      luminance of picture 0  */
    BYTE *luminPicture1;    /*      luminance of picture 1  */
```

```
    BYTE *luminPicture2;    /*      luminance of picture 2  */
    BYTE *chromUPicture0;   /*      chrominance of picture 0        */
    BYTE *chromUPicture1;   /*      chrominance of picture 1        */
    BYTE *chromUPicture2;   /*      chrominance of picture 2        */
    BYTE *chromVPicture0;
    BYTE *chromVPicture1;
    BYTE *chromVPicture2;

    BYTE *luminPicture0R;   /* reconstructed luminance of picture 0  */
    BYTE *luminPicture1R;   /* reconstructed luminance of picture 1  */
    BYTE *luminPicture2R;   /* reconstructed luminance of picture 2  */
    BYTE *chromUPicture0R;  /* reconstructed chrominance of picture 0  */
    BYTE *chromUPicture1R;  /* reconstructed chrominance of picture 1  */
    BYTE *chromUPicture2R;  /* reconstructed chrominance of picture 2  */
    BYTE *chromVPicture0R;
    BYTE *chromVPicture1R;
    BYTE *chromVPicture2R;
} tData;

typedef struct t_Parameter              /*      user parameter  */
{
        char    inputSequence[kMaxCharBufLen];
        char    outputSequence[kMaxCharBufLen];
        char    statusFName[kMaxCharBufLen];
        char    bitStreamFName[kMaxCharBufLen];

                int     sourceFileFormat; /* file type for input pictures */
                int     reconFileFormat;  /* file type for output pictures */
        int     firstPicture;
        int     lastPicture;
    int     temporal_reference;       /* GOP picture number */
    int     pictureNum;               /* sequence picture number */
        int     horizontal_size;  /* src&display picture width sample units */
        int     mb_width;         /* coded picture width in macroblock units */
        int     vertical_size;    /* display picture height in sample units */
        int     mb_height;        /* coded picture height in macroblocks units */
        int     pel_aspect_ratio; /* one of 14 legal values */
        int     picture_rate;     /* one of 8 legal values */
        double  float_picture_rate;  /* one of 8 legal values */

        double  bit_rate;           /* bit/sec */
        int     vbv_buffer_size;    /* bits, used to check vbv_delay */
        int     constrained_parameters_flag;

        int     N;    /* number of frames in GOP */
        int     M;    /* Distance between P pictures -- "M" factor */

        int     load_intra_quantizer_matrix;
        int     load_non_intra_quantizer_matrix;

        int     closed_GOP;
        int     broken_link;
    int     drop_frame_flag;
    long    time_code;

        int     luminWidth;  /* coded picture width in sample units */
        int     luminHeight; /* coded picture height in sample units */
        int     chromWidth;
        int     chromHeight;

        long    numLuminPixels; /* number of coded samples in picture */
        long    numChromPixels;

        int     numSlices;      /* number of slices in picture */
        int     numMBsPerSlice; /* number of macroblocks per slice */
        int     numMacroBlocks; /* number of macroblocks per picture */

    /* source picture dimensions in units of samples, clipped
       to not extend beyond the outermost macroblock or block origin */

        int     luminHeightMB;
        int     luminWidthMB;
        int     luminWidthDCT;
        int     chromWidthB;
        int     chromWidthDCT;
```

```
        int     mBLuminRowOffset;
        int     mBChromRowOffset;

        double  maxPictureDelay;
        int     maxVbvBufferSize;

        int     searchDistance;
        int     *forward_f_code;  /*  0-P, 1-1st B, 2-2nd B Picture   */
        int     *backward_f_code; /*  0-P, 1-1st B, 2-2nd B Picture   */
        int     *fRange;  /* 0-P Picture, 1-1st B Picture, 2-2nd B Picture   */
        int     *bRange; /*  0-P Picture, 1-1st B Picture, 2-2nd B Picture   */
        int     halfPel;  /* half pel search enable if set true */
        int     fileMV; /*  use file MV when possible if set true */

} tParameter;


typedef struct t_RateControl            /*      rate control    */
{
    int    headerBits;     /*  total number of header bits encoded   */
    int    totalGOPBits;   /*  total number of GOP bits encoded   */
    int    totalPBits;     /*  total number of current frame bits encoded */
    int    macroblock_address_increment;

    int    R;      /*      remaining # of bits assigned to the GOP */
    int    G;      /*      bit_rate * N / picture_rate     */
    double r;      /*      reaction parameter      */

    double avg_act;        /*      average actj of last frame   */
    double actjSum;        /*      sum of actj of current frame   */

    int     d0i;    /*  initial fullness of virtual buffers - I type    */
    int     d0p;    /*  initial fullness of virtual buffers - P type    */
    int     d0b;    /*  initial fullness of virtual buffers - B type    */
    int     dji;    /*  current fullness of virtual buffers - I type    */
    int     djp;    /*  current fullness of virtual buffers - P type    */
    int     djb;    /*  current fullness of virtual buffers - B type    */
    int     Tipb;   /*  bit_rate / (8 * picture_rate)   */
    int     Ti;     /*  target # bits for next frame in GOP - I type    */
    int     Tp;     /*  target # bits for next frame in GOP - P type    */
    int     Tb;     /*  target # bits for next frame in GOP - B type    */
    int     Tij;    /*  target # bits for each MB - I type  */
    int     Tpj;    /*  target # bits for each MB - P type  */
    int     Tbj;    /*  target # bits for each MB - B type  */
    int     bj;     /* # bits generated encoding all MBs up to MB j  */

    int     Np;     /*  # of P-frame remaining in current GOP  */
    int     Nb;     /*  # of B-frame remaining in current GOP  */

    double  Xi;     /*  global complexity measure - I type   */
    double  Xp;     /*  global complexity measure - P type  */
    double  Xb;     /*  global complexity measure - B type  */
    int     Si;     /*  # of bits generated by encoding current I frame */
    int     Sp;     /*  # of bits generated by encoding current P frame */
    int     Sb;     /*  # of bits generated by encoding current B frame */
    double  Qi;     /*  average quantization parameter - I type  */
    double  Qp;     /*  average quantization parameter - P type  */
    double  Qb;     /*  average quantization parameter - B type  */
    double  Qj;     /*  reference quantization parameter   */

    double  vbvOccupy; /* VBV buffer occupancy behave as B_n in VBV
                   operation */
    int     vbv_delay;
} tRateControl;
```

B.2.18 perfidct.c

```
/* File: PerformDct */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
```

```
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   perfdct.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:59  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

/* zz scan order offset */

static BYTE     zz[kDCTLen] =
{
     0,  1,  5,  6, 14, 15, 27, 28,
     2,  4,  7, 13, 16, 26, 29, 42,
     3,  8, 12, 17, 25, 30, 41, 43,
     9, 11, 18, 24, 31, 40, 44, 53,
    10, 19, 23, 32, 39, 45, 52, 54,
    20, 22, 33, 38, 46, 51, 55, 60,
    21, 34, 37, 47, 50, 56, 59, 61,
    35, 36, 48, 49, 57, 58, 62, 63
};


void    PerformDct(
    int             mbType,
    BYTE            currentMB[],
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             mbIndex)
{
    int             currentIndex, currentIndex1, currentIndex2, bufIndex,
            line, pix, row, col;

    float           dctBuf[kDCTLen], dctTrBuf[kDCTLen], dctTempBuf[kDCTLen];


    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex2 = line * kMBWidth + pix;
```

```
        for (row = 0, currentIndex = 0, currentIndex1 = currentIndex2;
                row < kDCTSize; row++, currentIndex1 += kMBWidthDCT)
        {
            for (col = 0; col < kDCTSize; col++,
                    currentIndex++, currentIndex1++)
            {
                dctBuf[currentIndex] = currentMBDiff[currentIndex1];
            }
        }

        /*      perform DCT     */

        dct8x8(dctBuf, dctTrBuf);

        /*      perform zz scan */

        for (row = 0; row < kDCTLen; row++)
                dctTempBuf[zz[row]] = dctTrBuf[row];

        /*      return transformed data         */

        for (row = 0, currentIndex = 0, currentIndex1 = currentIndex2;
                row < kDCTSize; row++, currentIndex1 += kMBWidthDCT)
        {
            for (col = 0; col < kDCTSize; col++,
                    currentIndex++, currentIndex1++)
            {
                currentMBDiff[currentIndex1] =
                        (dctTempBuf[currentIndex] >= 0.0) ?
                        (dctTempBuf[currentIndex] + 0.5) :
                        (dctTempBuf[currentIndex] - 0.5);

            }
        }
    }
}

/*      chrom DCT       */

/*      U       */

for (row = 0; row < kDCTLen; row++)
{
    dctBuf[row] = currentMBDiffU[row];
}

/*      perform DCT     */

dct8x8(dctBuf, dctTrBuf);

/*      perform zz scan */

for (row = 0; row < kDCTLen; row++)
        dctTempBuf[zz[row]] = dctTrBuf[row];

/*      return transformed data         */

for (currentIndex = 0; currentIndex < kDCTLen; currentIndex++)
{
    currentMBDiffU[currentIndex] = (dctTempBuf[currentIndex] >= 0.0) ?
            (dctTempBuf[currentIndex] + 0.5) :
            (dctTempBuf[currentIndex] - 0.5);
}

/*      V       */

for (row = 0; row < kDCTLen; row++)
{
    dctBuf[row] = currentMBDiffV[row];
}

/*      perform DCT     */

dct8x8(dctBuf, dctTrBuf);

/*      perform zz scan */
```

```
    for (row = 0; row < kDCTLen; row++)
        dctTempBuf[zz[row]] = dctTrBuf[row];

    /*      return transformed data         */

    for (currentIndex = 0; currentIndex < kDCTLen; currentIndex++)
    {
        currentMBDiffV[currentIndex] = (dctTempBuf[currentIndex] >= 0.0) ?
                (dctTempBuf[currentIndex] + 0.5) :
                (dctTempBuf[currentIndex] - 0.5);
    }
}

/* File: PerfIdct.c  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   PerfIdct.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

/* de-zz scan order offset */

static BYTE     dzz[64] =
{
     0,  1,  8, 16,  9,  2,  3, 10,
    17, 24, 32, 25, 18, 11,  4,  5,
    12, 19, 26, 33, 40, 48, 41, 34,
    27, 20, 13,  6,  7, 14, 21, 28,
    35, 42, 49, 56, 57, 50, 43, 36,
    29, 22, 15, 23, 30, 37, 44, 51,
    58, 59, 52, 45, 38, 31, 39, 46,
    53, 60, 61, 54, 47, 55, 62, 63
};


void    PerformIDct(
```

```
    int             mbType,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV)
{
    int             currentIndex, currentIndex1, currentIndex2, line, pix, row, col;
    float           dctBuf[kDCTLen], dctTempBuf[kDCTLen], dctTrBuf[kDCTLen];

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex2 = line * kMBWidth + pix;

            for (row = 0, currentIndex = 0, currentIndex1 = currentIndex2;
                    row < kDCTSize; row++, currentIndex1 += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++,
                        currentIndex++, currentIndex1++)
                {
                    /*      perform de-zz scan      */

                    dctTrBuf[dzz[currentIndex]] =
                            currentMBDiff[currentIndex1];
                }
            }

            /*      perform inverse DCT     */

            idct8x8(dctTrBuf, dctBuf);

            /*      return re-transformed data              */

            for (row = 0, currentIndex = 0, currentIndex1 = currentIndex2;
                    row < kDCTSize; row++, currentIndex1 += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++,
                        currentIndex++, currentIndex1++)
                {
                    currentMBDiff[currentIndex1] =
                            (dctBuf[currentIndex] > 0.0) ?
                            (dctBuf[currentIndex] + 0.5) :
                            (dctBuf[currentIndex] - 0.5);
                }
            }
        }
    }

    /*      chrom   */

    /*      U       */

    for (row = 0; row < kDCTLen; row++)
    {
        /*      perform de-zz scan      */

        dctTrBuf[dzz[row]] = currentMBDiffU[row];
    }

    /*      perform IDCT    */

    idct8x8(dctTrBuf, dctBuf);

    /*      return re-transformed data              */

    for (row = 0; row < kDCTLen; row++)
    {
        currentMBDiffU[row] = (dctBuf[row] > 0.0) ?
                (dctBuf[row] + 0.5) : (dctBuf[row] - 0.5);
    }

    /*      V       */

    for (row = 0; row < kDCTLen; row++)
    {
        /*      perform de-zz scan      */
```

```
        dctTrBuf[dzz[row]] = currentMBDiffV[row];
    }

    /*      perform DCT     */

    idct8x8(dctTrBuf, dctBuf);

    /*      return re-transformed data              */

    for (row = 0; row < kDCTLen; row++)
    {
        currentMBDiffV[row] = (dctBuf[row] > 0.0) ?
                (dctBuf[row] + 0.5) : (dctBuf[row] - 0.5);
    }
}
```

B.2.19 procpict.c

```
/* File: procpic.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */



/*
 *      $Log:   procpic.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.2  93/06/15  15:23:18  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "consts.h"
#include "mpeg1.h"
#include "flc.h"
#include "encoder.pro"
```

```
void    ProcPicture(
    tData           *data,
    tParameter      *parameter,
    tRateControl    *rateControl,
    BYTE            *bufY,          /* source picture buffer */
    BYTE            *bufU,
    BYTE            *bufV,
    BYTE            *bufYR,         /* reconstruction picture buffer */
    BYTE            *bufUR,
    BYTE            *bufVR,
    int             picture_coding_type,
    int             pictureIndex,   /* P=0,B1=1,B2=2,... */
    short           forwardMV[][2], /* motion vector field */
    short           backwardMV[][2],
    int             macroblock_type[],
    short           coded_block_pattern[],
    int             MQuant[])
{
    BYTE    currentMB[kMBLen];
    BYTE    currentMBU[kBLen];
    BYTE    currentMBV[kBLen];
    BYTE    *predictMBY, *predictMBU, *predictMBV;     /* forward reference */
    BYTE    *predictMBYB, *predictMBUB, *predictMBVB;  /* backwards reference */
    BYTE    *reconY, *reconU, *reconV;                 /* reconstruction pictute */
    int     currentMBDiff[kMBLen];                     /* DFD macroblock */
    int     currentMBDiffU[kBLen];
    int     currentMBDiffV[kBLen];

    static int      totalBits = 0;

    /* target location pointers. bufIndex is the reconstruction cursor */
    int             MB_row, MB_col, bufIndex, bufIndexC, MB_index;

    int             tempInt, newMbType, Txj;
    int             fRange, forward_f_code, bRange, backward_f_code;

    /*      set fRange and forward_f_code   */

    forward_f_code = parameter->forward_f_code[pictureIndex];
    fRange = parameter->fRange[pictureIndex];

    /*      set bRange and backward_f_code  */

    backward_f_code = parameter->backward_f_code[pictureIndex];
    bRange = parameter->bRange[pictureIndex];

    /* perform pre-frame analysis and write GOP header */

fprintf(stderr,"Preframe analysis now beginning\n");

    PreFrame(picture_coding_type, parameter, rateControl,
            forward_f_code, backward_f_code);


    switch (picture_coding_type)
    {
        case kIPicture:

            Txj = rateControl->Tij;
            break;

        case kPPicture:

            Txj = rateControl->Tpj;
            break;

        case kBPicture:

            Txj = rateControl->Tbj;
            break;
    }

printf(stderr, "Picture_type = %d\n", picture_coding_type);

    /* process each slice */
    for (MB_row = 0, MB_index = 0, bufIndex = 0, bufIndexC = 0;
```

```
            MB_row < parameter->numSlices; MB_row++, bufIndex +=
            parameter->mBLuminRowOffset,bufIndexC +=
                parameter->mBChromRowOffset)

    {
        switch (picture_coding_type)
        {
            case kBPicture:

                rateControl->djb += Write_Slice_Header(picture_coding_type, rateControl);
                break;

            case kPPicture:

                rateControl->djp += Write_Slice_Header(picture_coding_type, rateControl);
                break;

            case kIPicture:

                rateControl->dji += Write_Slice_Header(picture_coding_type, rateControl);
                break;
        }

fprintf(stderr, "Written slice header \n");

        /* process each macroblock */
        for (MB_col = 0; MB_col < parameter->numMBsPerSlice;
                MB_col++, MB_index++, bufIndex += kMBWidth, bufIndexC += kBWidth)
        {

            /* copy source MB into currentMB and currentMBDiff */

            CopyToBuf(bufIndex, bufIndexC, bufY, bufU, bufV, currentMB,
                    currentMBU, currentMBV, currentMBDiff,
                    currentMBDiffU, currentMBDiffV, parameter);

            /* update luma reconstruction pointers by bufIndex */

            reconY = bufYR + bufIndex;
            predictMBY = data->firstPredictY + bufIndex;
            predictMBYB = data->secondPredictY + bufIndex;

            /* update chrominance pointers   */

            reconU = bufUR + bufIndexC;
            reconV = bufVR + bufIndexC;
            predictMBU = data->firstPredictU + bufIndexC;
            predictMBV = data->firstPredictV + bufIndexC;
            predictMBUB = data->secondPredictU + bufIndexC;
            predictMBVB = data->secondPredictV + bufIndexC;

            /*      compute rate control status     */

            DoPreMBRC(picture_coding_type, rateControl, currentMB, &MQuant[MB_index]);

            if (! MB_col)
            {
                /*     Write slice header  */

                writeBits(cSliceStartCode + MB_row , bSliceStartCode, 1);
                writeBits(MQuant[MB_index], bQuantizerScale, 0);
                writeBits(cExtraBitSlice, bExtraBitSlice, 0);
            }

            /*      construct macroblock    */

            switch (picture_coding_type)
            {
                case kIPicture:

                    macroblock_type[MB_index] = ConstructI(reconY, reconU, reconV,
                            currentMB, currentMBU, currentMBV,
                            currentMBDiff, currentMBDiffU, currentMBDiffV,
                            parameter);
                    break;
```

```
    case kPPicture:

        macroblock_type[MB_index] = Construct(
                predictMBY, predictMBU, predictMBV,
                reconY, reconU, reconV,
                currentMB, currentMBU, currentMBV,
                forwardMV[MB_index], currentMBDiff,
                currentMBDiffU, currentMBDiffV,
                parameter);
        break;

    default:        /*      B picture       */

        macroblock_type[MB_index] = ConstructB(
                predictMBY, predictMBU, predictMBV,
                predictMBYB, predictMBUB, predictMBVB,
                reconY, reconU, reconV,
                currentMB, currentMBU, currentMBV,
                forwardMV[MB_index], backwardMV[MB_index],
                currentMBDiff, currentMBDiffU, currentMBDiffV, parameter);
        break;
}

/*      perform DCT and Zig-Zag scanning        */

PerformDct(macroblock_type[MB_index], currentMB, currentMBDiff,
        currentMBDiffU, currentMBDiffV, MB_index);

/*      perform quantization and de-quantization        */

if (macroblock_type[MB_index] & kIntraType)
    macroblock_type[MB_index] |= QuantizeI(&coded_block_pattern[MB_index],
            currentMBDiff, currentMBDiffU, currentMBDiffV,
            MQuant[MB_index]);
else
    macroblock_type[MB_index] |= Quantize(&coded_block_pattern[MB_index],
            currentMBDiff, currentMBDiffU, currentMBDiffV,
            MQuant[MB_index]);

if (macroblock_type[MB_index] & kCodedType)
{
    /*      check for modified MQuant       */

    if (MB_col && (MQuant[MB_index] != MQuant[MB_index - 1]))
        macroblock_type[MB_index] |= kModifiedMQ;
}
else if (MB_col)        /*      not coded       */
    MQuant[MB_index] = MQuant[MB_index - 1];

/*      update rate control status      */

tempInt = Write_MB(picture_coding_type, macroblock_type[MB_index],
        coded_block_pattern[MB_index], rateControl, parameter, MB_row,
        MB_col, forwardMV[MB_index], backwardMV[MB_index],
        currentMBDiff, currentMBDiffU, currentMBDiffV,
        MQuant[MB_index], fRange, forward_f_code, bRange,
        backward_f_code, &newMbType);

if (! tempInt)  /*      skipped MB      */
{
    /*      if here => MB_col != 0, MB_index != 0     */

    MQuant[MB_index] = MQuant[MB_index - 1];
}
else
{
    /*      reconstruct mb  */

    if (macroblock_type[MB_index] & kIntraType)
        IQuantizeI(coded_block_pattern[MB_index], currentMBDiff,
                currentMBDiffU, currentMBDiffV, MQuant[MB_index]);
    else
        IQuantize(coded_block_pattern[MB_index], currentMBDiff,
                currentMBDiffU, currentMBDiffV, MQuant[MB_index]);

    PerformIDct(macroblock_type[MB_index], currentMBDiff,
```

```
                        currentMBDiffU, currentMBDiffV);

                /* Copy current MB difference into reconstruction buffer */

                CopyFromBuf(picture_coding_type, macroblock_type[MB_index],
                        coded_block_pattern[MB_index], MB_row, MB_col, bufYR, bufUR,
                        bufVR, currentMBDiff, currentMBDiffU, currentMBDiffV,
                        MB_index, parameter);
            }

            /*      update quanitzation step size rate control data */

            switch(picture_coding_type)
            {
                case kBPicture:

                    rateControl->Qb += MQuant[MB_index];
                    break;

                case kPPicture:

                    rateControl->Qp += MQuant[MB_index];
                    break;

                case kIPicture:

                    rateControl->Qi += MQuant[MB_index];
                    break;
            }
        }
    }

    checkUpdateVbv(picture_coding_type, parameter->pictureNum, rateControl, parameter);

    PostFrame(picture_coding_type, rateControl, parameter);

    switch (picture_coding_type)
    {
        case kIPicture:

            totalBits += rateControl->Si;
            break;

        case kPPicture:

            totalBits += rateControl->Sp;
            break;

        case kBPicture:

            totalBits += rateControl->Sb;
            break;
    }

    /* Write reconstructed picture to file */
    if (WritePicture(parameter->pictureNum, picture_coding_type, bufYR, bufUR,
            bufVR, bufY, bufU, bufV, parameter))
    {
        writeBits (0, 0, -1);
        exit (-1);
    }
}
```

B.2.20 procseq.c

```
/* File:  procseq.c  */

/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
```

```
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   genbits.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "consts.h"
#include "mpeg1.h"
#include "flc.h"
#include "encoder.pro"

void ProcSequence(
    tParameter      *parameter,
    tData           *data,
    tRateControl    *rateControl)
{

    int             picture_coding_type;

    BYTE            *bufY, *bufU, *bufV;    /* source picture */
    BYTE            *bufYR, *bufUR, *bufVR; /* reconstruction picture */
    static short    (*forwardMV)[2], (*backwardMV)[2];
    int             index1, index2;
    int      temporal_reference_IP;
    static int      *macroblock_type;
    char            fName[16];

/*
*      macro block type:
*
*      bit:
*
*              1 - MC(1)/No MC (0)
*              2 - Intra(1)/Inter(0)
*              4 - Coded(1)/Not Coded(0)
*              5 - Forward MV
*              6 - Backward MV
*              7 - modified quantizer
*/
    static short *coded_block_pattern; /*coded block pattern*/
    static int *MQuant; /* macroblock quantization array */
    int     GOPnumber = -1;                 /* number of GOP's coded */

    /* allocate arrays */
    forwardMV = (short (*)[2])malloc(
             parameter->numMacroBlocks * sizeof(short [2]));
    backwardMV = (short (*)[2])malloc(
             parameter->numMacroBlocks * sizeof(short [2]));
```

```
    macroblock_type = Iarray(parameter->numMacroBlocks);
    coded_block_pattern = Sarray(parameter->numMacroBlocks);
    MQuant = Iarray(parameter->numMacroBlocks);

    /* write sequence header */
    Write_Sequence_Header(parameter, rateControl);

fprintf(stderr, "Sequence header written to buffer\n");

    /*      read first I source picture    */

    parameter->pictureNum = parameter->firstPicture;
    parameter->temporal_reference = temporal_reference_IP = 0;
        /* assume the first picture is I-picture */
    picture_coding_type = kIPicture;
    bufY = data->secondPredictY;
    bufU = data->secondPredictU;
    bufV = data->secondPredictV;
    bufYR = data->secondPredictYR;
    bufUR = data->secondPredictUR;
    bufVR = data->secondPredictVR;

    if (ReadPicture(parameter->pictureNum, bufY, bufU, bufV, parameter))
        exit(-1);

fprintf(stderr, "First picture read\n");

    /*      process each MB of first I picture      */

    ProcPicture(data, parameter, rateControl,
            bufY, bufU, bufV, bufYR, bufUR, bufVR,
            picture_coding_type, 0, forwardMV, backwardMV, macroblock_type,
            coded_block_pattern, MQuant);

fprintf(stderr,"Picture coding type= %d\n", picture_coding_type);

    /*      process sequence        */

    /* index1 points to the next I/P-picture */
    for (index1 = parameter->M;
            index1 <= (parameter->lastPicture - parameter->firstPicture);
            index1 += parameter->M)
    {
        /* Process BB...BP sequence */
        /* index2(!=0) points to the current B-picture */
        for (index2 = 0; index2 < parameter->M; index2++)
        {

            if(index2 == 0)
            {
                /* I,P-picture */
                parameter->pictureNum = parameter->firstPicture + index1;
            }
            else
            {
                /* B-picture */
                parameter->pictureNum = parameter->firstPicture
                    + index1 - parameter->M + index2;
            }

            /*      reassign reconstructed frame for next prediction                */

            switch (picture_coding_type)    /*      according to OLD picture_coding_type    */
            {
                case kIPicture:         /*      I and P pictures only   */
                case kPPicture:

                    bufY = data->secondPredictY;
                    bufU = data->secondPredictU;
                    bufV = data->secondPredictV;

                    data->secondPredictY = data->secondPredictYR;
                    data->secondPredictU = data->secondPredictUR;
                    data->secondPredictV = data->secondPredictVR;

                    data->secondPredictYR = bufY;
```

```
        data->secondPredictUR = bufU;
        data->secondPredictVR = bufV;

        break;
}

if (index2 == 0)
{
    /* I,P-picture */

    if (index1 % parameter->N == 0)
    {
    picture_coding_type = kIPicture;
    GOPnumber++;
    }
    else
    {
    picture_coding_type = kPPicture;
    }
}
else
{
    picture_coding_type = kBPicture;
}

/*      determine buffer location for picture storage   */

switch (picture_coding_type)
{
    case kIPicture:
    case kPPicture:

        /* source buffer */
        bufY = data->firstPredictY;
        bufU = data->firstPredictU;
        bufV = data->firstPredictV;

        /* rotate */
        data->firstPredictY = data->secondPredictY;
        data->firstPredictU = data->secondPredictU;
        data->firstPredictV = data->secondPredictV;

        data->secondPredictY = bufY;
        data->secondPredictU = bufU;
        data->secondPredictV = bufV;

        /* reconstruction buffer */
        bufYR = data->firstPredictYR;
        bufUR = data->firstPredictUR;
        bufVR = data->firstPredictVR;

        data->firstPredictYR = data->secondPredictYR;
        data->firstPredictUR = data->secondPredictUR;
        data->firstPredictVR = data->secondPredictVR;

        data->secondPredictYR = bufYR;
        data->secondPredictUR = bufUR;
        data->secondPredictVR = bufVR;

        break;

    default:

        bufY = data->predictY;
        bufU = data->predictU;
        bufV = data->predictV;

        bufYR = data->predictYR;
        bufUR = data->predictUR;
        bufVR = data->predictVR;

        break;
}
/* read source picture file into source buffer */

/* update temporal reference */
```

```
switch (picture_coding_type){
    case kIPicture:
    temporal_reference_IP = parameter->M - 1;
    parameter->temporal_reference = temporal_reference_IP;
    break;
    case kPPicture:
    temporal_reference_IP += parameter->M;
    parameter->temporal_reference = temporal_reference_IP;
    break;
    case kBPicture:
    parameter->temporal_reference = temporal_reference_IP + index2 - parameter->M;
    break;
}

if (ReadPicture(parameter->pictureNum, bufY, bufU, bufV, parameter))
{
    writeBits(0, 0, -1);    /*      flush buffer    */
    exit (-1);
}

/*      process picture */

/*      get motion vectors      */

switch (picture_coding_type)
{
    case kPPicture:

        /*      I/P to P - picture distance = 3 */

        if (parameter->halfPel)
            sprintf (fName, "MVFH%03d.%d", parameter->pictureNum,
                    parameter->searchDistance);
        else
            sprintf (fName, "MVF%03d.%d", parameter->pictureNum,
                    parameter->searchDistance);

        GetMotionVector(data->firstPredictY, data->firstPredictYR,
                bufY, forwardMV, parameter->M * parameter->searchDistance,
                parameter->halfPel, parameter->fileMV, fName, parameter);

        break;

    case kBPicture:

        /*      I/P to B - picture distance = index2    */

        if (parameter->halfPel)
            sprintf (fName, "MVFH%03d.%d", parameter->pictureNum,
                    parameter->searchDistance);
        else
            sprintf (fName, "MVF%03d.%d", parameter->pictureNum,
                    parameter->searchDistance);

        GetMotionVector(data->firstPredictY, data->firstPredictYR,
                bufY, forwardMV,
                index2 * parameter->searchDistance,
                parameter->halfPel, parameter->fileMV, fName, parameter);

        /*      B to I/P - picture distance = M - index2     */

        if (parameter->halfPel)
            sprintf (fName, "MVBH%03d.%d", parameter->pictureNum,
                    parameter->searchDistance);
        else
            sprintf (fName, "MVB%03d.%d", parameter->pictureNum,
                    parameter->searchDistance);

        GetMotionVector(data->secondPredictY, data->secondPredictYR,
                bufY, backwardMV,
                (parameter->M - index2) * parameter->searchDistance,
                parameter->halfPel, parameter->fileMV, fName, parameter);
        break;

}
```

```
            /*      process MBs     */

            ProcPicture(data, parameter, rateControl,
                    bufY, bufU, bufV, bufYR, bufUR, bufVR,
                    picture_coding_type, index2, forwardMV, backwardMV,
                    macroblock_type, coded_block_pattern, MQuant);
        }
    }

    writeBits(cSequenceEndCode, bSequenceEndCode, 1);       /* sequence_end_code */

    writeBits(0, 0, -1);    /*      flush buffer    */

    exit(0);
}
```

B.2.21 quantize.c

```
/* File: quantize.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   quant.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.3      93/08/20  21:15:53  hana
 * MPEG1 encoder minor revision
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

/*
 *      Default intra quantizer matrix
 *      On changing any part of this,
 *       you should make
 *         cLoadIntraIntraQuantierMatrix = 0
 *       in flc.h .
 */

#include <stdio.h>
#include "consts.h"
```

```
#include "mpeg1.h"
#include "encoder.pro"


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


BYTE    tQuant[64] =
{
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16
};

/* Quantize intra macroblock */
int     QuantizeI(
    short   *codedBPattern,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     MQuant)
{
    extern BYTE     tQuantIntra[],
            tQuant[];

    short   line, pix, row, col, quantIndex;
    int     currentIndex;
    float   tFloat;
    int     tInt;


    /*      lumin   */

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex = line * kMBWidth + pix;

            tFloat = currentMBDiff[currentIndex] / 8.0;
            currentMBDiff[currentIndex] =
                    (tFloat >= 0.0) ? tFloat + 0.5 : tFloat - 0.5;

            /* clip and saturate */
            if (currentMBDiff[currentIndex] > 255)
                currentMBDiff[currentIndex] = 255;
            else if (currentMBDiff[currentIndex] < -255)
                currentMBDiff[currentIndex] = -255;

            /*      exclude dc coeff        */

            quantIndex = 1;
            col = 1;
            currentIndex++;

            for (row = 0; row < kDCTSize; row++, currentIndex += kMBWidthDCT)
            {
                for ( ; col < kDCTSize; col++, currentIndex++, quantIndex++)
                {
                    tFloat = 8.0 * currentMBDiff[currentIndex] /
                            (float)(MQuant * tQuantIntra[quantIndex]);
```

```
                    tInt = (tFloat >= 0.0) ? tFloat + 0.5 : tFloat - 0.5;

                    /* clip and saturate */
                    if (tInt >= 255)
                        currentMBDiff[currentIndex] = 255;
                    else if (tInt <= -255)
                        currentMBDiff[currentIndex] = -255;
                    else
                        currentMBDiff[currentIndex] = tInt;
                }

                col = 0;
            }
        }
    }

    /*      chrom   */

    tFloat = currentMBDiffU[0] / 8.0;
    currentMBDiffU[0] = (tFloat >= 0.0) ? tFloat + 0.5 : tFloat - 0.5;

    if (currentMBDiffU[0] > 255)
        currentMBDiffU[0] = 255;
    else if (currentMBDiffU[0] < -255)
        currentMBDiffU[0] = -255;

    currentIndex = 1;

    for (row = 1; row < kDCTLen; row++, currentIndex++)
    {
        /*      U       */

        tFloat = 8.0 * currentMBDiffU[currentIndex] /
                (float)(MQuant * tQuantIntra[currentIndex]);
        tInt = (tFloat > 0) ? tFloat + 0.5 : tFloat - 0.5;

        if (tInt >= 255)
            currentMBDiffU[currentIndex] = 255;
        else if (tInt <= -255)
            currentMBDiffU[currentIndex] = -255;
        else
            currentMBDiffU[currentIndex] = tInt;
    }

    tFloat = currentMBDiffV[0] / 8.0;
    currentMBDiffV[0] = (tFloat >= 0.0) ? tFloat + 0.5 : tFloat - 0.5;

    if (currentMBDiffV[0] > 255)
        currentMBDiffV[0] = 255;
    else if (currentMBDiffV[0] < -255)
        currentMBDiffV[0] = -255;

    currentIndex = 1;

    for (row = 1; row < kDCTLen; row++, currentIndex++)
    {
        /*      V       */

        tFloat = 8.0 * currentMBDiffV[currentIndex] /
                (float)(MQuant * tQuantIntra[currentIndex]);
        tInt = (tFloat > 0) ? tFloat + 0.5 : tFloat - 0.5;

        if (tInt >= 255)
            currentMBDiffV[currentIndex] = 255;
        else if (tInt <= -255)
            currentMBDiffV[currentIndex] = -255;
        else
            currentMBDiffV[currentIndex] = tInt;
    }

    *codedBPattern = 0x3F;          /*      all blocks coded        */

    return(kCodedType);
}
```

```
int     Quantize(
    short   *codedBPattern,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     MQuant)
{
    extern BYTE     tQuantIntra[], tQuant[];

    short           code, line, pix, row, col, quantIndex;
    int             currentIndex;
    float           tFloat, tFloat2;
    int             tInt, tInt2;

    /*      lumin   */

    *codedBPattern = 0;

    code = 0x20;

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex = line * kMBWidth + pix;

            quantIndex = 0;

            for (row = 0; row < kDCTSize; row++, currentIndex += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++,
                        currentIndex++, quantIndex++)
                {
                    tFloat = 16.0 * currentMBDiff[currentIndex] /
                            (float) tQuant[quantIndex];
                    tInt = (tFloat >= 0.0) ? tFloat + 0.5 : tFloat - 0.5;

                    if (MQuant % 2) /*      odd             */
                        tInt2 = tInt / (2 * MQuant);
                    else                            /*      even    */
                    {
                        if (tInt < 0)
                            tInt2 = (tInt + 1) / (2 * MQuant);
                        else    /*      > 0, case == 0 is truncated to 0 anyway */
                            tInt2 = (tInt - 1) / (2 * MQuant);
                    }

                    if (tInt2 >= 255)
                        currentMBDiff[currentIndex] = 255;
                    else if (tInt2 <= -255)
                        currentMBDiff[currentIndex] = -255;
                    else
                        currentMBDiff[currentIndex] = tInt2;

                    if (currentMBDiff[currentIndex])
                        *codedBPattern |= code;         /*      non-zero coeff  */
                }
            }

            code >>= 1;
        }
    }

    /*      chrom   */

    currentIndex = 0;

    for (row = 0; row < kDCTLen; row++, currentIndex++)
    {
        /*      U       */

        tFloat = 16.0 * currentMBDiffU[currentIndex] /
                (float) tQuant[currentIndex];
        tInt = (tFloat >= 0) ? tFloat + 0.5 : tFloat - 0.5;
```

```
        if (MQuant % 2) /*      odd             */
            tInt2 = tInt / (2 * MQuant);
        else                            /*      even    */
        {
            if (tInt < 0)
                tInt2 = (tInt + 1) / (2 * MQuant);
            else
                tInt2 = (tInt - 1) / (2 * MQuant);
        }

        if (tInt2 >= 255)
            currentMBDiffU[currentIndex] = 255;
        else if (tInt2 <= -255)
            currentMBDiffU[currentIndex] = -255;
        else
            currentMBDiffU[currentIndex] = tInt2;

        if (currentMBDiffU[currentIndex])
            *codedBPattern |= 0x02;         /*      non-zero coeff  */
    }

    currentIndex = 0;

    for (row = 0; row < kDCTLen; row++, currentIndex++)
    {
        /*      V       */

        tFloat = 16.0 * currentMBDiffV[currentIndex] /
                (float) tQuant[currentIndex];
        tInt = (tFloat >= 0) ? tFloat + 0.5 : tFloat - 0.5;

        if (MQuant % 2) /*      odd             */
            tInt2 = tInt / (2 * MQuant);
        else                            /*      even    */
        {
            if (tInt < 0)
                tInt2 = (tInt + 1) / (2 * MQuant);
            else
                tInt2 = (tInt - 1) / (2 * MQuant);
        }

        if (tInt2 >= 255)
            currentMBDiffV[currentIndex] = 255;
        else if (tInt2 <= -255)
            currentMBDiffV[currentIndex] = -255;
        else
            currentMBDiffV[currentIndex] = tInt2;

        if (currentMBDiffV[currentIndex])
            *codedBPattern |= 0x01;         /*      non-zero coeff  */
    }

    if (*codedBPattern)
        return(kCodedType);
    else
        return(0);
}


/* Inverse quantize intra macorblock */
void    IQuantizeI(
    short   codedBPattern,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     MQuant)
{
    short   code, line, pix, row, col, quantIndex;
    int     currentIndex;
    int     tInt;

    /*      lumin   */

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
```

```
        {
            currentIndex = line * kMBWidth + pix;

            currentMBDiff[currentIndex] *= 8;

            if (currentMBDiff[currentIndex] < -2048)
                currentMBDiff[currentIndex] = -2048;
            else if (currentMBDiff[currentIndex] > 2047)
                currentMBDiff[currentIndex] = 2047;

            /*      exclude dc coeff        */

            currentIndex++;

            for (row = 0, col = 1, quantIndex = 1; row < kDCTSize;
                    row++, col = 0, currentIndex += kMBWidthDCT)
            {
                for ( ; col < kDCTSize; col++, currentIndex++, quantIndex++)
                {
                    if (currentMBDiff[currentIndex] != 0)
                    {
                        /*      rec = MQuant * 2 * QAC * WI / 16        */

                        tInt = MQuant * currentMBDiff[currentIndex] *
                                (int) tQuantIntra[quantIndex] / 8;

                        if (! (tInt % 2))               /*      even    */
                        {
                            if (tInt < 0)
                                tInt += 1;
                            else if (tInt > 0)
                                tInt -= 1;
                        }

                        if (tInt <= -2048)
                            currentMBDiff[currentIndex] = -2048;
                        else if (tInt >= 2047)
                            currentMBDiff[currentIndex] = 2047;
                        else
                            currentMBDiff[currentIndex] = tInt;
                    }
                }
            }
        }
    }

    /*      chrom   */

    currentMBDiffU[0] *= 8;

    if (currentMBDiffU[0] < -2048)
        currentMBDiffU[0] = -2048;
    else if (currentMBDiffU[0] > 2047)
        currentMBDiffU[0] = 2047;

    currentIndex = 1;

    for (row = 1; row < kDCTLen; row++, currentIndex++)
    {
        /*      U       */

        if (currentMBDiffU[currentIndex] != 0)
        {
            /*      rec = MQuant * 2 * QAC * WI / 16        */

            tInt = MQuant * currentMBDiffU[currentIndex] *
                    (int) tQuantIntra[currentIndex] / 8;

            if (! (tInt % 2))               /*      even    */
            {
                if (tInt < 0)
                    tInt += 1;
                else if (tInt > 0)
                    tInt -= 1;
            }
```

```
            if (tInt <= -2048)
                currentMBDiffU[currentIndex] = -2048;
            else if (tInt >= 2047)
                currentMBDiffU[currentIndex] = 2047;
            else
                currentMBDiffU[currentIndex] = tInt;
        }
    }

    currentMBDiffV[0] *= 8;

    if (currentMBDiffV[0] < -2048)
        currentMBDiffV[0] = -2048;
    else if (currentMBDiffV[0] > 2047)
        currentMBDiffV[0] = 2047;

    currentIndex = 1;

    for (row = 1; row < kDCTLen; row++, currentIndex++)
    {
        /*      V       */

        if (currentMBDiffV[currentIndex] != 0)
        {
            /*      rec = MQuant * 2 * QAC * WI / 16        */

            tInt = MQuant * currentMBDiffV[currentIndex] *
                    (int) tQuantIntra[currentIndex] / 8;

            if (! (tInt % 2))               /*      even    */
            {
                if (tInt < 0)
                    tInt += 1;
                else if (tInt > 0)
                    tInt -= 1;
            }

            if (tInt <= -2048)
                currentMBDiffV[currentIndex] = -2048;
            else if (tInt >= 2047)
                currentMBDiffV[currentIndex] = 2047;
            else
                currentMBDiffV[currentIndex] = tInt;
        }
    }
}

/* Inverse quantize non-intra macroblock */

void    IQuantize(
    short   codedBPattern,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     MQuant)
{
    short   code, line, pix, row, col, quantIndex, offset1, offset2;
    int     currentIndex;
    float   tFloat, tFloat2;
    int     tInt, tInt2;

    /*      lumin   */

    code = 0x20;

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            if (! (codedBPattern & code))
            {
                code >>= 1;
                continue;
            }

            code >>= 1;
```

```
            currentIndex = line * kMBWidth + pix;

            for (row = 0, quantIndex = 0; row < kDCTSize; row++,
                    currentIndex += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++,
                        currentIndex++, quantIndex++)
                {
                    if (currentMBDiff[currentIndex])
                    {
                        if (currentMBDiff[currentIndex] > 0)
                            tInt = (2 * currentMBDiff[currentIndex] + 1) *
                                    MQuant * (int) tQuant[quantIndex] / 16;
                        else    /*      < 0             */
                            tInt = (2 * currentMBDiff[currentIndex] - 1) *
                                    MQuant * (int) tQuant[quantIndex] / 16;

                        if (! (tInt % 2))               /*      even    */
                        {
                            if (tInt < 0)
                                tInt += 1;
                            else if (tInt > 0)
                                tInt -= 1;
                        }

                        if (tInt <= -2048)
                            currentMBDiff[currentIndex] = -2048;
                        else if (tInt >= 2047)
                            currentMBDiff[currentIndex] = 2047;
                        else
                            currentMBDiff[currentIndex] = tInt;
                    }
                }
            }
        }
    }

    /*      chrom   */

    if (codedBPattern & 0x2)
    {
        /*      U       */

        for (row = 0, currentIndex = 0; row < kDCTLen; row++, currentIndex++)
        {
            if (currentMBDiffU[currentIndex])
            {
                if (currentMBDiffU[currentIndex] > 0)
                    tInt = (2 * currentMBDiffU[currentIndex] + 1) *
                            MQuant * (int) tQuant[currentIndex] / 16;
                else
                    tInt = (2 * currentMBDiffU[currentIndex] - 1) *
                            MQuant * (int) tQuant[currentIndex] / 16;

                if (! (tInt % 2))               /*      even    */
                {
                    if (tInt < 0)
                        tInt += 1;
                    else if (tInt > 0)
                        tInt -= 1;
                }

                if (tInt <= -2048)
                    currentMBDiffU[currentIndex] = -2048;
                else if (tInt >= 2047)
                    currentMBDiffU[currentIndex] = 2047;
                else
                    currentMBDiffU[currentIndex] = tInt;
            }
        }
    }

    if (codedBPattern & 0x1)
    {
        /*      V       */
```

```
        for (row = 0, currentIndex = 0; row < kDCTLen; row++, currentIndex++)
        {
            if (currentMBDiffV[currentIndex])
            {
                if (currentMBDiffV[currentIndex] > 0)
                    tInt = (2 * currentMBDiffV[currentIndex] + 1) *
                            MQuant * (int) tQuant[currentIndex] / 16;
                else
                    tInt = (2 * currentMBDiffV[currentIndex] - 1) *
                            MQuant * (int) tQuant[currentIndex] / 16;

                if (! (tInt % 2))               /*      even    */
                {
                    if (tInt < 0)
                        tInt += 1;
                    else if (tInt > 0)
                        tInt -= 1;
                }

                if (tInt <= -2048)
                    currentMBDiffV[currentIndex] = -2048;
                else if (tInt >= 2047)
                    currentMBDiffV[currentIndex] = 2047;
                else
                    currentMBDiffV[currentIndex] = tInt;
            }
        }
    }
}
```

B.2.22 ratectrl.c

```
/* File: preframe.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   preframe.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
```

```
 *
 * Revision 1.1.1.1  93/03/29  11:28:00  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "flc.h"
#include "encoder.pro"

/* move this into rateControl later */
#define Kp      1.0
#define Kb      1.4

/* Perform pre-frame analysis as per TM Step 1 */
void    PreFrame(
    short           picture_coding_type,
    tParameter      *parameter,
    tRateControl    *rateControl,
    short           forward_f_code,
    short           backward_f_code)
{
    static int firstIPicture = TRUE;

    rateControl->totalPBits = 0;

fprintf(stderr,"Now beginning pre-frame analysis\n");
    switch (picture_coding_type)
    {
        case kBPicture1:                /*      B frame */
        case kBPicture2:

            rateControl->Sb = 0;
            rateControl->Qb = 0.0;
            rateControl->djb = rateControl->d0b;

            rateControl->Tb = rateControl->R / (rateControl->Nb +
                    (rateControl->Np * Kb * rateControl->Xp /
                    (Kp * rateControl->Xb)));
            if (rateControl->Tb < rateControl->Tipb)
                rateControl->Tb = rateControl->Tipb;

            /* calculate vbv_delay */
            rateControl->vbv_delay
                = calcVbvDelay(picture_coding_type, rateControl, parameter);

            /*      bit count - picture     */

            rateControl->djb += Write_Picture_Header(picture_coding_type, rateControl,
                    forward_f_code, backward_f_code,
                        parameter->temporal_reference);

            rateControl->Tbj = (rateControl->Tb - rateControl->Sb) /
                    parameter->numMacroBlocks;

            break;

        case kPPicture:         /*      P frame */

            rateControl->Sp = 0;
            rateControl->Qp = 0.0;
            rateControl->djp = rateControl->d0p;

            rateControl->Tp = rateControl->R / (rateControl->Np +
                    (rateControl->Nb * Kp * rateControl->Xb /
                    (Kb * rateControl->Xp)));
            if (rateControl->Tp < rateControl->Tipb)
                rateControl->Tp = rateControl->Tipb;

            /* calculate vbv_delay */
            rateControl->vbv_delay
```

```
                = calcVbvDelay(picture_coding_type, rateControl, parameter);

            /*      bit count - picture     */

            rateControl->djp += Write_Picture_Header(picture_coding_type, rateControl,
                    forward_f_code, backward_f_code,
                        parameter->temporal_reference);

            rateControl->Tpj = (rateControl->Tp - rateControl->Sp) /
                    parameter->numMacroBlocks;

            break;

        case kIPicture:         /*      I frame */

            if(firstIPicture) {
                rateControl->Si = rateControl->headerBits;
                firstIPicture = FALSE;
            }
            else
                rateControl->Si = 0;
            rateControl->Qi = 0.0;
            rateControl->dji = rateControl->d0i;

fprintf(stderr, "Now calcuating rate control parameters for I picture\n");

            rateControl->R += rateControl->G;
            rateControl->Np = parameter->N / parameter->M - 1;
            rateControl->Nb = parameter->N - rateControl->Np - 1;

fprintf(stderr, "Np = %d, Nb = %d\n", rateControl->Np, rateControl->Nb);
            rateControl->Ti = rateControl->R / (1 + (rateControl->Np *
                    rateControl->Xp / (Kp * rateControl->Xi)) +
                    (rateControl->Nb * rateControl->Xb / (Kb *
                    rateControl->Xi)));
            if (rateControl->Ti < rateControl->Tipb)
                rateControl->Ti = rateControl->Tipb;

            /*      bit count - group of picture    */

fprintf(stderr, "Now calling GOP_Header function\n");

            rateControl->dji += Write_GOP_Header(parameter,rateControl);

            /* calculate vbv_delay */
            rateControl->vbv_delay
                = calcVbvDelay(picture_coding_type, rateControl, parameter);
            /*      bit count - picture     */

            rateControl->dji += Write_Picture_Header(picture_coding_type, rateControl,
                    forward_f_code, backward_f_code, parameter->temporal_reference);

            rateControl->Tij = (rateControl->Ti - rateControl->Si) /
                    parameter->numMacroBlocks;

            break;
    }

    rateControl->actjSum  = 0.0;
}


void    PostFrame(
    short           picture_coding_type,
    tRateControl    *rateControl,
    tParameter      *parameter)
{
    switch (picture_coding_type)
    {
        case kBPicture1:        /*      B frame         */
        case kBPicture2:

            rateControl->Qb /= parameter->numMacroBlocks;
            rateControl->Xb = rateControl->Sb * rateControl->Qb;
            rateControl->R -= rateControl->Sb;
            rateControl->d0b = rateControl->djb;
```

```
            rateControl->Nb--;
            break;

        case kPPicture:         /*      P frame         */

            rateControl->Qp /= parameter->numMacroBlocks;
            rateControl->Xp = rateControl->Sp * rateControl->Qp;
            rateControl->R -= rateControl->Sp;
            rateControl->d0p = rateControl->djp;
            rateControl->Np--;
            break;

        case kIPicture:         /*      I frame         */

            rateControl->Qi /= parameter->numMacroBlocks;
            rateControl->Xi = rateControl->Si * rateControl->Qi;
            rateControl->R -= rateControl->Si;
            rateControl->d0i = rateControl->dji;
            break;
    }

    rateControl->avg_act = rateControl->actjSum / parameter->numMacroBlocks;
}


void    DoPreMBRC(
    short           picture_coding_type,
    tRateControl    *rateControl,
    BYTE            *currentMB,
    int             *MQuant)
{
    short   line, pix, row, col, currentIndex, currentIndex1;
    float   sum, P_mean, var_sblk, min_var_sblk, actj, N_actj;

    /*      calc actj       */

    min_var_sblk = 1.0e10;  /*      initialize to something large   */

    /*      frame   */

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex1 = line * kMBWidth + pix;

            /*      P_mean  */

            sum = 0.0;

            for (row = 0, currentIndex = currentIndex1; row < kDCTSize;
                    row++, currentIndex += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++, currentIndex++)
                {
                    sum += currentMB[currentIndex];
                }
            }

            P_mean = sum / kDCTLen;

            /*      var_sblk        */

            sum = 0.0;

            for (row = 0, currentIndex = currentIndex1; row < kDCTSize;
                    row++, currentIndex += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++, currentIndex++)
                {
                    var_sblk = ((float) currentMB[currentIndex] - P_mean);
                    sum += var_sblk * var_sblk;
                }
            }

            var_sblk = sum / kDCTLen;
```

```
            if (var_sblk < min_var_sblk)
                min_var_sblk = var_sblk;
        }
    }

    actj = 1.0 + min_var_sblk;

    rateControl->actjSum += actj;

    N_actj = (2.0 * actj + rateControl->avg_act) /
            (actj + 2.0 * rateControl->avg_act);

    switch (picture_coding_type)
    {
        case kBPicture1:
        case kBPicture2:

            rateControl->Qj = rateControl->djb * 31.0 / rateControl->r;
            break;

        case kPPicture:

            rateControl->Qj = rateControl->djp * 31.0 / rateControl->r;
            break;

        case kIPicture:

            rateControl->Qj = rateControl->dji * 31.0 / rateControl->r;
            break;
    }

    /* step 3 ? */
    *MQuant = rateControl->Qj * N_actj + 0.5;

    if (*MQuant < 1)
        *MQuant = 1;
    else if (*MQuant > 31)
        *MQuant = 31;

}


int     calcVbvDelay(
    short           picture_coding_type,
    tRateControl    *rateControl,
    tParameter      *parameter)
{
    int     delay;
    float   bufferAst;

fprintf(stderr,"Now calculating vbvdelay\n");

    bufferAst = rateControl->vbvOccupy + (float)parameter->bit_rate /
        parameter->float_picture_rate;

    switch(picture_coding_type) { /* subtract bits for picture header */
        case kBPicture1:                /*      B frame */
        case kBPicture2:
            bufferAst -= (rateControl->Sb + bPictureStartCode);
            break;
        case kPPicture:         /*      P frame */
            bufferAst -= (rateControl->Sp + bPictureStartCode);
            break;
        case kIPicture:                 /*      I frame */
            bufferAst -= (rateControl->Si + bPictureStartCode);
            break;
    }

    delay = (int)(90000 * bufferAst / parameter->bit_rate);

fprintf(stderr,"vbvdelay = %d \n", delay);

    return(delay);
}
```

```
void    checkUpdateVbv(
    short           picture_coding_type,
    int             pictureNum,
    tRateControl    *rateControl,
    tParameter      *parameter)
{
    float   bitsThisFrame,
        bitRatePerFrame;

    switch(picture_coding_type) {
        case kBPicture1:                /*      B frame */
        case kBPicture2:
            bitsThisFrame = rateControl->Sb;
            break;
        case kPPicture:         /*      P frame */
            bitsThisFrame = rateControl->Sp;
            break;
        case kIPicture:                 /*      I frame */
            bitsThisFrame = rateControl->Si;
            break;
    }

    bitRatePerFrame = (float)parameter->bit_rate / parameter->float_picture_rate;

    /* check if d_n+1 > B_n + ( 2 R/P ) - B */
    if( bitsThisFrame <=
       rateControl->vbvOccupy + 2.0 * bitRatePerFrame - parameter->vbv_buffer_size)
        fprintf(stderr, "VBV Buffer Over Flow : picNum = %d, coded bits = %d\n",
                pictureNum, (int)bitsThisFrame);
    /* check if d_n+1 <= B_n + ( R/P ) */
    else if(bitsThisFrame > rateControl->vbvOccupy + bitRatePerFrame)
        fprintf(stderr, "VBV Buffer Under Flow : picNum = %d, coded bits = %d\n",
                pictureNum, (int)bitsThisFrame);

    /* update vbv_buffer */
    rateControl->vbvOccupy += bitRatePerFrame;
    rateControl->vbvOccupy -= bitsThisFrame;

    return;
}
```

B.2.23 readpict.c

```
/* File: readpic.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
```

```
 *
 */

/*
 *      $Log:   readpic.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

/* need to add .tga, port. bitmap, and .yuv */

short   ReadPicture(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter)
{
    int ret;

    switch(parameter->sourceFileFormat)
    {
    case 0:
        ret = Read_SIF(pictureNumber, y_buffer, cb_buffer, cr_buffer, parameter);
        break;
    case 1:
        ret = Read_TGA(pictureNumber, y_buffer, cb_buffer, cr_buffer, parameter);
        break;
    case 2:
        ret = Read_PPM(pictureNumber, y_buffer, cb_buffer, cr_buffer, parameter);
        break;
    case 3:
        ret = Read_YUV(pictureNumber, y_buffer, cb_buffer, cr_buffer, parameter);
        break;

    }
    return (ret);
}

short   Read_SIF(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter)
{
    int ret;
    int     i, j;

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    static BYTE *buffer;
    BYTE    *b;
    BYTE    *y;
    BYTE    *u;
    BYTE    *v;

    if (!buffer)
    {
      buffer = (BYTE *)malloc(parameter->luminWidth*2);
    }
    sprintf (fname, "%s%03d.SIF", parameter->inputSequence, pictureNumber);
```

```
    if ((fptr = fopen(fname, "rb")) == NULL)
    {
        printf("\nopen ERROR on filename: %s", fname);
        return (1);
    }

    y = y_buffer;
    u = cb_buffer;
    v = cr_buffer;

    for (i = 0; i < parameter->luminHeight; i++)
    {
        fread(buffer, sizeof(BYTE), parameter->luminWidth*2, fptr);

        for (j = 0, b = buffer; j < parameter->luminWidth/2; j++)
        {
            *(u++) = *(b++);
            *(y++) = *(b++);
            *(v++) = *(b++);
            *(y++) = *(b++);
        }
    }

    fclose(fptr);

    /* decimate chrominance in vertical direction */
    u = cb_buffer;
    v = cr_buffer;


    convert422to420(1, u, parameter->chromWidth,
         parameter->chromHeight*2, FORWARD, parameter);
    convert422to420(1, v, parameter->chromWidth,
         parameter->chromHeight*2, FORWARD, parameter);

    return (0);
}

short   Read_TGA(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter)
{

fprintf(stderr,"Sorry -- TGA file format has not been implemented yet.");

exit(-1);
}


short   Read_PPM(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter)
{


fprintf(stderr,"Sorry -- PPM file format has not been implemented yet.");
exit(-1);
}

/* Read Stanford-compatible raw 4:2:0 YUV file format */
short   Read_YUV(
    int             pictureNumber,
    BYTE            *y_buffer,
    BYTE            *cb_buffer,
    BYTE            *cr_buffer,
    tParameter      *parameter)
{
    int ret;
    int     i, j, col, line;
```

```
    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    static  BYTE *buffer;
    BYTE    *b;
    BYTE    *y;
    BYTE    *u;
    BYTE    *v;

        int lw = parameter->luminWidth;
        int lh = parameter->luminHeight;
        int cw = parameter->chromWidth;
        int ch = parameter->chromHeight;

fprintf(stderr,"Reading YUV input picture\n");

    if (!buffer)
    {
      buffer = (BYTE *)malloc(parameter->luminWidth);
    }

fprintf(stderr, "Malloced buffer\n");

    sprintf (fname, "%s%03d.yuv", parameter->inputSequence, pictureNumber);

    if ((fptr = fopen(fname, "rb")) == NULL)
    {
        fprintf(stderr,"\nopen ERROR on filename: %s", fname);
        return (1);
    }

fprintf(stderr, "Opened file\n");

    y = y_buffer;

    for (line = 0; line < lh; line++)
    {
        fread(buffer, sizeof(BYTE), lw, fptr);

        for (col = 0, b = buffer; col < lw; col++)
        {
                        *(y++) = *(b++);
        }
    }

    u = cb_buffer;


    for (line = 0; line < ch; line++)
    {
        fread(buffer, sizeof(BYTE), cw, fptr);

        for (col = 0, b = buffer; col < cw; col++)
        {
                        *(u++) = *(b++);
        }
    }

    v = cr_buffer;

    for (line = 0; line < ch; line++)
    {
        fread(buffer, sizeof(BYTE), cw, fptr);

        for (col = 0, b = buffer; col < cw; col++)
        {
                        *(v++) = *(b++);
        }
    }

    fclose(fptr);
    return (0);
}
```

B.2.24 reconstr.c

```
/* File: reconstr.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   reconstr.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

void    FrameConstr(
    BYTE    *predictY,
    BYTE    *predictU,
    BYTE    *predictV,
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    short   MV[],
    BYTE    *current,
    BYTE    *currentU,
    BYTE    *currentV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter)

{
    int     offset, mvx, mvy, index1, index2;
    BYTE    *constY2, *constU2, *constV2;
    BYTE    *frameIndex, *frameIndexU, *frameIndexV;
    short   mv[2];
    BYTE    *currentIndex, *currentIndexU, *currentIndexV;
    int     *diffIndex, *diffIndexU, *diffIndexV;

    short   f0off2, f0off3, f0off4;
```

```
/*      reconstruct MB  */

constY2 = constY;

/*      lumin   */

mvx = MV[0] / 2;
mvy = MV[1] / 2;
offset = (mvy * parameter->luminWidth) + mvx;
frameIndex = (BYTE *) (predictY + offset);

GetLoc(MV, &f0off2, &f0off3, &f0off4, parameter->luminWidth);

currentIndex = current;
diffIndex = diff;

for (index1 = 0; index1 < kMBHeight; index1++,
        constY2 += parameter->luminWidthMB,
            frameIndex += parameter->luminWidthMB)
{
    for (index2 = 0; index2 < kMBWidth; index2++, constY2++,
        frameIndex++, currentIndex++, diffIndex++)
    {
        /*      add 2 for integer rounding      */

        *constY2 = ((int) *frameIndex  +
                (int) *(frameIndex + f0off2) +
                (int) *(frameIndex + f0off3) +
                (int) *(frameIndex + f0off4) + 2) / 4;

        *diffIndex = *currentIndex - *constY2;

    }
}

/*      chrom   */

mv[0] = MV[0] / 2;
mv[1] = MV[1] / 2;
GetLoc(mv, &f0off2, &f0off3, &f0off4, parameter->chromWidth);

mvx = MV[0] / 4;
mvy = MV[1] / 4;
offset = mvy * parameter->chromWidth + mvx;
constU2 = constU;
constV2 = constV;
frameIndexU = (BYTE *) (predictU + offset);
frameIndexV = (BYTE *) (predictV + offset);
currentIndexU = currentU;
currentIndexV = currentV;
diffIndexU = diffU;
diffIndexV = diffV;

for (index1 = 0; index1 < kBHeight; index1++,
        constU2 += parameter->chromWidthB,
        constV2 += parameter->chromWidthB,
                frameIndexU += parameter->chromWidthB,
                frameIndexV += parameter->chromWidthB)
{
    for (index2 = 0; index2 < kBWidth; index2++, constU2++,
        constV2++, frameIndexU++, frameIndexV++,
            currentIndexU++, currentIndexV++,
                diffIndexU++, diffIndexV++)
    {
        /*      add 2 for integer rounding      */

        *constU2 = ((int) *frameIndexU +
                (int) *(frameIndexU + f0off2) +
                (int) *(frameIndexU + f0off3) +
                (int) *(frameIndexU + f0off4) + 2) / 4;
        *constV2 = ((int) *frameIndexV +
                (int) *(frameIndexV + f0off2) +
                (int) *(frameIndexV + f0off3) +
                (int) *(frameIndexV + f0off4) + 2) / 4;
```

```
            *diffIndexU = *currentIndexU - *constU2;
            *diffIndexV = *currentIndexV - *constV2;

        }
    }
}


/* Reconstructed interpolated (bi-directional) macroblock */
void    FrameIConstr(
    BYTE    *predictY,
    BYTE    *predictU,
    BYTE    *predictV,
    BYTE    *predictYB,
    BYTE    *predictUB,
    BYTE    *predictVB,
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    short   MV[],
    short   MVB[],
    BYTE    *current,
    BYTE    *currentU,
    BYTE    *currentV,
    int     *diff,
    int     *diffU,
    int     *diffV,
    tParameter      *parameter)

{
    int     offset, mvx, mvy, index1, index2;
    BYTE    *constY2, *constU2, *constV2;
    BYTE    *frameIndex, *frameIndexU, *frameIndexV;
    BYTE    *frameIndexB, *frameIndexUB, *frameIndexVB;

    BYTE    *currentIndex, *currentIndexU, *currentIndexV;
    int     *diffIndex, *diffIndexU, *diffIndexV;

    short   mv[2];
    short   f0off2, f0off3, f0off4, f0off2B, f0off3B, f0off4B;

    int for_pel, back_pel;

    /*      reconstruct MB  */

    constY2 = constY;
    constU2 = constU;
    constV2 = constV;

    /*      lumin   */

    mvx = MV[0] / 2;
    mvy = MV[1] / 2;
    offset = (mvy * parameter->luminWidth) + mvx;
    frameIndex = (BYTE *) (predictY + offset);

    GetLoc(MV, &f0off2, &f0off3, &f0off4, parameter->luminWidth);

    mvx = MVB[0] / 2;
    mvy = MVB[1] / 2;
    offset = (mvy * parameter->luminWidth) + mvx;
    frameIndexB = (BYTE *) (predictYB + offset);

    GetLoc(MVB, &f0off2B, &f0off3B, &f0off4B, parameter->luminWidth);

    currentIndex = current;
    diffIndex = diff;

    for (index1 = 0; index1 < kMBHeight; index1++, constY2 +=
        parameter->luminWidthMB, frameIndex += parameter->luminWidthMB,
            frameIndexB += parameter->luminWidthMB)
    {
        for (index2 = 0; index2 < kMBWidth; index2++, constY2++,
                frameIndex++, frameIndexB++, currentIndex++,
                diffIndex++)
        {
```

```
            /*      add 2 for integer rounding      */
            for_pel = ((int) *frameIndex +
                   (int) *(frameIndex + f0off2) +
                   (int) *(frameIndex + f0off3) +
                   (int) *(frameIndex + f0off4) + 2) / 4;

            back_pel = ((int) *frameIndexB +
                    (int) *(frameIndexB + f0off2B) +
                    (int) *(frameIndexB + f0off3B) +
                    (int) *(frameIndexB + f0off4B) + 2) / 4;

            /*      add 1 for integer rounding      */
            *constY2 = (for_pel + back_pel + 1) / 2;

            *diffIndex = *currentIndex - *constY2;

        }
    }

    /*      chrom   */

    mv[0] = MV[0] / 2;
    mv[1] = MV[1] / 2;
    GetLoc(mv, &f0off2, &f0off3, &f0off4, parameter->chromWidth);
    mv[0] = MVB[0] / 2;
    mv[1] = MVB[1] / 2;
    GetLoc(mv, &f0off2B, &f0off3B, &f0off4B, parameter->chromWidth);

    mvx = MV[0] / 4;
    mvy = MV[1] / 4;
    offset = mvy * parameter->chromWidth + mvx;
    frameIndexU = (BYTE *) (predictU + offset);
    frameIndexV = (BYTE *) (predictV + offset);
    mvx = MVB[0] / 4;
    mvy = MVB[1] / 4;
    offset = mvy * parameter->chromWidth + mvx;
    frameIndexUB = (BYTE *) (predictUB + offset);
    frameIndexVB = (BYTE *) (predictVB + offset);

    currentIndexU = currentU;
    currentIndexV = currentV;
    diffIndexU = diffU;
    diffIndexV = diffV;

    for (index1 = 0; index1 < kBHeight; index1++, constU2 +=
        parameter->chromWidthB, constV2 += parameter->chromWidthB,
        frameIndexU += parameter->chromWidthB, frameIndexV +=
        parameter->chromWidthB, frameIndexUB += parameter->chromWidthB,
        frameIndexVB += parameter->chromWidthB)
    {
        for (index2 = 0; index2 < kBWidth; index2++, constU2++, constV2++,
                frameIndexU++, frameIndexV++, frameIndexUB++,
                    frameIndexVB++, currentIndexU++,
                    currentIndexV++, diffIndexU++, diffIndexV++)

        {
            /*      add 2 for integer rounding      */
            for_pel = ((int) *frameIndexU +
                   (int) *(frameIndexU + f0off2) +
                   (int) *(frameIndexU + f0off3) +
                   (int) *(frameIndexU + f0off4) + 2) / 4;

            back_pel = ((int) *frameIndexUB +
                    (int) *(frameIndexUB + f0off2B) +
                    (int) *(frameIndexUB + f0off3B) +
                    (int) *(frameIndexUB + f0off4B) + 2) / 4;

            /*      add 1 for integer rounding      */
            *constU2 = (for_pel + back_pel + 1) / 2;

            for_pel = ((int) *frameIndexV +
                   (int) *(frameIndexV + f0off2) +
                   (int) *(frameIndexV + f0off3) +
                   (int) *(frameIndexV + f0off4) + 2) / 4;
```

```
            back_pel = ((int) *frameIndexVB +
                    (int) *(frameIndexVB + f0off2B) +
                    (int) *(frameIndexVB + f0off3B) +
                    (int) *(frameIndexVB + f0off4B) + 2) / 4;

            *constV2 = (for_pel + back_pel + 1) / 2;

            *diffIndexU = *currentIndexU - *constU2;
            *diffIndexV = *currentIndexV - *constV2;

        }
    }
}


/* convert motion vector into 1-D pointer picture buffer co-ordinates */
void    GetLoc(
    short   MV[],
    short   *f0off2,
    short   *f0off3,
    short   *f0off4,
    short   width)
{
    /*      frame   */

    if (MV[0] >= 0)
    {
        if (MV[1] >= 0)
        {
            switch (((MV[0] % 2) << 1) + (MV[1] % 2))
            {
                case 0:         /*      no half pel     */

                    *f0off2 = 0;
                    *f0off3 = 0;
                    *f0off4 = 0;
                    break;

                case 1:         /*      y half pel      */

                    *f0off2 = 0;
                    *f0off3 = width;
                    *f0off4 = *f0off3;
                    break;

                case 2:         /*      x half pel      */

                    *f0off2 = 1;
                    *f0off3 = 0;
                    *f0off4 = *f0off2;
                    break;

                case 3:         /*      x, y half pel   */

                    *f0off2 = 1;
                    *f0off3 = width;
                    *f0off4 = *f0off3 + *f0off2;
                    break;
            }
        }
        else    /*      MV[1] < 0       */
        {
            switch (((MV[0] % 2) << 1) + (-MV[1] % 2))
            {
                case 0:         /*      no half pel     */

                    *f0off2 = 0;
                    *f0off3 = 0;
                    *f0off4 = 0;
                    break;

                case 1:         /*      y half pel      */

                    *f0off2 = 0;
                    *f0off3 = -width;
                    *f0off4 = *f0off3;
```

```
                    break;

                case 2:         /*      x half pel      */

                    *f0off2 = 1;
                    *f0off3 = 0;
                    *f0off4 = *f0off2;
                    break;

                case 3:         /*      x, y half pel   */

                    *f0off2 = 1;
                    *f0off3 = -width;
                    *f0off4 = *f0off3 + *f0off2;
                    break;
            }
        }
    }
    else    /*      MV[0] < 0       */
    {
        if (MV[1] >= 0)
        {
            switch (((-MV[0] % 2) << 1) + (MV[1] % 2))
            {
                case 0:         /*      no half pel     */

                    *f0off2 = 0;
                    *f0off3 = 0;
                    *f0off4 = 0;
                    break;

                case 1:         /*      y half pel      */

                    *f0off2 = 0;
                    *f0off3 = width;
                    *f0off4 = *f0off3;
                    break;

                case 2:         /*      x half pel      */

                    *f0off2 = -1;
                    *f0off3 = 0;
                    *f0off4 = *f0off2;
                    break;

                case 3:         /*      x, y half pel   */

                    *f0off2 = -1;
                    *f0off3 = width;
                    *f0off4 = *f0off3 + *f0off2;
                    break;
            }
        }
        else    /*      MV[1] < 0       */
        {
            switch (((-MV[0] % 2) << 1) + (-MV[1] % 2))
            {
                case 0:         /*      no half pel     */

                    *f0off2 = 0;
                    *f0off3 = 0;
                    *f0off4 = 0;
                    break;

                case 1:         /*      y half pel      */

                    *f0off2 = 0;
                    *f0off3 = -width;
                    *f0off4 = *f0off3;
                    break;

                case 2:         /*      x half pel      */

                    *f0off2 = -1;
                    *f0off3 = 0;
                    *f0off4 = *f0off2;
```

```
                    break;

                case 3:         /*      x, y half pel   */

                    *f0off2 = -1;
                    *f0off3 = -width;
                    *f0off4 = *f0off3 + *f0off2;
                    break;
            }
        }
    }
}
```

B.2.25 stats.c

```
/* File: stats.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   stats.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:56  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

void    statistic(
    int             pictureNum,
    short           pictureType,
    tParameter      *parameter,
    BYTE            *original,
    BYTE            *originalU,
```

```
    BYTE            *originalV,
    BYTE            *reconst,
    BYTE            *reconstU,
    BYTE            *reconstV,
    int             numLuminPixels,
    int             numChromPixels)
{
    FILE            *statFptr, *fopen();
    double          Ysnr, Usnr, Vsnr;

    /*      calculates signal to noise ratio        */

    if ((statFptr = fopen (parameter->statusFName, "a")) == NULL)
    {
        printf ("error opening file '%s', terminating ...\n",
                parameter->statusFName);
        exit(-1);
    }

    snr(reconst, original, &Ysnr, reconstU, originalU, &Usnr,
            reconstV, originalV, &Vsnr , numLuminPixels, numChromPixels);

    fprintf(statFptr, "pictureNum = %d\tpictureType = %s\t", pictureNum,
            (pictureType == kIPicture) ? "I picture" :
            (pictureType == kPPicture) ? "P picture" : "B picture");
    fprintf(statFptr, "\tYsnr = %.2f\tUsnr = %.2f\tVsnr = %.2f\n",
            Ysnr, Usnr, Vsnr);

    fclose(statFptr);
}


void    snr(
    BYTE    *ycod,
    BYTE    *ysrc,
    double  *ysnr,
    BYTE    *ucod,
    BYTE    *usrc,
    double  *usnr,
    BYTE    *vcod,
    BYTE    *vsrc,
    double  *vsnr,
    int     numLuminPixels,
    int     numChromPixels)
{
    int     dif, i;
    double  acc = 0.0;

    for (i = 0; i < numLuminPixels; i++)
    {
        dif = ycod[i] - ysrc[i];
        acc += dif * dif;
    }

    acc /= numLuminPixels;
    *ysnr = 20 * log10(255 / sqrt(acc));

    acc = 0.0;

    for (i = 0; i < numChromPixels; i++)
    {
        dif = ucod[i] - usrc[i];
        acc += dif * dif;
    }

    acc /= numChromPixels;
    *usnr = 20 * log10(255 / sqrt(acc));

    acc = 0.0;

    for (i = 0; i < numChromPixels; i++)
    {
        dif = vcod[i] - vsrc[i];
        acc += dif * dif;
    }
```

```
    acc /= numChromPixels;
    *vsnr = 20 * log10(255 / sqrt(acc));
}

/*
 *      $Log:   stats.c,v $
 * Revision 1.1.1.1  93/03/29  11:27:56  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

void    statistic(
    int             pictureNum,
    short           pictureType,
    tParameter      *parameter,
    BYTE            *original,
    BYTE            *originalU,
    BYTE            *originalV,
    BYTE            *reconst,
    BYTE            *reconstU,
    BYTE            *reconstV,
    int             numLuminPixels,
    int             numChromPixels)
{
    FILE            *statFptr, *fopen();
    double          Ysnr, Usnr, Vsnr;

    /*      calculates signal to noise ratio        */

    if ((statFptr = fopen (parameter->statusFName, "a")) == NULL)
    {
        printf ("error opening file '%s', terminating ...\n",
                parameter->statusFName);
        exit(-1);
    }

    snr(reconst, original, &Ysnr, reconstU, originalU, &Usnr,
            reconstV, originalV, &Vsnr , numLuminPixels, numChromPixels);

    fprintf(statFptr, "pictureNum = %d\tpictureType = %s\t", pictureNum,
            (pictureType == kIPicture) ? "I picture" :
            (pictureType == kPPicture) ? "P picture" : "B picture");
    fprintf(statFptr, "\tYsnr = %.2f\tUsnr = %.2f\tVsnr = %.2f\n",
            Ysnr, Usnr, Vsnr);

    fclose(statFptr);
}


void    snr(
    BYTE    *ycod,
    BYTE    *ysrc,
    double  *ysnr,
    BYTE    *ucod,
    BYTE    *usrc,
    double  *usnr,
    BYTE    *vcod,
    BYTE    *vsrc,
    double  *vsnr,
    int     numLuminPixels,
    int     numChromPixels)
{
    int     dif, i;
    double  acc = 0.0;

    for (i = 0; i < numLuminPixels; i++)
```

```
    {
        dif = ycod[i] - ysrc[i];
        acc += dif * dif;
    }

    acc /= numLuminPixels;
    *ysnr = 20 * log10(255 / sqrt(acc));

    acc = 0.0;

    for (i = 0; i < numChromPixels; i++)
    {
        dif = ucod[i] - usrc[i];
        acc += dif * dif;
    }

    acc /= numChromPixels;
    *usnr = 20 * log10(255 / sqrt(acc));

    acc = 0.0;

    for (i = 0; i < numChromPixels; i++)
    {
        dif = vcod[i] - vsrc[i];
        acc += dif * dif;
    }

    acc /= numChromPixels;
    *vsnr = 20 * log10(255 / sqrt(acc));
}
```

B.2.26 transfer.c

```
/* File: transfer.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   transfer.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:56  oosa
 * MPEG1 encoder/decoder initial revision
 *
```

```
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

/* copy one macroblock from source to destination */
void    CopyToBuf(
    int     bufIndex,
    int     bufIndexC,
    BYTE    bufY[],
    BYTE    bufU[],
    BYTE    bufV[],
    BYTE    *currentMB,
    BYTE    *currentMBU,
    BYTE    *currentMBV,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    tParameter      *parameter)
{
    int     row, col, currentIndex;

    /*      lumin   */

    for (row = 0, currentIndex = 0; row < kMBHeight;
            row++, bufIndex += parameter->luminWidthMB)
    {
        for (col = 0; col < kMBWidth; col++, currentIndex++, bufIndex++)
        {
            currentMB[currentIndex] = bufY[bufIndex];
            currentMBDiff[currentIndex] = bufY[bufIndex];
        }
    }

    /*      chrom   */

    for (row = 0, currentIndex = 0; row < kBHeight;
            row++, bufIndexC += parameter->chromWidthB)
    {
        for (col = 0; col < kBWidth; col++, currentIndex++, bufIndexC++)
        {
            currentMBU[currentIndex] = bufU[bufIndexC];
            currentMBV[currentIndex] = bufV[bufIndexC];
            currentMBDiffU[currentIndex] = bufU[bufIndexC];
            currentMBDiffV[currentIndex] = bufV[bufIndexC];
        }
    }
}


/* copy entire picture */

void    CopyFromBuf(
    short           pictureType,
    int             mbType,
    short           codedBPattern,
    int             mbRow,
    int             mbCol,
    BYTE            *bufYR,
    BYTE            *bufUR,
    BYTE            *bufVR,
    int             currentMBDiff[],
    int             currentMBDiffU[],
    int             currentMBDiffV[],
    int             mbIndex,
    tParameter      *parameter)
{
    int             bufIndex, bufIndex1, currentIndex, temp;
    short           code, line, pix, row, col;

    if (mbType & kIntraType)        /*      intra   */
```

```
{
    bufIndex = mbRow * kMBHeight * parameter->luminWidth + mbCol * kMBWidth;

    /*      lumin   */

    for (row = 0, currentIndex = 0; row < kMBHeight; row++,
            bufIndex += parameter->luminWidthMB)
    {
        for (col = 0; col < kMBWidth; col++, currentIndex++, bufIndex++)
        {
            bufYR[bufIndex] = (currentMBDiff[currentIndex] >= 255) ? 255 :
                    ((currentMBDiff[currentIndex] <= 0) ? 0 :
                    currentMBDiff[currentIndex]);
        }
    }

    /*      chrom   */

    bufIndex = mbRow * kBHeight * parameter->chromWidth + mbCol * kBWidth;

    for (row = 0, currentIndex = 0; row < kBHeight; row++,
            bufIndex += parameter->chromWidthB)
    {
        for (col = 0; col < kBWidth; col++, currentIndex++, bufIndex++)
        {
            bufUR[bufIndex] = (currentMBDiffU[currentIndex] >= 255) ? 255 :
                    ((currentMBDiffU[currentIndex] <= 0) ? 0 :
                    currentMBDiffU[currentIndex]);
            bufVR[bufIndex] = (currentMBDiffV[currentIndex] >= 255) ? 255 :
                    ((currentMBDiffV[currentIndex] <= 0) ? 0 :
                    currentMBDiffV[currentIndex]);
        }
    }
}
else    /*      inter   */
{
    /*      lumin   */

    code = 0x20;

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        bufIndex1 = (mbRow * kMBHeight + line) * parameter->luminWidth +
                mbCol * kMBWidth;

        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            if (! (codedBPattern & code))
            {
                code >>= 1;
                continue;
            }

            bufIndex = bufIndex1 + pix;

            code >>= 1;

            currentIndex = line * kMBWidth + pix;

            for (row = 0; row < kDCTSize;
                    row++, currentIndex += kMBWidthDCT,
                    bufIndex += parameter->luminWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++, currentIndex++,
                        bufIndex++)
                {
                    temp = bufYR[bufIndex] + currentMBDiff[currentIndex];
                    bufYR[bufIndex] =
                            (temp >= 255) ? 255 : ((temp <= 0) ? 0 : temp);
                }
            }
        }
    }

    /*      chrom - U       */
```

```
        if (codedBPattern & 0x02)
        {
            bufIndex = mbRow * kBHeight * parameter->chromWidth + mbCol * kBWidth;
/* ? */
            for (row = 0, currentIndex = 0; row < kBHeight; row++,
                    bufIndex += parameter->chromWidthDCT)
            {
                for (col = 0; col < kBWidth; col++, currentIndex++, bufIndex++)
                {
                    temp = bufUR[bufIndex] + currentMBDiffU[currentIndex];
                    bufUR[bufIndex] =
                            (temp >= 255) ? 255 : ((temp <= 0) ? 0 : temp);
                }
            }
        }

        /*      chrom - V       */

        if (codedBPattern & 0x01)
        {
            bufIndex = mbRow * kBHeight * parameter->chromWidth + mbCol * kBWidth;

            for (row = 0, currentIndex = 0; row < kBHeight; row++,
                    bufIndex += parameter->chromWidthDCT)
            {
                for (col = 0; col < kBWidth; col++, currentIndex++, bufIndex++)
                {
                    temp = bufVR[bufIndex] + currentMBDiffV[currentIndex];
                    bufVR[bufIndex] =
                            (temp >= 255) ? 255 : ((temp <= 0) ? 0 : temp);
                }
            }
        }
    }
}
```

B.2.27 writebit.c

```
/* File: writebit.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   writebit.c,v $
```

```
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:28:01  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:30:43  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"

#define kBufSize        1024
#define kIntNumBits     32

FILE    *gBitStream;

void    writeBits(
    unsigned int    bits,
    short           numBits,
    short           startCode)
{

    extern FILE             *gBitStream;

    static unsigned int     buf[kBufSize] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    static short            wdPtr = 0, wdIndex = 0;

    short                   srcStart, index;
    unsigned int            tempBits;
    int                      i;
    unsigned int         temp;

    if (startCode == -1)    /*      flush buffer    */
    {
        for (i = 0; i < wdIndex + 1; i++)
        {
            temp = buf[i];
            putc(temp>>24,gBitStream);
            putc(temp>>16,gBitStream);
            putc(temp>>8,gBitStream);
            putc(temp,gBitStream);
        }
        /*fwrite(buf, sizeof (unsigned int), wdIndex + 1, gBitStream);*/
        fclose(gBitStream);
        return;
    }

    if (! numBits)
        return;

    srcStart = kIntNumBits - numBits;

    if (startCode == 1)             /*      check if byte align             */
    {
        while (wdPtr % 8)
        {
            /*      do byte alignment       */

            wdPtr++;

            if (wdPtr == kIntNumBits)
            {
                wdIndex++;
                wdPtr = 0;

                if (wdIndex == kBufSize)
                {
                    for (i = 0; i < kBufSize; i++)
                    {
                        temp = buf[i];
                        putc(temp>>24,gBitStream);
```

```
                    putc(temp>>16,gBitStream);
                    putc(temp>>8,gBitStream);
                    putc(temp,gBitStream);
                }
                /*fwrite(buf, sizeof (unsigned int), kBufSize, gBitStream);*/
                wdIndex = 0;

                for (index = 0; index < kBufSize; index++)
                    buf[index] = 0;         /*      clear buffer    */
            }
        }
    }
}

if (srcStart > wdPtr)
{
    tempBits = bits << (srcStart - wdPtr);

    buf[wdIndex] |= tempBits;
    wdPtr += numBits;
}
else if (srcStart == wdPtr)
{
    buf[wdIndex] |= bits;
    wdPtr = 0;
    wdIndex++;

    if (wdIndex == kBufSize)
    {
        for (i = 0; i < kBufSize; i++)
        {
            temp = buf[i];
            putc(temp>>24,gBitStream);
            putc(temp>>16,gBitStream);
            putc(temp>>8,gBitStream);
            putc(temp,gBitStream);
        }
        /*fwrite(buf, sizeof (unsigned int), kBufSize, gBitStream);*/
        wdIndex = 0;

        for (index = 0; index < kBufSize; index++)
            buf[index] = 0;         /*      clear buffer    */
    }
}
else    /*      does not fit in one location    */
{
    tempBits = bits >> (wdPtr - srcStart);
    buf[wdIndex] |= tempBits;
    numBits = wdPtr - srcStart;             /*      new numBits     */
    wdPtr = 0;
    wdIndex++;

    if (wdIndex == kBufSize)
    {
        for (i = 0; i < kBufSize; i++)
        {
            temp = buf[i];
            putc(temp>>24,gBitStream);
            putc(temp>>16,gBitStream);
            putc(temp>>8,gBitStream);
            putc(temp,gBitStream);
        }
        /*fwrite(buf, sizeof (unsigned int), kBufSize, gBitStream);*/
        wdIndex = 0;

        for (index = 0; index < kBufSize; index++)
            buf[index] = 0;         /*      clear buffer    */
    }

    tempBits = bits << (kIntNumBits - numBits);
    buf[wdIndex] |= tempBits;
    wdPtr += numBits;
}
}
```

B.2.28 writepic.c

```
/* File: WritePic.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   writepic.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "consts.h"
#include "mpeg1.h"
#include "encoder.pro"


short   WritePicture(
    int             pictureNum,    /* Needed for file extension */
    short           pictureType,   /* used only for printed statistics */
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    BYTE            *y_orig,
    BYTE            *cb_orig,
    BYTE            *cr_orig,
    tParameter      *parameter)
{

    int ret;

    statistic(pictureNum, pictureType, parameter, y_orig, cb_orig, cr_orig,
        y_recon, cb_recon, cr_recon,parameter->numLuminPixels,
        parameter->numChromPixels);
```

```
    switch (parameter->reconFileFormat)
    {
    case 0:
        ret = Write_SIF(pictureNum, y_recon, cb_recon, cr_recon, parameter);
        break;
    case 1:
        ret = Write_TGA(pictureNum, y_recon, cb_recon, cr_recon, parameter);
        break;
    case 2:
        ret = Write_PPM(pictureNum, y_recon, cb_recon, cr_recon, parameter);
        break;
    case 3:
        ret = Write_YUV(pictureNum, y_recon, cb_recon, cr_recon, parameter);
        break;

    }

    return(ret);
}


int     Write_SIF(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter)
{
    register        i,j;

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *buffer;
    BYTE    *b;
    BYTE    *y;
    BYTE    *u;
    BYTE    *v;
    BYTE    *tmpuR;
    BYTE    *tmpvR;
    int     numChromPixels, luminWidthHalf, luminWidth2;

    numChromPixels = (8 * parameter->mb_width) * (8 * parameter->mb_height);
    luminWidthHalf = parameter->horizontal_size / 2;
    luminWidth2 = parameter->horizontal_size * 2;
    tmpuR = (BYTE *) malloc(numChromPixels * sizeof(BYTE));
    tmpvR = (BYTE *) malloc(numChromPixels * sizeof(BYTE));
    buffer = (BYTE *) malloc(luminWidth2 * sizeof(BYTE));

    y = y_recon;
    u = cb_recon;
    v = cr_recon;

    for (i = 0; i < numChromPixels; i++)
    {
        tmpuR[i] = u[i];
        tmpvR[i] = v[i];
    }

    /* interpolate chrominance in Y direction */

    convert420to422(cb_recon, 8 * parameter->mb_width, 16* parameter->mb_height);
    convert420to422(cr_recon, 8 * parameter->mb_width, 16* parameter->mb_height);

    sprintf (fname, "%s%03d.SIF", parameter->outputSequence, pictureNum);

    printf("%s\n",fname);

    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (1);
    }

    /* store picture */

    y = y_recon;
```

```
    u = cb_recon;
    v = cr_recon;

    for (i = 0; i < parameter->vertical_size; i++)
    {
        for (j = 0, b = buffer; j < luminWidthHalf; j++)
        {
            *(b++) = *(u++);
            *(b++) = *(y++);
            *(b++) = *(v++);
            *(b++) = *(y++);
        }

        fwrite(buffer, sizeof(BYTE), luminWidth2, fptr);
        y += 16 * parameter->mb_width - 2 * luminWidthHalf;
        u += 8 * parameter->mb_width - luminWidthHalf;
        v += 8 * parameter->mb_width - luminWidthHalf;
    }

    fclose(fptr);

    u = cb_recon;
    v = cr_recon;

    for (i = 0; i < numChromPixels; i++)
    {
        u[i] = tmpuR[i];
        v[i] = tmpvR[i];
    }

    free(tmpuR);
    free(tmpvR);
    free(buffer);

    return (0);
}

int     Write_YUV(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter)
{
    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *buffer;
    BYTE    *b;
    BYTE    *y;
    BYTE    *u;
    BYTE    *v;
    register line, col;
    int     lw, lh, cw, ch;

    lw = parameter->horizontal_size;
    lh = parameter->vertical_size;
    cw = lw >> 1;   /* chroma width is always half luma for 4:2:0 */
    ch = lh >> 1;   /* ditto for height */

    /* allocate "line buffer" of greatest common multiple dimensions */
    buffer = (BYTE *) malloc(lw * sizeof(BYTE));

    sprintf (fname, "%s%03d.yuv", parameter->outputSequence, pictureNum);
    printf("%s\n",fname);

    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (-1);
    }

    /* store picture */

    y = y_recon;

    for (line = 0; line <  lh; line++)
```

```
    {
        for (col = 0, b = buffer; col < lw; col++)
        {
            *(b++) = *(y++);
        }
        fwrite(buffer, sizeof(BYTE), lw, fptr);
        y += (16 * parameter->mb_width) - lw;

    }

    u = cb_recon;

    for (line = 0; line < ch; line++)
    {
        for (col = 0, b = buffer; col < cw; col++)
        {
            *(b++) = *(u++);
        }

        fwrite(buffer, sizeof(BYTE), cw, fptr);
        u += (8 * parameter->mb_width) - cw;

    }


    v = cr_recon;

    for (line = 0; line < ch; line++)
    {
        for (col = 0, b = buffer; col < cw; col++)
        {
            *(b++) = *(v++);
        }

        fwrite(buffer, sizeof(BYTE), cw, fptr);
        v += (8 * parameter->mb_width) - cw;

    }

    free(buffer);
    fclose(fptr);
    return (0);
}

int     Write_TGA(
    int             aPictureNum,
    BYTE            *y,
    BYTE            *u,
    BYTE            *v,
    tParameter      *parameter)
{
    int             i, j;
    static unsigned char    tga24[14]={0,0,2,0,0,0,0, 0,0,0,0,0,24,32};

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *u2;
    BYTE    *v2;
    int             r, g, b;

        int mb_width = parameter->mb_width;
        int mb_height = parameter->mb_height;
        int vertical_size = parameter->vertical_size;
        int horizontal_size = parameter->horizontal_size;

    u2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));
    v2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));

    /* interpolate chrominance in both directions */

    convert420to444(u, u2, 8 * mb_width, 8 * mb_height);
    convert420to444(v, v2, 8 * mb_width, 8 * mb_height);

    sprintf (fname, "%s%03d.tga", parameter->outputSequence, aPictureNum);

    printf("%s\n",fname);
```

```
    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (1);
    }

    /* write Targa header */
    for (i = 0; i < 12; i++)
        putc(tga24[i], fptr);

    putc(horizontal_size, fptr);
    putc(horizontal_size >> 8, fptr);
    putc(vertical_size, fptr);
    putc(vertical_size >> 8,fptr);
    putc(tga24[12], fptr);
    putc(tga24[13], fptr);

    /* store picture */

    for (j = 0; j < vertical_size; j++)
    {
        for (i = 0; i < horizontal_size; i++)
        {

            yuvtorgb(*y++, *u2++, *v2++, &r, &g, &b);

            putc(b, fptr);
            putc(g, fptr);
            putc(r, fptr);
        }

        y  += 16 * mb_width - horizontal_size;
        u2 += 16 * mb_width - horizontal_size;
        v2 += 16 * mb_width - horizontal_size;
    }

    fclose(fptr);

    free(u2);
    free(v2);

    return (0);
}

int     Write_PPM(
    int                     aPictureNum,
    BYTE            *y,
    BYTE            *u,
    BYTE            *v,
    tParameter      *parameter)
{
    int             i, j;

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *u2;
    BYTE    *v2;
    int             r, g, b;

        int mb_width = parameter->mb_width;
        int mb_height = parameter->mb_height;
        int vertical_size = parameter->vertical_size;
        int horizontal_size = parameter->horizontal_size;

    u2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));
    v2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));

    /* interpolate chrominance in both directions */

    convert420to444(u, u2, 8 * mb_width, 8 * mb_height);
    convert420to444(v, v2, 8 * mb_width, 8 * mb_height);

    sprintf (fname, "%s%03d.ppm", parameter->outputSequence, aPictureNum);

    printf("%s\n",fname);
```

```
    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (1);
    }

    /* write PPM header */
    fprintf(fptr,"P6\n%d %d\n255\n",horizontal_size,vertical_size);

    /* store picture */

    for (j = 0; j < vertical_size; j++)
    {
        for (i = 0; i < horizontal_size; i++)
        {

            yuvtorgb(*y++, *u2++, *v2++, &r, &g, &b);

            putc(r, fptr);
            putc(g, fptr);
            putc(b, fptr);
        }

        y  += 16 * mb_width - horizontal_size;
        u2 += 16 * mb_width - horizontal_size;
        v2 += 16 * mb_width - horizontal_size;
    }

    fclose(fptr);

    free(u2);
    free(v2);

    return (0);
}

void    yuvtorgb(
    int             y,
    int             u,
    int             v,
    int             *pr,
    int             *pg,
    int             *pb)
{
    int             r, g, b;
    double  yf, uf, vf, rmy, gmy, bmy, rf, gf, bf;

    /* scale to 0..1 (Y) / -1..+1 (U,V) */

    yf = (y - 16) / (double)(235-16);
    uf = (u - 128) / (double)(240-16);
    vf = (v - 128) / (double)(240-16);

    /* compute color differences R-Y, G-Y, B-Y */

    rmy = ((1.0-0.299)/0.5) * vf; /* 1.402*vf */
    bmy = ((1.0-0.114)/0.5) * uf; /* 1.772*uf */
    gmy = -(0.299*rmy + 0.114*bmy)/0.587;

    /* compute color primaries R, G, B */

    rf = yf + rmy;
    bf = yf + bmy;
    gf = yf + gmy;

    /* scale to 0..255 */

    r = floor(256.0 * rf + 0.5);
    if (r < 0)
        r = 0;
    if (r > 255)
        r = 255;
    g = floor(256.0 * gf + 0.5);
    if (g < 0)
        g = 0;
```

```
    if (g > 255)
        g = 255;
    b = floor(256.0 * bf + 0.5);
    if (b < 0)
        b = 0;
    if (b > 255)
        b = 255;

    *pr = r;
    *pg = g;
    *pb = b;

}
```

B.3 Decoder

B.3.1 constr.c

```
/* File:  constr.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).

 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   constr.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
```

```
#include "types.h"
#include "decode.h"
#include "decode.pro"

/* construction backwards or forwards macroblock prediction */
void    Construct_Mono_Pred(
    BYTE    *predictY,
    BYTE    *predictU,
    BYTE    *predictV,
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    short   MV[]
    )

{
    int     offset, mvx, mvy, index1, index2;
    BYTE    *constY2, *constU2, *constV2;
    BYTE    *frameIndex, *frameIndexU, *frameIndexV;
    int     mv[2];
    int     luminWidth, chromWidth;
    int     f0off2, f0off3, f0off4;

    /*      reconstruct MB  */
    luminWidth = 16 * mb_width;
    chromWidth = 8 * mb_width;

    constY2 = constY;

    /*      lumin   */

    mvx = MV[0] >> 1;
    mvy = MV[1] >> 1;
    offset = (mvy * luminWidth) + mvx;
    f0off2 = MV[0] & 1;
    f0off3 = (MV[1] & 1) * luminWidth;
    f0off4 = f0off2 + f0off3;
    frameIndex = predictY + offset;

    for (index1 = 0; index1 < kMBHeight; index1++,
            constY2 += luminWidth - kMBWidth,
            frameIndex += luminWidth - kMBWidth)
    {
        for (index2 = 0; index2 < kMBWidth; index2++,
                constY2++, frameIndex++)
        {
            /*      add 2 for integer rounding (always non-negative)        */

            *constY2 = ((int) *frameIndex  +
                    (int) *(frameIndex + f0off2) +
                    (int) *(frameIndex + f0off3) +
                    (int) *(frameIndex + f0off4) + 2) / 4;

        }
    }

    /*      chrom   */

    mv[0] = MV[0] / 2;
    mv[1] = MV[1] / 2;
    f0off2 = mv[0] & 1;
    f0off3 = (mv[1] & 1) * chromWidth;
    f0off4 = f0off2 + f0off3;
    mvx = mv[0] >> 1;
    mvy = mv[1] >> 1;
    offset = (mvy * chromWidth) + mvx;
    frameIndexU = predictU + offset;
    frameIndexV = predictV + offset;
    constU2 = constU;
    constV2 = constV;

    for (index1 = 0; index1 < kBHeight; index1++,
            constU2 += chromWidth - kBWidth, constV2 += chromWidth - kBWidth,
            frameIndexU += chromWidth - kBWidth,
            frameIndexV += chromWidth - kBWidth)
    {
```

```
        for (index2 = 0; index2 < kBWidth; index2++,
                constU2++, constV2++, frameIndexU++, frameIndexV++)
        {
            *constU2 = ((int) *frameIndexU +
                    (int) *(frameIndexU + f0off2) +
                    (int) *(frameIndexU + f0off3) +
                    (int) *(frameIndexU + f0off4) + 2) / 4;

            *constV2 = ((int) *frameIndexV +
                    (int) *(frameIndexV + f0off2) +
                    (int) *(frameIndexV + f0off3) +
                    (int) *(frameIndexV + f0off4) + 2) / 4;
        }
    }
}


/* Construct bi-directional/interpolated prediction */
void    Construct_Bi_Pred(
    BYTE    *predictY,
    BYTE    *predictU,
    BYTE    *predictV,
    BYTE    *predictYB,
    BYTE    *predictUB,
    BYTE    *predictVB,
    BYTE    *constY,
    BYTE    *constU,
    BYTE    *constV,
    short   MV[],
    short   MVB[]
    )

{
    int             offset, mvx, mvy, index1, index2;
    BYTE    *constY2, *constU2, *constV2;
    BYTE    *frameIndex, *frameIndexU, *frameIndexV;
    BYTE    *frameIndexB, *frameIndexUB, *frameIndexVB;
    int             mv[2];
    int             f0off2, f0off3, f0off4, f0off2B, f0off3B, f0off4B;
    int             pel_for, pel_back;
    int             luminWidth, luminWidthMB, chromWidth, chromWidthB;

    /*      reconstruct MB  */
    luminWidth = 16 * mb_width;
    luminWidthMB = luminWidth - kMBWidth;
    chromWidth = 8 * mb_width;
    chromWidthB = chromWidth - kBWidth;

    /*      lumin   */

    mvx = MV[0] >> 1;
    mvy = MV[1] >> 1;
    offset = (mvy * luminWidth) + mvx;
    f0off2 = MV[0] & 1;
    f0off3 = (MV[1] & 1) * luminWidth;
    f0off4 = f0off2 + f0off3;
    frameIndex = predictY + offset;

    mvx = MVB[0] >> 1;
    mvy = MVB[1] >> 1;
    offset = (mvy * luminWidth) + mvx;
    f0off2B = MVB[0] & 1;
    f0off3B = (MVB[1] & 1) * luminWidth;
    f0off4B = f0off2B + f0off3B;
    frameIndexB = predictYB + offset;

    constY2 = constY;

    for (index1 = 0; index1 < kMBHeight; index1++,
            constY2 += luminWidth - kMBWidth,
            frameIndex += luminWidth - kMBWidth,
            frameIndexB += luminWidth - kMBWidth)
    {
        for (index2 = 0; index2 < kMBWidth; index2++,
            constY2++, frameIndex++, frameIndexB++)
        {
```

```
        /*      add 2 for integer rounding (always non-negative)        */

        pel_for = ((int) *frameIndex +
                (int) *(frameIndex + f0off2) +
                (int) *(frameIndex + f0off3) +
                (int) *(frameIndex + f0off4) + 2) / 4;

        pel_back = ((int) *frameIndexB +
                (int) *(frameIndexB + f0off2B) +
                (int) *(frameIndexB + f0off3B) +
                (int) *(frameIndexB + f0off4B) + 2) / 4;

        /*      add 1 for integer rounding (always non-negative)        */

        *constY2 = (pel_for + pel_back + 1) / 2;
    }
}

/*      chrom   */

mv[0] = MV[0] / 2;
mv[1] = MV[1] / 2;
f0off2 = mv[0] & 1;
f0off3 = (mv[1] & 1) * chromWidth;
f0off4 = f0off2 + f0off3;
mvx = mv[0] >> 1;
mvy = mv[1] >> 1;
offset = (mvy * chromWidth) + mvx;
frameIndexU = predictU + offset;
frameIndexV = predictV + offset;

mv[0] = MVB[0] / 2;
mv[1] = MVB[1] / 2;
f0off2B = mv[0] & 1;
f0off3B = (mv[1] & 1) * chromWidth;
f0off4B = f0off2B + f0off3B;
mvx = mv[0] >> 1;
mvy = mv[1] >> 1;
offset = (mvy * chromWidth) + mvx;
frameIndexUB = predictUB + offset;
frameIndexVB = predictVB + offset;

constU2 = constU;
constV2 = constV;

for (index1 = 0; index1 < kBHeight; index1++,
        constU2 += chromWidth - kBWidth,
        constV2 += chromWidth - kBWidth,
        frameIndexU += chromWidth - kBWidth,
        frameIndexV += chromWidth - kBWidth,
        frameIndexUB += chromWidth - kBWidth,
        frameIndexVB += chromWidth - kBWidth)
{
    for (index2 = 0; index2 < kBWidth; index2++, constU2++, constV2++,
            frameIndexU++, frameIndexV++, frameIndexUB++, frameIndexVB++)
    {
        pel_for = ((int) *frameIndexU +
                (int) *(frameIndexU + f0off2) +
                (int) *(frameIndexU + f0off3) +
                (int) *(frameIndexU + f0off4) + 2) / 4;

        pel_back = ((int) *frameIndexUB +
                (int) *(frameIndexUB + f0off2B) +
                (int) *(frameIndexUB + f0off3B) +
                (int) *(frameIndexUB + f0off4B) + 2) / 4;

        *constU2 = (pel_for + pel_back + 1) / 2;

        pel_for = ((int) *frameIndexV +
                (int) *(frameIndexV + f0off2) +
                (int) *(frameIndexV + f0off3) +
                (int) *(frameIndexV + f0off4) + 2) / 4;

        pel_back = ((int) *frameIndexVB +
                (int) *(frameIndexVB + f0off2B) +
                (int) *(frameIndexVB + f0off3B) +
```

```
                    (int) *(frameIndexVB + f0off4B) + 2) / 4;

            *constV2 = (pel_for + pel_back + 1) / 2;
        }
    }
}
```

B.3.2 consts.h

```
/* consts.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *  $Log:   consts.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

#define kMaxCharBufLen   200

#define FALSE 0
#define TRUE  1

#define BYTE         unsigned char
#define RND(i,x)     i = ((x)>0.0 ? floor((x)+0.5) : ceil((x)-0.5))
#define TRUNC(i,x)   i = ((x)>0.0 ? floor((x)) : ceil((x))

#define kMBWidth             16      /*  width of Macro Block in pel              */
#define kMBHeight                16      /*  height of Macro Block in pel */

#define kIPicture    1           /*  I picture type        */
#define kPPicture    2           /*  P picture type        */
#define kBPicture    3           /*  B picture type */
```

```
#define kMBLen       (kMBWidth * kMBHeight)         /*     # of pels in MB
    */
#define kDCTSize 8           /*  width and height of DCT Block in pel      */
#define kDCTLen      (kDCTSize * kDCTSize)          /*     # of pels in DCT Block */
#define kBWidth      8           /*  width of Chrom Block in pel                          */
#define kBHeight 8           /*  height of Chrom Block in pel                     */
#define kBLen        (kBWidth * kBHeight)     /*     # of pels in block            */
#define kMBWidthDCT          (kMBWidth - kDCTSize)

/* macroblock type fields */
#define kMbIntra     0x0001  /*  macroblock_intra                                     */
#define kMbPattern       0x0002  /*  macroblock_pattern
    */
#define kMbBackward      0x0004  /*  macroblock_motion_backward                       */
#define kMbForward       0x0008  /*  macroblock_motion_forward                        */
#define kMbQuant     0x0010  /*  macroblock_quant                                     */
```

B.3.3 convert.c

```
/* file: convert.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   convert.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:55  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <malloc.h>
#include "consts.h"
#include "types.h"
#include "decode.pro"
```

```
void    convert420to422(
    BYTE    *chromPicture,
    int             chromWidth,
    int             chromHeight)
{
    int             i;
    int             j;
    int             *a;
    int             *b;

    a = (int *) malloc(chromHeight * sizeof(int));
    b = (int *) malloc(chromHeight * sizeof(int));

    for(j = 0; j < chromWidth; j++)
    {
        for(i = 0; i < chromHeight / 2; i++)
            *(a + i) = *(chromPicture + i * chromWidth + j);

        filter(chromHeight / 2, a, b);

        for(i = 0; i < chromHeight; i++)
            *(chromPicture + i * chromWidth + j) = *(b + i);
    }

    free(a);
    free(b);
}

void    filter(
    int     n,
    int     *F,
    int     *G)
{
    int i, k, M;
    int     tmp;


    M = 2 * n;

    F[n] = F[n - 1];  /* for 'interpolation' of last point */

    for(i = 0; i < n; i++)
    {
        k = 2 * i;
        G[k] = F[i];

        tmp = F[i] + F[i + 1];

        /* normalize and round */
        G[k + 1] = (tmp >= 0) ? ((tmp + 1) / 2) : -((-tmp + 1) / 2);
    }

    return;
}

void convert420to444(
    BYTE    *chrom420,
    BYTE    *chrom444,
    int             chromWidth,
    int             chromHeight)
{
    int             chromWidth2, i, j;
    BYTE    *src, *dst;

    /* bilinear interpolation filter */

    chromWidth2 = 2 * chromWidth;

    /* horizontal direction */

    for (j = 0; j < chromHeight; j++)
    {
        src = chrom420 + j * chromWidth;
        dst = chrom444 + j * chromWidth2;
        dst[0] = src[0];
        for (i = 1; i < chromWidth; i++)
```

```
        {
            dst[2*i-1] = (3 * src[i-1] + src[i] + 2) >> 2;
            dst[2*i]   = (src[i-1] + 3 * src[i] + 2) >> 2;
        }
        dst[chromWidth2-1] = src[chromWidth-1];
    }

    /* vertical direction, (in-place, bottom-up) */

    for (i = 0; i < chromWidth2; i++)
    {
        src = chrom444 + i + (chromHeight-2) * chromWidth2;
        dst = chrom444 + i + (2*(chromHeight-2)+1) * chromWidth2;
        dst[2*chromWidth2] = src[chromWidth2];
        for (j = chromHeight - 2; j >= 0; j--)
        {
            dst[chromWidth2] = (src[0] + 3 * src[chromWidth2] + 2) >> 2;
            dst[0]           = (3 * src[0] + src[chromWidth2] + 2) >> 2;
            dst -= 2 * chromWidth2;
            src -= chromWidth2;
        }
    }
}
```

B.3.4 decode.c

```
/* file: decode.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   decoder.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.2  93/06/15  15:42:23  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
 *
 */

#include <stdio.h>
```

```
#include "consts.h"
#include "types.h"
#include "decode.pro"

#define GLOBAL
#include "decode.h"

int     main (
    int     argc,
    char    *argv[])
{
    tData                   data;
    tParameter              parameter;

    /* initialize decoder */

    Initial(&parameter, (argc > 1) ? argv[1] : "decode.ini");

    /*      process sequence        */

    Process_Sequence(&parameter, &data);

    return(0);
}
```

B.3.5 decode.h

```
/* File:   decode.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   decoder.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 */

#ifdef GLOBAL
#define EXTERN
#else
#define EXTERN extern
```

```
#endif

EXTERN int horizontal_size, vertical_size, mb_width, mb_height;
EXTERN FILE *TraceFile;
EXTERN int TraceLevel;
```

B.3.6 decode.ini

```
test.mpg      /* name of file containing coded bit stream */
rec           /* output path and file prefix*/
3             /* output format: 0=SIF, 1=TGA, 2=PPM, 3=YUV */
-             /* name of file containing trace output, -=stdout */
2             /* trace level (0: no tracing) */
```

B.3.7 decode.pro

```
/* File: decode.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

void Construct_Mono_Pred(
    BYTE *predictY,
    BYTE *predictU,
    BYTE *predictV,
    BYTE *constY,
    BYTE *constU,
    BYTE *constV,
    short   MV[]
    );

void Construct_Bi_Pred(
    BYTE *predictY,
    BYTE *predictU,
    BYTE *predictV,
    BYTE *predictYB,
    BYTE *predictUB,
    BYTE *predictVB,
    BYTE *constY,
    BYTE *constU,
    BYTE *constV,
    short   MV[],
    short   MVB[]
    );

int Get_Bits(
```

```
    unsigned int *destBuf,
    int             numBits);

int Next_Bits(
    unsigned int *destBuf,
    int             numBits);

int Next_Start_Code(void);

int Get(
    int numBits);

int Get_AC_Code(
    BYTE        *bitCount,
    BYTE        *bitPattern,
    int                     tblLen,
    int                    *signBit,
    int             *run,
    int             *level,
    int             first);

int Get_Code(
    BYTE        *bitCount,
    BYTE        *bitPattern,
    int             tblLen);

int Find_Code(
    BYTE        *bitCount,
    BYTE        *bitPattern,
    int             tblLen,
    unsigned int buf,
    int             numBits);

int Get_Coefs(
    int     *currentMBDiff,
    int     currentIndex,
    int     offset);

void Perform_IDCT(
    int     mbType,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV);

int Write_Picture(
    int         aPictureNum,
    BYTE    *y,
    BYTE    *u,
    BYTE    *v,
    tParameter   *aParameter);

int Write_SIF(
    int         aPictureNum,
    BYTE    *luminPictureR,
    BYTE    *chromUPictureR,
    BYTE    *chromVPictureR,
    tParameter   *aParameter);

int Write_TGA(
    int         aPictureNum,
    BYTE    *y,
    BYTE    *u,
    BYTE    *v,
    tParameter   *aParameter);

int Write_PPM(
    int         aPictureNum,
    BYTE    *y,
    BYTE    *u,
    BYTE    *v,
    tParameter   *aParameter);

void yuvtorgb(
    int     y,
    int     u,
    int     v,
```

```
    int     *pr,
    int     *pg,
    int     *pb);

void convert420to422(
    BYTE *chromPicture,
    int     chromWidth,
    int     chromHeight);

void filter(
    int n,
    int *F,
    int *G);

void convert420to444(
    BYTE *chrom420,
    BYTE *chrom444,
    int     chromWidth,
    int     chromHeight);

void Init_DCT(void);

void idct8x8(
    float   dct_img[],
    float   recon_img[]);

void Initial(
    tParameter      *aParameter,
    char        *parameterFName);

void Init_Data(
    tData           *aData);

BYTE *Barray(
    int size);

short *Sarray(
    int size);

void Inverse_Quantize_Intra_MB(
    short   codedBPattern,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     MQuant);

void Inverse_Quantize_Non_Intra_MB(
    short   codedBPattern,
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     MQuant);

int Process_MB(
    int     *currentMBDiff,
    int     *currentMBDiffU,
    int     *currentMBDiffV,
    int     *DCPredictionY,
    int     *DCPredictionU,
    int     *DCPredictionV,
    int     mbType,
    int     codedBPattern);

int Process_Block(
    int     i,
    int     patternCode,
    int     mbIntra,
    int     *DCPrediction,
    int     *dctRecon);

int Process_Picture(
    int         pictureNum,
    tParameter  *parameter,
    tData       *data);

int Process_Sequence(
```

```
    tParameter   *parameter,
    tData        *data);

voidProcess_Sequence_Header(void);

voidExtension_And_User_Data();

int Process_Slice(
    short        pictureType,
    BYTE    *bufY,
    BYTE    *bufU,
    BYTE    *bufV,
    tData        *data,
    int          forward_f_code,
    int          fullPelForwardV,
    int          backward_f_code,
    int          fullPelBackwardV);

int Get_MBA_Inc(void);

int Get_MV(
    int     f_code,
    int     fullPelV,
    short   MV[2],
    short   PMV[]);

voidCopy_From_Buf(
    int          mbType,
    short        codedBPattern,
    int          mbRow,
    int          mbCol,
    BYTE    *bufYR,
    BYTE    *bufUR,
    BYTE    *bufVR,
    int          currentMBDiff[],
    int          currentMBDiffU[],
    int          currentMBDiffV[]);
```

B.3.8 flc.h

```
/* flc.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
 * Disclaimer of Warranty
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *  $Log:   flc.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
```

```
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.3      93/08/20  21:15:53  hanamura
 * MPEG1 encoder minor revision
 *
 * Revision 1.2  93/06/15  14:59:40  oosa
 * *** empty log message ***
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

/* fixed length codes and field widths */

#define bStartCode                          32                          /* bslbf       */

/*  sequence flc */

#define cSequenceHeaderCode                 0x000001B3         /*      bslbf   */
#define cSequenceEndCode                0x000001B7             /*      bslbf   */

#define bHorizontalSize                     12                          /*      uimsbf  */
#define bVerticalSize                       12                          /*      uimsbf  */
#define bPelAspectRatio                     4                           /*      uimsbf  */
#define bPictureRate                    4                      /*      uimsbf  */
#define bBitRate                        18                     /*      uimsbf  */
#define bMarkerBit                          1                           /*      "1"             */
#define bVbvBufferSize                      10                          /*      uimsbf  */
#define bConstrainedParameterFlag       1
#define bLoadIntraQuantizerMatrix       1
#define bIntraQuantizer                     8                           /*      uimsbf  */
#define bLoadNonIntraQuantizerMatrix    1
#define bNonIntraQuantizer                  8                           /*      uimsbf  */


/*  group of picture flc */

#define cGroupStartCode                     0x000001B8         /*      bslbf   */

#define bTimeCode                           25                          /*      NEED    */
#define bClosedGOP                          1                           /*      NEED    */
#define bBrokenLink                         1                           /*      NEED    */


/*  picture flc  */

#define cPictureStartCode                   0x00000100         /*      bslbf   */

#define bTemporalReference                  10                          /*      uimsbf  */
#define bPictureCodingType                  3                           /*      uimsbf  */
#define bVbvDelay                           16                          /*      uimsbf  */
#define bFullPelForwardVector           1
#define bForwardFCode                       3                           /*      uimsbf  */
#define bFullPelBackwardVector              1
#define bBackwardFCode                      3                           /*      uimsbf  */
#define bExtraBitPicture                1                      /*      uimsbf  */


/*  slice flc    */

#define cSliceCodeMin                       0x00000101         /*      bslbf   */
#define cSliceCodeMax                       0x000001AF         /*      bslbf   */

#define bQuantizerScale                     5                           /*      uimsbf  */
#define bExtraBitSlice                      1                           /*      uimsbf  */


/* extension and user flc */

#define cExtensionStartCode                 0x000001B5         /*      bslbf   */
#define cUserDataStartCode                  0x000001B2         /*      bslbf   */
```

B.3.9 getbits.c

```
/* getbits.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   getbits.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:56  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "decode.pro"

#define kBufSize        4096
#define kByteAlign      0x80
#define kWdPtrStart     0x80

FILE    *gBitStream;

static unsigned char    buf[kBufSize + 4];
static unsigned int             wdPtr = kWdPtrStart;
static int                              wdIndex = 0;
static int                              numInBuf = 0;

int     Get_Bits(
    unsigned int    *destBuf,
    int                             numBits)
{
    int     index;

    *destBuf = 0;

    for (index = 0; index < numBits; index++)
    {
        /* refresh buffer if empty */
```

```
        if (wdIndex >= numInBuf)
        {
            /* read next kBufSize bytes */
            if ((numInBuf = fread(buf + 4, sizeof(unsigned char), kBufSize,
                                 gBitStream)) <= 0)
                return(0);             /*      end of bit stream       */

            wdIndex = 0;
            wdPtr = kWdPtrStart;
        }

        /* get next bit */

        *destBuf = ((buf[wdIndex + 4] & wdPtr) ?
                (*destBuf << 1) | 0x01 : (*destBuf << 1));

        /* update bit pointer */
        if (wdPtr == 0x01)
        {
            wdIndex++;
            wdPtr = kWdPtrStart;
        }
        else
            wdPtr >>= 1;
    }

    return(1);
}

int     Next_Bits(
    unsigned int    *destBuf,
    int                             numBits)
{
    unsigned int    wdPtrSave;
    int                             wdIndexSave;
    int                             index;

    /* save current buffer state */
    wdPtrSave = wdPtr;
    wdIndexSave = wdIndex;

    *destBuf = 0;

    for (index = 0; index < numBits; index++)
    {
        /* refresh buffer if empty */
        if (wdIndex >= numInBuf)
        {
            /* save last 4 bytes of old content */
            buf[0]=buf[numInBuf];
            buf[1]=buf[numInBuf + 1];
            buf[2]=buf[numInBuf + 2];
            buf[3]=buf[numInBuf + 3];

            wdIndexSave -= numInBuf;

            /* read next kBufSize bytes */
            if ((numInBuf = fread(buf + 4, sizeof(unsigned char), kBufSize,
                                 gBitStream)) <= 0)
                return(0);             /*      end of bit stream       */

            wdIndex = 0;
            wdPtr = kWdPtrStart;
        }

        /* get next bit */

        *destBuf = ((buf[wdIndex + 4] & wdPtr) ?
                (*destBuf << 1) | 0x01 : (*destBuf << 1));

        /* update bit pointer */
        if (wdPtr == 0x01)
        {
            wdIndex++;
            wdPtr = kWdPtrStart;
        }
```

```
        else
            wdPtr >>= 1;
    }

    /* restore previous buffer state */
    wdPtr = wdPtrSave;
    wdIndex = wdIndexSave;

    return(1);
}

int Next_Start_Code()
{
    unsigned int buf;

    /* locate next start code */

    if (wdPtr != kByteAlign) /* not byte aligned */
    {
        /* skip stuffed zero bits */
        wdPtr=kByteAlign;
        wdIndex++;
    }

    if (!Next_Bits(&buf, 24))
        return(0); /* end of bitstream */

    while (buf != 0x000001)
    {
        if (!Get_Bits(&buf, 8)) /* zero byte */
            return(0);
        if (!Next_Bits(&buf, 24))
            return(0);
    }

    return(1);
}

int Get(
    int numBits)
{
    unsigned int buf;

    if (! Get_Bits(&buf, numBits))
    {
        printf ("Error (Get - End Of File)\n");
        return(0);
    }

    return(buf);
}
```

B.3.10 getcode.c

```
/* File: getcode.c  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
```

```
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   getcode.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:57  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "vlc.h"
#include "decode.pro"

int     Get_AC_Code(
    BYTE                    *bit_count,
    BYTE                    *bit_pattern,
    int                     table_entries,
    int                     *sign_bit,
    int                     *run,
    int                     *level,
    int                     first)
{
    unsigned int    buf, tempInt = 0;
    int                             num_bits = 0, code;

    while (Get_Bits(&buf, 1))
    {
        num_bits++;
        tempInt = (tempInt << 1);

        /*      check for Escape code   */

        if ((num_bits == AC_escape_length) && ((tempInt | buf) == AC_escape_value))
        {
            /*      escape encountered      */

            Get_Bits(&buf, 6);               /*      get run code    */

            *run = buf;

            Get_Bits(&buf, 8);               /*      get level code  */

            if (buf == 128)
            {
                /*      28 bit code - negative level    */

                Get_Bits(&buf, 8);       /*      get level code  */
                if((buf == 0) || (buf > 128))
                    printf ("Error (Get_AC_Code) - invalid level %d\n", buf);
                *level = buf - 256;
            }
            else if (buf == 0)
            {
                /*      28 bit code - positive level    */
```

```
                Get_Bits(&buf, 8);       /*      get level code  */
                if(buf < 128)
                    printf ("Error (Get_AC_Code) - invalid level %d\n", buf);
                *level = buf;
            }
            else
            {
                /*  20 bit code */
                if (buf == 0)
                    printf ("Error (Get_AC_Code) - invalid level %d\n", buf);

                *level = (buf<128) ? buf : (buf - 256);
            }

            return(-2);             /*      Escape encounter        */
        }

        if (first && (num_bits == AC_first_length) && (tempInt == AC_first_value))
        {
            /* modified table entry for dct_coeff_first */
            code = 1;
        }
        else
            code = Find_VLC(bit_count, bit_pattern, table_entries, tempInt, num_bits);

        if (code != table_entries)
        {
            /* don't return if code==0 (EOB) but buf==1 (EOB has buf==0) */
            if (code || (! buf))
            {
                *sign_bit = buf;

                return(code);
            }
        }

        tempInt |= buf;
    }

    return(-1);
}


int     Get_Code(
    BYTE                    *bit_count,
    BYTE                    *bit_pattern,
    int                     table_entries)
{
    unsigned int    buf, tempInt = 0;
    int             num_bits = 0, code;

    while (Get_Bits(&buf, 1))
    {
        num_bits++;
        tempInt = (tempInt << 1) | buf;

        code = Find_VLC(bit_count, bit_pattern, table_entries, tempInt, num_bits);

        if (code != table_entries)
        {
            return(code);
        }
    }

    return(-1);
}


int     Find_VLC(
    BYTE                    *bit_count,
    BYTE                    *bit_pattern,
    int                     table_entries,
    unsigned int            buf,
    int                     num_bits)
{
    int             index, maxBits = 0;
```

```
    for (index = 0; index < table_entries; index++)
    {
        if ((bit_count[index] == num_bits) && (bit_pattern[index] == buf))
            return(index);

        if (maxBits < bit_count[index])
            maxBits = bit_count[index];
    }

    if (num_bits >= maxBits)
    {
        printf ("Error (Find_VLC) - %d 0x%x\n", num_bits, buf);
        return(-1);                     /*      mismatch table and buf  */
    }

    return(index);
}


/* formerly file: getcoeffs.c  */

int     Get_AC_Coefs(
    int             *currentMBDiff,
    int             currentIndex,
    int             offset)
{
    extern BYTE     AC_length[], AC_value[];
    extern short    AC_run[], AC_level[];

    int             row, code, sign, run, level, index, col;

    col = currentIndex;

    for (row = 0; row < kDCTSize; row++, currentIndex += offset)
    {
        for ( ; col < kDCTSize; col++, currentIndex++)
        {
            /*      get (first or next) DCT coefficient  */

            if ((code = Get_AC_Code(AC_length, AC_value,
                112, &sign, &run, &level, currentIndex==0)) == -1)
            {
                printf ("Error (Get_AC_Coefs) dct_coef_first - row %d col %d code %x\n", row,
col, code);
                return(0);
            }

            if (code == 0)
                return(1);              /*      EOB encountered */

            if (code != -2)         /*      not escape      */
            {
                run = AC_run[code];
                level = sign ? -AC_level[code] : AC_level[code];
            }

            for (index = 0; index < run; index++)
            {
                col++;
                currentIndex++;

                if (col == kDCTSize)
                {
                    row++;
                    col = 0;
                    currentIndex += offset;
                }
            }

            currentMBDiff[currentIndex] = level;

        }

        col = 0;
    }
```

```
    /*      remove EOB code */

    if ((code = Get_AC_Code(AC_length, AC_value, 112,
            &sign, &run, &level, 0)) == -1)
    {
        printf ("Error (Get_AC_Coefs) dct_next: row %d col %d code %x\n", row, col, code);
        return(0);
    }

    if (code)
    {
        printf ("Error (Get_AC_Coefs) - remove EOB\n");
        return(0);
    }

    return(1);
}
```

B.3.11 global.h

```
/* global.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   global.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
*/

#define AC_escape_value         1
#define AC_escape_length        6
#define AC_first_length         2       /*  dct_coeff_first bit count */
#define AC_first_value          2       /*  dct_coeff_first pattern */
```

```
BYTE    MBA_length[35] = {
    11, 1, 3, 3, 4, 4, 5, 5, 7, 7, 8, 8, 8, 8, 8, 8,
       10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,
       11,11,11,11};

BYTE    MBA_value[35] = {
     8, 1, 3, 2, 3, 2, 3, 2, 7, 6,11,10, 9, 8, 7, 6,
       23,22,21,20,19,18,35,34,33,32,31,30,29,28,27,
       26,25,24,15};

BYTE    CBP_length[66] = {0,
    5, 5, 6, 4, 7, 7, 8, 4, 7, 7,
    8, 5, 8, 8, 8, 4, 7, 7, 8, 5,
    8, 8, 8, 6, 8, 8, 9, 5, 8, 8,
    9, 4, 7, 7, 8, 6, 8, 8, 9, 5,
    8, 8, 8, 5, 8, 8, 9, 5, 8, 8,
    8, 5, 8, 8, 9, 5, 8, 8, 9, 3,
    5, 5, 6};

BYTE    CBP_value[66] = {0,
    11,  9, 13, 13, 23, 19, 31, 12, 22, 18,
    30, 19, 27, 23, 19, 11, 21, 17, 29, 17,
    25, 21, 17, 15, 15, 13,  3, 15, 11,  7,
     7, 10, 20, 16, 28, 14, 14, 12,  2, 16,
    24, 20, 16, 14, 10,  6,  6, 18, 26, 22,
    18, 13,  9,  5,  5, 12,  8,  4,  4,  7,
    10,  8, 12};

BYTE    DC_Y_length[9]   = {3, 2, 2, 3, 3, 4, 5,  6,  7};
BYTE    DC_Y_value[9]  = {4, 0, 1, 5, 6,14,30, 62,126};
BYTE    DC_C_length[9]  = {2, 2, 2, 3, 4, 5, 6,  7,  8};
BYTE    DC_C_value[9] = {0, 1, 2, 6,14,30,62,126,254};

BYTE    AC_length[112] = {
     2, 3, 4, 5, 5, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8,
     9, 9, 9, 9, 9, 9, 9, 9,11,11,11,11,11,11,11,11,
    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};

BYTE    AC_value[112] = {
     2, 6, 6, 8,10,10,14,12,12,14,10, 8,12, 8,14,10,
    76,66,74,72,78,70,68,64,20,24,22,30,18,28,26,16,
    58,48,38,32,54,40,56,36,60,42,34,62,52,50,46,44,
    52,50,48,46,44,42,40,38,36,34,32,62,60,58,56,54,
    62,60,58,56,54,52,50,48,46,44,42,40,38,36,34,32,
    48,46,44,42,40,38,36,34,32,62,60,58,56,54,52,50,
     38,36,34,32,40,52,50,48,46,44,42,62,60,58,56,54
};

short   AC_run[112] = {
     0, 0, 1, 0, 2, 0, 3, 4, 1, 5, 6, 7, 0, 2, 8, 9,
     0, 0, 1, 3,10,11,12,13, 0, 1, 2, 4, 5,14,15,16,
     0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8,17,18,19,20,21,
     0, 0, 0, 0, 1, 1, 2, 3, 5, 9,10,22,23,24,25,26,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
     0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
     1, 1, 1, 1, 6,11,12,13,14,15,16,27,28,29,30,31};

short   AC_level[112] = {
     0, 1, 1, 2, 1, 3, 1, 1, 2, 1, 1, 1, 4, 2, 1, 1,
     5, 6, 3, 2, 1, 1, 1, 1, 7, 4, 3, 2, 2, 1, 1, 1,
     8, 9,10,11, 5, 4, 3, 3, 2, 2, 2, 1, 1, 1, 1, 1,
    12,13,14,15, 6, 7, 5, 4, 3, 2, 2, 1, 1, 1, 1, 1,
    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, 8, 9,10,11,12,13,14,
    15,16,17,18, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1
};

BYTE    MV_length[33] = {
     1, 3, 4, 5, 7, 8, 8, 8, 10,10,10,11,11,11,11,11,11,
        3, 4, 5, 7, 8, 8, 8, 10,10,10,11,11,11,11,11,11};

BYTE    MV_value[33] = {
```

```
     1, 2, 2, 2, 6,10, 8, 6, 22,20,18,34,32,30,28,26,24,
        3, 3, 3, 7,11, 9, 7, 23,21,19,35,33,31,29,27,25};

BYTE    MB_typeI_length[2] = {1, 2};
BYTE    MB_typeI_value[2] = {1, 1};
BYTE    MB_typeI_switch[2] = {kMbIntra, kMbIntra | kMbQuant};

BYTE    MB_typeP_length[7] = {1, 2, 3, 5, 5, 5, 6};
BYTE    MB_typeP_value[7] = {1, 1, 1, 3, 2, 1, 1};
short   MB_typeP_switch[7] = {kMbForward | kMbPattern,
        kMbPattern, kMbForward, kMbIntra,
        kMbQuant | kMbForward | kMbPattern,
        kMbQuant | kMbPattern, kMbQuant | kMbIntra};

BYTE    MB_typeB_length[11] = {2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 6};
BYTE    MB_typeB_value[11] = {2, 3, 2, 3, 2, 3, 3, 2, 3, 2, 1};
short   MB_typeB_switch[11] = {kMbForward | kMbBackward,
        kMbForward | kMbBackward | kMbPattern,
        kMbBackward, kMbBackward | kMbPattern, kMbForward,
        kMbForward | kMbPattern, kMbIntra,
        kMbQuant | kMbForward | kMbBackward | kMbPattern,
        kMbQuant | kMbForward | kMbPattern,
        kMbQuant | kMbBackward |   kMbPattern,
        kMbQuant | kMbIntra};
```

B.3.12 idct.c

```
/* idct.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *  $Log:   idct.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:55  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */
```

```
#include <math.h>
#include "consts.h"

#ifndef PI
#ifdef M_PI
#define PI  M_PI
#else
#define PI  3.14159265358979323846
#endif
#endif

double  TwiddleFor[kDCTSize][kDCTSize];
double  TwiddleInv[kDCTSize][kDCTSize];

/**********************************************************************
*
* Module name: Init_DCT
*
* Initializes the DCT scale and twiddle factors
*
**********************************************************************/

void Init_DCT(void)
{
    extern double   TwiddleFor[][kDCTSize];
    extern double   TwiddleInv[][kDCTSize];

    /* Local variables */

    double  alpha[kDCTSize];
    short i,j;

    /* Execute */

    alpha[0] = sqrt((double)(1.0 / kDCTSize));

    for(i = 1; i < kDCTSize; i++)
        alpha[i] = sqrt((double)(2.0 / kDCTSize));

    for(i = 0; i < kDCTSize; i++)
    {
        for(j = 0; j < kDCTSize; j++)
        {
            TwiddleFor[i][j] = alpha[i] * cos((double)((2 * j + 1) * i * PI) /
                    (2 * kDCTSize));
            TwiddleInv[i][j] = alpha[j] * cos((double)((2 * i + 1) * j * PI) /
                    (2 * kDCTSize));
        }
    }
    /*
    alpha[0] = 1.0 / (2.0 * kDCTSize);

    for(i = 1; i < kDCTSize; i++)
        alpha[i] = 1.0 / kDCTSize;

    for(i = 0; i < kDCTSize; i++)
    {
        for(j = 0; j < kDCTSize; j++)
        {
            TwiddleFor[i][j] = 2.0 * cos((double)((2 * j + 1) * i * PI) /
                    (2 * kDCTSize));
            TwiddleInv[i][j] = alpha[j] * cos((double)((2 * i + 1) * j * PI) /
                    (2 * kDCTSize));
        }
    }
    */
}

/**********************************************************************
*
* Module name: IDCT
*
* Calculates an inverse discrete cosine transform
*
**********************************************************************/
```

```
void idct8x8(
    float   dct_img[],
    float   recon_img[])
{
    extern double   TwiddleInv[][kDCTSize];

    /* Local variables */

    short   i, m, n, index, index1;
    float   tmpImg[kDCTLen];

    /* Execute */

    for(i = 0, index = 0; i < kDCTSize; i++, index += kDCTSize)
    {
        for(m = 0; m < kDCTSize; m++)
        {
            tmpImg[index + m] = 0;

            for(n = 0; n < kDCTSize; n++)
            {
                tmpImg[index + m] += dct_img[index + n] * TwiddleInv[m][n];
            }
        }
    }

    for(i = 0; i < kDCTSize; i++)
    {
        for(m = 0, index = i; m < kDCTSize; m++, index += kDCTSize)
        {
            recon_img[index] = 0;

            for(n = 0, index1 = i; n < kDCTSize; n++, index1 += kDCTSize)
            {
                recon_img[index] += tmpImg[index1] * TwiddleInv[m][n];
            }
        }
    }
}
```

B.3.13 initial.c

```
/* File:  initial.c  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
```

```
 */

/*
 *      $Log:   initial.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.4  93/09/03  01:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.1.1.1  93/03/29  11:27:55  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <malloc.h>
#include "consts.h"
#include "types.h"
#include "decode.h"
#include "decode.pro"

void    Initial(
    tParameter              *aParameter,
    char                    *parameterFName)
{
    extern FILE     *gBitStream;

    FILE    *fptr;
    char    lineBuf[kMaxCharBufLen];

    /* get user input data */

    if ((fptr = fopen (parameterFName, "r")) == 0)
    {
        printf ("input parameter file '%s' not found\n", parameterFName);
        exit(-1);
    }

    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", aParameter->bitStreamFName);
    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", aParameter->outputSequence);
    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &(aParameter->outputFormat));
    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%s", aParameter->statusFName);
    fgets (lineBuf, kMaxCharBufLen, fptr);
    sscanf (lineBuf, "%d", &TraceLevel);

    fclose (fptr);

    if ((gBitStream = fopen (aParameter->bitStreamFName, "rb")) == NULL)
    {
        printf ("error opening bitstream file '%s' ... Terminating ...\n",
                aParameter->bitStreamFName);
        exit(-1);
    }

    if (TraceLevel > 0)
    {
        if (aParameter->statusFName[0] == '-')
          TraceFile = stdout;
        else if ((TraceFile = fopen (aParameter->statusFName, "w")) == NULL)
        {
            printf ("error creating trace file '%s' ... Terminating ...\n",
                    aParameter->statusFName);
            exit(-1);
        }
    }
```

```
    Init_DCT();
}


void Init_Data(
    tData                   *aData)
{
    int numLuminPixels, numChromPixels;

    numLuminPixels = (16 * mb_width) * (16 * mb_height);
    numChromPixels = (8 * mb_width) * (16 * mb_height);

    /* Reconstructed picture array memory allocation */

    aData->luminPicture0R = Barray(numLuminPixels);
    aData->luminPicture1R = Barray(numLuminPixels);
    aData->luminPicture2R = Barray(numLuminPixels);

    aData->chromUPicture0R = Barray(numChromPixels);
    aData->chromUPicture1R = Barray(numChromPixels);
    aData->chromUPicture2R = Barray(numChromPixels);

    aData->chromVPicture0R = Barray(numChromPixels);
    aData->chromVPicture1R = Barray(numChromPixels);
    aData->chromVPicture2R = Barray(numChromPixels);

    /*      assign working pointers */

    aData->firstPredictYR = aData->luminPicture0R;
    aData->firstPredictUR = aData->chromUPicture0R;
    aData->firstPredictVR = aData->chromVPicture0R;
    aData->secondPredictYR = aData->luminPicture1R;
    aData->secondPredictUR = aData->chromUPicture1R;
    aData->secondPredictVR = aData->chromVPicture1R;
    aData->predictYR = aData->luminPicture2R;
    aData->predictUR = aData->chromUPicture2R;
    aData->predictVR = aData->chromVPicture2R;
}

BYTE    *Barray(
    int     size)
{
    BYTE    *a;

    a = (BYTE *) malloc((unsigned) size * sizeof(BYTE));

    if (!a)
    {
        printf("\nBarray: Failure in allocating arrary memory\n");
        exit(-1);
    }

    return a;
}

short *Sarray(
    int     size)
{
    short   *a;

    a = (short *) malloc((unsigned) size * sizeof(short));

    if (!a)
    {
        printf("\nSarray: Failure in allocating arrary memory\n");
        exit(-1);
    }

    return a;
}
```

B.3.14 iquant.c

```
/* File: iquant.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */


/*
 *      $Log:   iquant.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:55  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "quant.h"
#include "decode.pro"


void    Inverse_Quantize_Intra_MB(
    short   codedBPattern,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             MQuant)
{
    int             line, pix, row, col, quantIndex;
    int             currentIndex;
    int             tInt;

    /*      lumin   */

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex = line * kMBWidth + pix;
```

```
            currentMBDiff[currentIndex] *= 8;

            if (currentMBDiff[currentIndex] < -2048)
                currentMBDiff[currentIndex] = -2048;
            else if (currentMBDiff[currentIndex] > 2047)
                currentMBDiff[currentIndex] = 2047;

            /*      exclude dc coeff        */

            currentIndex++;

            for (row = 0, col = 1, quantIndex = 1; row < kDCTSize;
                    row++, col = 0, currentIndex += kMBWidthDCT)
            {
                for ( ; col < kDCTSize; col++, currentIndex++, quantIndex++)
                {
                    if (currentMBDiff[currentIndex] != 0)
                    {
                        /*      rec = MQuant * 2 * QAC * WI / 16        */

                        tInt = (MQuant * currentMBDiff[currentIndex] *
                                (int) tQuantIntra[quantIndex]) / 8;

                        if (! (tInt % 2))               /*      even    */
                        {
                            if (tInt < 0)
                                tInt += 1;
                            else if (tInt > 0)
                                tInt -= 1;
                        }

                        if (tInt <= -2048)
                            currentMBDiff[currentIndex] = -2048;
                        else if (tInt >= 2047)
                            currentMBDiff[currentIndex] = 2047;
                        else
                            currentMBDiff[currentIndex] = tInt;
                    }
                }
            }
        }
    }

    /*      chrom   */

    currentMBDiffU[0] *= 8;

    if (currentMBDiffU[0] < -2048)
        currentMBDiffU[0] = -2048;
    else if (currentMBDiffU[0] > 2047)
        currentMBDiffU[0] = 2047;

    currentIndex = 1;

    for (row = 1; row < kDCTLen; row++, currentIndex++)
    {
        /*      U       */

        if (currentMBDiffU[currentIndex] != 0)
        {
            /*      rec = MQuant * 2 * QAC * WI / 16        */

            tInt = (MQuant * currentMBDiffU[currentIndex] *
                    (int) tQuantIntra[currentIndex]) / 8;

            if (! (tInt % 2))               /*      even    */
            {
                if (tInt < 0)
                    tInt += 1;
                else if (tInt > 0)
                    tInt -= 1;
            }

            if (tInt <= -2048)
                currentMBDiffU[currentIndex] = -2048;
            else if (tInt >= 2047)
```

```
                currentMBDiffU[currentIndex] = 2047;
            else
                currentMBDiffU[currentIndex] = tInt;
        }
    }

    currentMBDiffV[0] *= 8;

    if (currentMBDiffV[0] < -2048)
        currentMBDiffV[0] = -2048;
    else if (currentMBDiffV[0] > 2047)
        currentMBDiffV[0] = 2047;

    currentIndex = 1;

    for (row = 1; row < kDCTLen; row++, currentIndex++)
    {
        /*      V       */

        if (currentMBDiffV[currentIndex] != 0)
        {
            /*      rec = MQuant * 2 * QAC * WI / 16        */

            tInt = (MQuant * currentMBDiffV[currentIndex] *
                    (int) tQuantIntra[currentIndex]) / 8;

            if (! (tInt % 2))               /*      even    */
            {
                if (tInt < 0)
                    tInt += 1;
                else if (tInt > 0)
                    tInt -= 1;
            }

            if (tInt <= -2048)
                currentMBDiffV[currentIndex] = -2048;
            else if (tInt >= 2047)
                currentMBDiffV[currentIndex] = 2047;
            else
                currentMBDiffV[currentIndex] = tInt;
        }
    }
}


void    Inverse_Quantize_Non_Intra_MB(
    short   codedBPattern,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             MQuant)
{
    int             code, line, pix, row, col, quantIndex;
    int             currentIndex;
    int             tInt;

    /*      lumin   */

    code = 0x20;

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            if (! (codedBPattern & code))
            {
                code >>= 1;
                continue;
            }

            code >>= 1;

            currentIndex = line * kMBWidth + pix;

            for (row = 0, quantIndex = 0; row < kDCTSize; row++,
                    currentIndex += kMBWidthDCT)
```

```
        {
            for (col = 0; col < kDCTSize; col++,
                    currentIndex++, quantIndex++)
            {
                if (currentMBDiff[currentIndex])
                {
                    if (currentMBDiff[currentIndex] > 0)
                        tInt = ((2 * currentMBDiff[currentIndex] + 1) *
                                MQuant * (int) tQuant[quantIndex]) / 16;
                    else    /*      < 0             */
                        tInt = ((2 * currentMBDiff[currentIndex] - 1) *
                                MQuant * (int) tQuant[quantIndex]) / 16;

                    if (! (tInt % 2))               /*      even    */
                    {
                        if (tInt < 0)
                            tInt += 1;
                        else if (tInt > 0)
                            tInt -= 1;
                    }

                    if (tInt <= -2048)
                        currentMBDiff[currentIndex] = -2048;
                    else if (tInt >= 2047)
                        currentMBDiff[currentIndex] = 2047;
                    else
                        currentMBDiff[currentIndex] = tInt;
                }
            }
        }
    }
}

/*      chrom   */

if (codedBPattern & 0x2)
{
    /*      U       */

    for (row = 0, currentIndex = 0; row < kDCTLen; row++, currentIndex++)
    {
        if (currentMBDiffU[currentIndex])
        {
            if (currentMBDiffU[currentIndex] > 0)
                tInt = ((2 * currentMBDiffU[currentIndex] + 1) *
                        MQuant * (int) tQuant[currentIndex]) / 16;
            else
                tInt = ((2 * currentMBDiffU[currentIndex] - 1) *
                        MQuant * (int) tQuant[currentIndex]) / 16;

            if (! (tInt % 2))               /*      even    */
            {
                if (tInt < 0)
                    tInt += 1;
                else if (tInt > 0)
                    tInt -= 1;
            }

            if (tInt <= -2048)
                currentMBDiffU[currentIndex] = -2048;
            else if (tInt >= 2047)
                currentMBDiffU[currentIndex] = 2047;
            else
                currentMBDiffU[currentIndex] = tInt;
        }
    }
}

if (codedBPattern & 0x1)
{
    /*      V       */

    for (row = 0, currentIndex = 0; row < kDCTLen; row++, currentIndex++)
    {
        if (currentMBDiffV[currentIndex])
        {
```

```
                if (currentMBDiffV[currentIndex] > 0)
                    tInt = ((2 * currentMBDiffV[currentIndex] + 1) *
                            MQuant * (int) tQuant[currentIndex]) / 16;
                else
                    tInt = ((2 * currentMBDiffV[currentIndex] - 1) *
                            MQuant * (int) tQuant[currentIndex]) / 16;

                if (! (tInt % 2))              /*      even    */
                {
                    if (tInt < 0)
                        tInt += 1;
                    else if (tInt > 0)
                        tInt -= 1;
                }

                if (tInt <= -2048)
                    currentMBDiffV[currentIndex] = -2048;
                else if (tInt >= 2047)
                    currentMBDiffV[currentIndex] = 2047;
                else
                    currentMBDiffV[currentIndex] = tInt;
            }
        }
    }
}
```

B.3.15 makefile

```
# makefile for decoder based on GNU gcc

OBJ= decode.o getbits.o picture.o slice.o sequence.o initial.o \
        iquant.o getcode.o writepic.o perfidct.o idct.o \
        convert.o mb.o constr.o transfer.o

CFLAGS = -c -O2 -Wall
COMPILER = gcc

decode: $(OBJ)
    $(COMPILER) $(OBJ) -o decode -lm

decode.o:       decode.c
        $(COMPILER) $(CFLAGS) decode.c

transfer.o:     transfer.c
        $(COMPILER) $(CFLAGS) transfer.c

constr.o:       constr.c
        $(COMPILER) $(CFLAGS) constr.c

convert.o:      convert.c
        $(COMPILER) $(CFLAGS) convert.c

initial.o:      initial.c
        $(COMPILER) $(CFLAGS) initial.c

iquant.o:       iquant.c
        $(COMPILER) $(CFLAGS) iquant.c

perfidct.o:     perfidct.c
        $(COMPILER) $(CFLAGS) perfidct.c

idct.o:         idct.c
        $(COMPILER) $(CFLAGS) idct.c

writepic.o:     writepic.c
        $(COMPILER) $(CFLAGS) writepic.c

getbits.o:      getbits.c
        $(COMPILER) $(CFLAGS) getbits.c

slice.o:        slice.c
        $(COMPILER) $(CFLAGS) slice.c

picture.o:      picture.c
```

```
        $(COMPILER) $(CFLAGS) picture.c

sequence.o:     sequence.c
        $(COMPILER) $(CFLAGS) sequence.c

getcode.o:      getcode.c
        $(COMPILER) $(CFLAGS) getcode.c

mb.o:           mb.c
        $(COMPILER) $(CFLAGS) mb.c

constr.c: consts.h types.h decode.h decode.pro
GetBits.c: consts.h types.h decode.pro
GetCode.c: consts.h types.h vlc.h decode.pro
PerfIdct.c: consts.h types.h decode.pro
writepic.c: consts.h types.h decode.h decode.pro
convert.c: consts.h types.h decode.pro
decode.c: consts.h types.h decode.pro decode.h
idct.c: consts.h
initial.c: consts.h types.h decode.h decode.pro
iquant.c: consts.h types.h quant.h decode.pro
mb.c: consts.h types.h vlc.h decode.pro
picture.c: consts.h types.h flc.h decode.h decode.pro
sequence.c: consts.h types.h flc.h decode.h decode.pro
slice.c: consts.h types.h flc.h vlc.h global.h decode.h decode.pro
transfer.c: consts.h types.h decode.h decode.pro
```

B.3.16 mb.c

```
/* File: mb.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   mb.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:57  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
```

```
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "vlc.h"
#include "decode.h"
#include "decode.pro"


int     Process_MB(
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV,
    int             *dct_dc_y_past,
    int             *dct_dc_cb_past,
    int             *dct_dc_cr_past,
    int             macroblock_type,
    int             coded_block_pattern)
{
    int             block_number, line, pix, mbIntra;

    mbIntra = (macroblock_type & kMbIntra) != 0;
    block_number = 0;

    /* Y blocks */

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            if (!Process_Block(block_number, coded_block_pattern,
                mbIntra, dct_dc_y_past, currentMBDiff + line * kMBWidth + pix))
                return(0);

            block_number++;
        }
    }

    /* Cb */

    if (!Process_Block(4, coded_block_pattern, mbIntra, dct_dc_cb_past, currentMBDiffU))
        return(0);

    /* Cr */

    if (!Process_Block(5, coded_block_pattern, mbIntra, dct_dc_cr_past, currentMBDiffV))
        return(0);

    return(1);
}

int Process_Block(
    int             block_number,
    int             coded_block_pattern,
    int             mbIntra,
    int             *dct_dc_past,
    int             *dct_recon)
{
    extern BYTE     DC_Y_length[];
    extern BYTE     DC_Y_value[];
    extern BYTE     DC_C_length[];
    extern BYTE     DC_C_value[];

    int             currentIndex, size, diff_DC;
    unsigned int    buf;

    if ((coded_block_pattern & (32>>block_number)) || mbIntra)
    {
        currentIndex = 0;
        if (mbIntra)
        {
            /*      get dct_dc_size */

            if (block_number < 4)
```

```
                /* luminance */
                size = Get_Code(DC_Y_length, DC_Y_value, 9);
            else
                /* chrominance */
                size = Get_Code(DC_C_length, DC_C_value, 9);

            if (size == -1)
            {
                printf ("Error (Process_Block) - GetCode - %d\n", block_number);
                return(0);
            }

            if (size != 0)
            {
                /* get dct_dc_differential */
                if (Get_Bits(&buf, size) == 0)
                {
                    printf ("Error (Process_Block) - Get_Bits after fetching DC coef - %d\n",
block_number);
                    return(0);
                }

                /* compute dct_zz[0] */
                if (buf & (1 << (size - 1)))
                    diff_DC = buf; /* positive */
                else
                    diff_DC = (-1 << size) | (buf + 1); /* negative */
            }
            else
                diff_DC = 0; /* zero */

            dct_recon[0] = *dct_dc_past + (diff_DC * 1);
            *dct_dc_past = dct_recon[0];
            currentIndex = 1;
        }

        /*      get AC coeffs   */

        if (! Get_AC_Coefs(dct_recon, currentIndex,
                        (block_number < 4) ? kMBWidthDCT : 0))
        {
            printf ("Error (Process_Block) - Get_AC_Coefs - %d\n",
                        block_number);
            return(0);
        }
    }

    return(1);
}
```

B.3.17 perfidct.c

```
/* perfidct.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
```

```
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   perfidct.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include "consts.h"
#include "types.h"
#include "decode.pro"

/* de-zz scan order offset */

static BYTE     inverse_zig_zag_scan[64] =
{
     0,  1,  8, 16,  9,  2,  3, 10,
    17, 24, 32, 25, 18, 11,  4,  5,
    12, 19, 26, 33, 40, 48, 41, 34,
    27, 20, 13,  6,  7, 14, 21, 28,
    35, 42, 49, 56, 57, 50, 43, 36,
    29, 22, 15, 23, 30, 37, 44, 51,
    58, 59, 52, 45, 38, 31, 39, 46,
    53, 60, 61, 54, 47, 55, 62, 63
};


void    Perform_IDCT(
    int             mbType,
    int             *currentMBDiff,
    int             *currentMBDiffU,
    int             *currentMBDiffV)
{
    int             currentIndex, currentIndex1, currentIndex2, line, pix, row, col;
    float           dctBuf[kDCTLen], dctTrBuf[kDCTLen];

    for (line = 0; line < kMBHeight; line += kDCTSize)
    {
        for (pix = 0; pix < kMBWidth; pix += kDCTSize)
        {
            currentIndex2 = line * kMBWidth + pix;

            for (row = 0, currentIndex = 0, currentIndex1 = currentIndex2;
                    row < kDCTSize; row++, currentIndex1 += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++,
                        currentIndex++, currentIndex1++)
                {
                    /*      perform de-zz scan      */

                    dctTrBuf[inverse_zig_zag_scan[currentIndex]] =
                            currentMBDiff[currentIndex1];
                }
            }

            /*      perform inverse DCT     */

            idct8x8(dctTrBuf, dctBuf);
```

```
            /*      return re-transformed data              */

            for (row = 0, currentIndex = 0, currentIndex1 = currentIndex2;
                    row < kDCTSize; row++, currentIndex1 += kMBWidthDCT)
            {
                for (col = 0; col < kDCTSize; col++,
                        currentIndex++, currentIndex1++)
                {
                    currentMBDiff[currentIndex1] =
                            (dctBuf[currentIndex] > 0.0) ?
                            (dctBuf[currentIndex] + 0.5) :
                            (dctBuf[currentIndex] - 0.5);
                }
            }
        }
    }

    /*      chrom   */

    /*      U       */

    for (row = 0; row < kDCTLen; row++)
    {
        /*      perform de-zz scan      */

        dctTrBuf[inverse_zig_zag_scan[row]] = currentMBDiffU[row];
    }

    /*      perform IDCT    */

    idct8x8(dctTrBuf, dctBuf);

    /*      return re-transformed data              */

    for (row = 0; row < kDCTLen; row++)
    {
        currentMBDiffU[row] = (dctBuf[row] > 0.0) ?
                (dctBuf[row] + 0.5) : (dctBuf[row] - 0.5);
    }

    /*      V       */

    for (row = 0; row < kDCTLen; row++)
    {
        /*      perform de-zz scan      */

        dctTrBuf[inverse_zig_zag_scan[row]] = currentMBDiffV[row];
    }

    /*      perform DCT     */

    idct8x8(dctTrBuf, dctBuf);

    /*      return re-transformed data              */

    for (row = 0; row < kDCTLen; row++)
    {
        currentMBDiffV[row] = (dctBuf[row] > 0.0) ?
                (dctBuf[row] + 0.5) : (dctBuf[row] - 0.5);
    }
}
```

B.3.18 picture.c

```
/* picture.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
```

```
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   procPicture.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.2  93/06/15  15:41:31  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "flc.h"
#include "decode.h"
#include "decode.pro"

int     Process_Picture(
    int             pictureNum,
    tParameter      *parameter,
    tData           *data)
{
    unsigned int    buf;
    BYTE            *buf_Y, *buf_Cb, *buf_Cr;
    int             picture_coding_type, forward_f_code, backward_f_code;
    int             fullPelForwardV, fullPelBackwardV;
    int             temporal_reference, vbv_delay;
    static char     *typeTab[8]={"forbidden","intra-coded (I)",
                      "predictive-coded (P)",
                      "bidirectionally-predictive-coded (B)",
                      "dc intra-coded (D)", "reserved",
                      "reserved", "reserved"};


    /* picture layer */

    temporal_reference = Get(bTemporalReference);
    picture_coding_type = Get(bPictureCodingType);
    vbv_delay = Get(bVbvDelay);

    if (TraceLevel >= 1)
    {
        fprintf(TraceFile, "Picture Header:\n");
        fprintf(TraceFile, " temporal reference: %d\n", temporal_reference);
        fprintf(TraceFile, " picture coding type: %s\n", typeTab[picture_coding_type]);
        fprintf(TraceFile, " VBV delay: %d\n", vbv_delay);
    }
```

```
    if ((picture_coding_type == kPPicture) || (picture_coding_type == kBPicture))
    {
        fullPelForwardV = Get(bFullPelForwardVector);
        forward_f_code = Get(bForwardFCode);

        if (TraceLevel >= 1)
        {
            fprintf(TraceFile, " full pel forward vector: %s\n",
                fullPelForwardV ? "yes" : "no");
            if (fullPelForwardV)
                fprintf(TraceFile,
                    " forward_f_code: %d, (range %d to %d)\n", forward_f_code,
                    -(8 << forward_f_code), (8 << forward_f_code) - 1);
            else
                fprintf(TraceFile,
                    " forward_f_code: %d, (range %.1f to %.1f)\n",
                    forward_f_code,
                    -0.5*(8 << forward_f_code),
                    0.5*((8 << forward_f_code) - 1));
        }
    }

    if (picture_coding_type == kBPicture)
    {
        fullPelBackwardV = Get(bFullPelBackwardVector);
        backward_f_code = Get(bBackwardFCode);

        if (TraceLevel >= 1)
        {
            fprintf(TraceFile, " full pel backward vector: %s\n",
                fullPelBackwardV ? "yes" : "no");
            if (fullPelBackwardV)
                fprintf(TraceFile,
                    " backward_f_code: %d, (range %d to %d)\n", backward_f_code,
                    -(8 << backward_f_code), (8 << backward_f_code) - 1);
            else
                fprintf(TraceFile,
                    " backward_f_code: %d, (range %.1f to %.1f)\n",
                    backward_f_code,
                    -0.5*(8 << backward_f_code),
                    0.5*((8 << backward_f_code) - 1));
        }
    }

    /* skip extra_bit_picture & extra_information_picture */
    Get_Bits(&buf, 1);
    while(buf != 0) {
        Get_Bits(&buf, 8);
        Get_Bits(&buf, 1);
    }

    Next_Start_Code();

    Extension_And_User_Data();

    if (picture_coding_type == kBPicture)
    {
        buf_Y = data->predictYR;
        buf_Cb = data->predictUR;
        buf_Cr = data->predictVR;
    }
    else
    {
        buf_Y = data->firstPredictYR;
        buf_Cb = data->firstPredictUR;
        buf_Cr = data->firstPredictVR;

        /* Rotate the prediction buffers */
        data->firstPredictYR = data->secondPredictYR;
        data->firstPredictUR = data->secondPredictUR;
        data->firstPredictVR = data->secondPredictVR;

        data->secondPredictYR = buf_Y;
        data->secondPredictUR = buf_Cb;
        data->secondPredictVR = buf_Cr;
```

```
        /* write the previous I/P-picture */
        if(pictureNum != 0){
            if (Write_Picture(pictureNum,
                    data->firstPredictYR,
                    data->firstPredictUR,
                    data->firstPredictVR,
                    parameter))
                exit (-1);
        }
    }

    /* decode a picture */
    do
    {
        if (! Process_Slice(picture_coding_type, buf_Y, buf_Cb, buf_Cr, data,
                forward_f_code, fullPelForwardV,
                backward_f_code, fullPelBackwardV))
        {
            printf ("Error (procPicture) - procSlice\n");
            return(0);
        }
        Next_Bits(&buf, bStartCode);
    }
    while ((buf >= cSliceCodeMin) && (buf <= cSliceCodeMax));

    /* B-pictures are displayed right after they are decoded. */
    if(picture_coding_type == kBPicture){
        /* write B-picture */
        if (Write_Picture(pictureNum, buf_Y, buf_Cb, buf_Cr, parameter))
        exit (-1);
    }

    return(1);
}
```

B.3.19 quant.h

```
/* File: quant.h  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *  $Log:   quant.h,v $
```

```
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.3      93/08/20  21:15:53  hana
 * MPEG1 encoder minor revision
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

/*
 *  Default intra quantizer matrix
 *      On changing any part of this,
 *   you should make
 *     cLoadIntraIntraQuantierMatrix = 0
 *   in flc.h .
 */

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


/*
 *  Default non intra quantizer matrix
 *      On changing any part of this,
 *   you should make
 *     cLoadNonIntraQuantierMatrix = 0
 *   in flc.h .
 */

BYTE tQuant[64] =
{
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16,
    16, 16, 16, 16, 16, 16, 16, 16
};
```

B.3.20 sequence.c

```
/* file: sequence.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
```

```
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   sequence.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.2  93/06/15  15:41:31  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "flc.h"
#include "decode.h"
#include "decode.pro"

int     Process_Sequence(
    tParameter      *parameter,
    tData           *data)
{
    unsigned int buf;
    int pictureNum;
    int time_code, closed_GOP, broken_link;

    /* video sequence layer */

    pictureNum = 0;

    Next_Start_Code();

    do
    {
        if (! Get_Bits(&buf, bStartCode))
            return(0);

        if (buf != cSequenceHeaderCode)
        {
            printf ("Error (procSequence - sequence header)\n");
            return(0);
        }

        /* handle sequence header */

        Process_Sequence_Header();

        /* initialize data structures */
        if (pictureNum == 0)
            Init_Data(data);

        do
        {
            if (! Get_Bits(&buf, bStartCode))
                return(0);
```

```
        if (buf != cGroupStartCode)
        {
            printf ("Error (procSequence - group start code)\n");
            return(0);
        }

        /* group of pictures layer */

        /*      handle GOP Header       */

        time_code = Get(bTimeCode);
        closed_GOP = Get(bClosedGOP);
        broken_link = Get(bBrokenLink);

        if (TraceLevel >= 1)
        {
            fprintf(TraceFile, "Group of pictures:\n");
            fprintf(TraceFile,
                " time code (hh:mm:ss:pp): %02d:%02d:%02d:%02d\n",
                (time_code >> 19) & 0x1f, (time_code >> 13) & 0x3f,
                (time_code >> 6) & 0x3f, time_code & 0x3f);
            fprintf(TraceFile, "closed GOP: %s\n",
                closed_GOP ? "yes" : "no");
            fprintf(TraceFile, "broken link: %s\n",
                broken_link ? "yes" : "no");
        }

        Next_Start_Code();

        Extension_And_User_Data();

        do
        {
            if (! Get_Bits(&buf, bStartCode))
                return(0);

            if (buf != cPictureStartCode)
            {
                printf ("Error (procSequence - picture start code)\n");
                return(0);
            }

            Process_Picture(pictureNum, parameter, data);

            printf("Decoded Picture: %d\n",pictureNum);

            pictureNum++;

            Next_Bits(&buf, bStartCode);
        }
        while (buf == cPictureStartCode);

    }
    while (buf == cGroupStartCode);

}
while (buf == cSequenceHeaderCode);

if (! Get_Bits(&buf, bStartCode))
    return(0);

if (buf != cSequenceEndCode)
{
    printf ("Error (procSequence - sequence end code)\n");
    return(0);
}

/* write the last I/P-picture */
if (Write_Picture(pictureNum,
    data->secondPredictYR,
    data->secondPredictUR,
    data->secondPredictVR,
    parameter))
    exit (-1);

return(0);
```

```
}

void    Process_Sequence_Header(void)
{
    extern BYTE     tQuantIntra[], tQuant[];

    int             index;
    int             pel_aspect_ratio, picture_rate, bit_rate, marker_bit;
    int             vbv_buffer_size, constrained_parameters_flag;
    int             load_intra_quantizer_matrix, load_non_intra_quantizer_matrix;
    static char     *aspectTab[16] = {"forbidden", "1.0000 (VGA etc.)", "0.6735",
                "0.7031 (16:9, 625line)", "0.7615", "0.8055",
                "0.8437 (16:9, 525line)", "0.8935",
                "0.9157 (CCIR601, 625line)", "0.9815", "1.0255", "1.0695",
                "1.0950 (CCIR601, 525line)", "1.1575", "1.2015", "reserved"};
    static char *rateTab[9] = {"forbidden", "23.976", "24", "25", "29.97",
                "30", "50", "59.94", "60"};

    /*      process sequence header */

    horizontal_size = Get(bHorizontalSize);
    vertical_size = Get(bVerticalSize);
    pel_aspect_ratio = Get(bPelAspectRatio);
    picture_rate = Get(bPictureRate);
    bit_rate = Get(bBitRate);
    marker_bit = Get(bMarkerBit);
    vbv_buffer_size = Get(bVbvBufferSize);
    constrained_parameters_flag = Get(bConstrainedParameterFlag);

    mb_width = (horizontal_size + 15)/16;
    mb_height = (vertical_size + 15)/16;

    load_intra_quantizer_matrix = Get(bLoadIntraQuantizerMatrix);

    if (load_intra_quantizer_matrix)
        for (index = 0; index < 64; index++)
            tQuantIntra[index] = Get(bIntraQuantizer);

    load_non_intra_quantizer_matrix = Get(bLoadNonIntraQuantizerMatrix);

    if (load_non_intra_quantizer_matrix)
        for (index = 0; index < 64; index++)
            tQuant[index] = Get(bNonIntraQuantizer);

    if(TraceLevel>=1)
    {
        fprintf(TraceFile, "Sequence Header:\n");
        fprintf(TraceFile, " picture size (H x V): %d x %d\n",
            horizontal_size, vertical_size);
        fprintf(TraceFile, " macroblocks  (H x V): %d x %d\n",
            mb_width, mb_height);
        fprintf(TraceFile, " pel aspect ratio: %s\n",
            aspectTab[pel_aspect_ratio]);
        fprintf(TraceFile, " picture rate: %s pictures per second\n",
            (picture_rate<9) ? rateTab[picture_rate] : "reserved");
        if (bit_rate == 0x3ffff)
            fprintf(TraceFile, " bit rate: variable\n");
        else
            fprintf(TraceFile, " bit rate: %.1f kbits/sec\n", 0.4 * bit_rate);
        fprintf(TraceFile, " VBV buffer size: %d Kbyte\n", 2 * vbv_buffer_size);
        fprintf(TraceFile, " constrained parameter stream: %s\n",
            constrained_parameters_flag ? "yes" : "no");
        fprintf(TraceFile, " load intra quantizer matrix: %s\n",
            load_intra_quantizer_matrix ? "yes" : "no");
        if (load_intra_quantizer_matrix && (TraceLevel >= 2))
            for (index = 0; index < 64; index++)
                fprintf(TraceFile,"%3d%c", tQuantIntra[index],
                    ((index % 8) == 7) ? '\n' : ' ');
        fprintf(TraceFile, " load non intra quantizer matrix: %s\n",
            load_non_intra_quantizer_matrix ? "yes" : "no");
        if (load_non_intra_quantizer_matrix && (TraceLevel >= 2))
            for (index = 0; index < 64; index++)
                fprintf(TraceFile,"%3d%c", tQuant[index],
                    ((index % 8) == 7) ? '\n' : ' ');
    }
```

```
    Next_Start_Code();

    Extension_And_User_Data();
}

void    Extension_And_User_Data()
{
    unsigned int buf;

    Next_Bits(&buf, bStartCode);
    if (buf == cExtensionStartCode)
    {
        /* get extension data */
        Get_Bits(&buf, bStartCode);
        if (TraceLevel>=1)
            fprintf(TraceFile, "Extension Data:\n");
        Next_Bits(&buf, 24);
        while (buf != 0x000001)
        {
            Get_Bits(&buf, 8);
            if (TraceLevel >= 2)
                fprintf(TraceFile, "%02x", buf);
            Next_Bits(&buf, 24);
        }

        if (TraceLevel >= 2)
            putc('\n', TraceFile);

        Next_Start_Code();
    }

    Next_Bits(&buf, bStartCode);
    if (buf == cUserDataStartCode)
    {
        /* get user data */
        Get_Bits(&buf, bStartCode);
        if (TraceLevel>=1)
            fprintf(TraceFile, "User Data:\n");
        Next_Bits(&buf, 24);
        while (buf != 0x000001)
        {
            Get_Bits(&buf, 8);
            if (TraceLevel >= 2)
                fprintf(TraceFile, "%02x", buf);
            Next_Bits(&buf, 24);
        }

        if (TraceLevel >= 2)
            putc('\n', TraceFile);

        Next_Start_Code();
    }
}
```

B.3.21 slice.c

```
/* slice.c */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
```

```
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   slice.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.2  93/06/15  15:41:31  oosa
 *
 *
 * Revision 1.1  1993/03/10  20:34:26  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "flc.h"
#include "vlc.h"
#include "global.h"
#include "decode.h"
#include "decode.pro"

int     Process_Slice(
    short           picture_coding_type,
    BYTE            *buf_Y,
    BYTE            *buf_Cb,
    BYTE            *buf_Cr,
    tData           *data,
    int                     forward_f_code,
    int                     fullPelForwardV,
    int                     backward_f_code,
    int                     fullPelBackwardV)
{
    unsigned int    buf;
    int     currentMBDiff[kMBLen];
    int     currentMBDiffU[kBLen];
    int     currentMBDiffV[kBLen];
    int     dct_dc_y_past, dct_dc_cb_past, dct_dc_cr_past;
    short   forwardMV[2];
    short   backwardMV[2];
    short   PMVFor[2];
    short   PMVBack[2];
    int     macroblock_type;
    int     bufOffsetY, bufOffsetC;
    int     MQuant, coded_block_pattern, MBA_incr= 0;
    int     index, row, code;
    int     slice_vertical_position, MBA;
    int     MB_row, MB_col;
    int     Skipped;

    /* remove slice start code */

    if (! Get_Bits(&buf, bStartCode))
    {
        printf ("Error (procSlice) -- No slice start code!\n");
        return(0);
    }
```

```
    slice_vertical_position = buf & 0x000000ff;
    MBA = (slice_vertical_position - 1) * mb_width - 1;

    /* get quantizer scale */
    MQuant = Get(bQuantizerScale);

    if (TraceLevel >= 2)
    {
        fprintf(TraceFile, "Slice Header: ");
        fprintf(TraceFile, " vertical position:%3d", slice_vertical_position);
        fprintf(TraceFile, " quantizer scale:%2d\n", MQuant);
    }

    /* skip extra_bit_slice & extra_information_slice */

    Get_Bits(&buf, 1);
    while(buf != 0)
    {
        Get_Bits(&buf, 8);
        Get_Bits(&buf, 1);
    }

    /* reset all predictors */
    dct_dc_y_past = dct_dc_cb_past = dct_dc_cr_past = 128;
    PMVFor[0] = PMVFor[1] = 0;
    PMVBack[0] = PMVBack[1] = 0;

    for (index = 0; ; index++)
    {

        if (MBA_incr == 0)
        {

            Next_Bits(&buf, 23);
            if (buf == 0)
            {
                Next_Start_Code();
                return(1);
            }

            /* get macroblock address increment */

            MBA_incr = Get_MBA_Inc();

            if (MBA_incr < 1)
            {
                printf ("Error (procSlice - MBA) - %d\n", index);
                return(0);
            }

            if (index == 0) /* first macroblock in slice */
            {
                MBA += MBA_incr; /* add horizontal offset */
                if (TraceLevel >= 2)
                    fprintf(TraceFile, "               horizontal offset:%3d\n",
                        MBA_incr - 1);
                MBA_incr = 1; /* don't skip first MB */
            }
        }

        Skipped = (MBA_incr != 1);

        MB_row = MBA / mb_width;
        MB_col = MBA % mb_width;

        bufOffsetY = MB_row * kMBHeight * (16 * mb_width) + MB_col * kMBWidth;
        bufOffsetC = MB_row * kBHeight  * (8 * mb_width) + MB_col * kBWidth;

        if (TraceLevel >= 3)
            fprintf(TraceFile,
                "  MB(%2d,%2d)", MB_row, MB_col);

        if (! Skipped)
        {

            /* get macroblock_type */
```

```
switch (picture_coding_type)
{
case kIPicture:
    code = Get_Code(MB_typeI_length, MB_typeI_value, 2);
    if (code >= 0)
        macroblock_type = MB_typeI_switch[code];
    break;
case kPPicture:
    code = Get_Code(MB_typeP_length, MB_typeP_value, 7);
    if (code >= 0)
        macroblock_type = MB_typeP_switch[code];
    break;
case kBPicture:
    code = Get_Code(MB_typeB_length, MB_typeB_value, 11);
    if (code >= 0)
        macroblock_type = MB_typeB_switch[code];
    break;
default:
    code = -1;
}

if (code == -1)
{
    printf ("Error (procSlice - macroblock_type) - %d\n", index);
    return(0);
}

if (macroblock_type & kMbQuant)
{
    /*      get quantizer_scale     */

    Get_Bits(&buf, 5);
    MQuant = buf;
}

/* reset motion vector predictors if MB is intra coded */

if (macroblock_type & kMbIntra)
{
    PMVFor[0] = 0;
    PMVFor[1] = 0;
    PMVBack[0] = 0;
    PMVBack[1] = 0;
}

/* get forward motion vector if present */

if (macroblock_type & kMbForward)
{
    if (! Get_MV(forward_f_code, fullPelForwardV,
            forwardMV, PMVFor))
    {
        printf ("Error (procSlice - Get_MV) - %d\n", index);
        return(0);
    }
    if (TraceLevel >= 3)
    {
        fprintf(TraceFile, " MVf(%3d,%3d)",
            forwardMV[0], forwardMV[1]);
    }
}
else if (TraceLevel >= 3)
    fprintf(TraceFile, "             ");

/* get backward motion vector if present */

if (macroblock_type & kMbBackward)
{
    if (! Get_MV(backward_f_code, fullPelBackwardV,
            backwardMV, PMVBack))
    {
        printf ("Error (procSlice - Get_MV) - %d\n", index);
        return(0);
    }
    if (TraceLevel >= 3)
```

```
            {
                fprintf(TraceFile, " MVb(%3d,%3d)",
                    backwardMV[0], backwardMV[1]);
            }
        }
        else if (TraceLevel >= 3)
            fprintf(TraceFile, "            ");

        if (TraceLevel >= 3)
        {
            fprintf(TraceFile, " q:%2d",MQuant);
            fprintf(TraceFile, ", type: ");
            if (macroblock_type & kMbIntra) putc('I', TraceFile);
            if (macroblock_type & kMbForward) putc('F', TraceFile);
            if (macroblock_type & kMbBackward) putc('B', TraceFile);
            if (macroblock_type & kMbPattern) putc('C', TraceFile);
            if (macroblock_type & kMbQuant) putc('Q', TraceFile);
            putc('\n', TraceFile);
        }

    }
    else if (TraceLevel >= 3)
            fprintf(TraceFile, "                                  type: Skipped\n");

    if ((!(macroblock_type & kMbIntra)) || Skipped)
    {
        /* MB is non-intra coded */

        /* reset DC predictors */
        dct_dc_y_past = 128;
        dct_dc_cb_past = 128;
        dct_dc_cr_past = 128;

        /*      reconstruct using MVs   */
        if (Skipped && (picture_coding_type == kPPicture))
            /* Skipped MBs in P picture don't have MVs */
            macroblock_type = 0;

        if ((macroblock_type & kMbForward) && (macroblock_type & kMbBackward))
        {
            /*      interpolated    */

            Construct_Bi_Pred(
                    data->firstPredictYR + bufOffsetY,
                    data->firstPredictUR + bufOffsetC,
                    data->firstPredictVR + bufOffsetC,
                    data->secondPredictYR + bufOffsetY,
                    data->secondPredictUR + bufOffsetC,
                    data->secondPredictVR + bufOffsetC,
                    buf_Y + bufOffsetY,
                    buf_Cb + bufOffsetC,
                    buf_Cr + bufOffsetC,
                    forwardMV, backwardMV);
        }
        else if (macroblock_type & kMbForward)
        {
            /*      forward */

            Construct_Mono_Pred(
                    data->firstPredictYR + bufOffsetY,
                    data->firstPredictUR + bufOffsetC,
                    data->firstPredictVR + bufOffsetC,
                    buf_Y + bufOffsetY,
                    buf_Cb + bufOffsetC,
                    buf_Cr + bufOffsetC, forwardMV);
        }
        else if (macroblock_type & kMbBackward)
        {
            /*      backward        */

            Construct_Mono_Pred(
                    data->secondPredictYR + bufOffsetY,
                    data->secondPredictUR + bufOffsetC,
                    data->secondPredictVR + bufOffsetC,
                    buf_Y + bufOffsetY,
                    buf_Cb + bufOffsetC,
```

```
                        buf_Cr + bufOffsetC, backwardMV);
            }
            else if (picture_coding_type == kPPicture)
            {
                /*      no MC - P Picture only  */

                PMVFor[0] = 0;
                PMVFor[1] = 0;

                forwardMV[0] = 0;
                forwardMV[1] = 0;

                Construct_Mono_Pred(data->firstPredictYR + bufOffsetY,
                        data->firstPredictUR + bufOffsetC,
                        data->firstPredictVR + bufOffsetC,
                        buf_Y + bufOffsetY,
                        buf_Cb + bufOffsetC,
                        buf_Cr + bufOffsetC, forwardMV);
            }
        }

        if (! Skipped)
        {
            /*      get coded block pattern */

            if (macroblock_type & kMbIntra)
                coded_block_pattern = 0x03F;          /*      all blocks      */
            else if (macroblock_type & kMbPattern)
            {
                if ((coded_block_pattern = Get_Code(CBP_length, CBP_value,
                        66)) == -1)
                {
                    printf ("Error (procSlice - coded_block_pattern) - %d\n",
                            index);
                    return(0);
                }
            }
            else
                coded_block_pattern = 0;

            /* clear DCT coefficient arrays */

            for (row = 0; row < kMBLen; row++)
                currentMBDiff[row] = 0;

            for (row = 0; row < kBLen; row++)
            {
                currentMBDiffU[row] = 0;
                currentMBDiffV[row] = 0;
            }

            if (! Process_MB(currentMBDiff, currentMBDiffU,
                    currentMBDiffV, &dct_dc_y_past, &dct_dc_cb_past,
                    &dct_dc_cr_past, macroblock_type, coded_block_pattern))
            {
                printf ("Error (procSlice) - after VLDing MB %d\n", index);
                return(0);
            }

            if (macroblock_type & kMbIntra)
                Inverse_Quantize_Intra_MB(coded_block_pattern,
                 currentMBDiff, currentMBDiffU, currentMBDiffV,
                 MQuant);
            else
                Inverse_Quantize_Non_Intra_MB(coded_block_pattern, currentMBDiff,
currentMBDiffU, currentMBDiffV, MQuant);

            Perform_IDCT(macroblock_type, currentMBDiff, currentMBDiffU, currentMBDiffV);

            /*      copy decoded data into picture buf      */

            Copy_From_Buf(macroblock_type, coded_block_pattern, MB_row, MB_col,
                    buf_Y, buf_Cb, buf_Cr,
                    currentMBDiff, currentMBDiffU, currentMBDiffV);

        }
```

```
        MBA_incr--;
        MBA++;
    }

    return(1);
}

int Get_MBA_Inc()
{
    int             code, MBA_incr;

    MBA_incr = 0;

    while (1)
    {
        if ((code = Get_Code(MBA_length, MBA_value, 35)) == -1)
            return(-1);

        if (code == 0)
            MBA_incr += 33;   /* macroblock_escape */
        else if (code != (35 - 1))
        {
            /* not macroblock_stuffing */
            MBA_incr += code;
            break;
        }
    }

    return(MBA_incr);
}

int     Get_MV(
    int     f_code,
    int     fullPelV,
    short   MV[2],
    short   PMV[])
{
    unsigned int    buf;
    int                             delta, sign, motionCodeMag, index;
    int                             f, low, high, range;

    f = 1 << (f_code - 1);
    low = -16*f;
    high = 16*f - 1;
    range = 32*f;

    /*      get diff vectors        */

    for (index = 0; index < 2; index++)
    {
        /*      get vlc_code_magnitude  */

        if ((motionCodeMag = Get_Code(MV_length, MV_value,
                33)) == -1)
        {
            printf ("Error (Get_Code)\n");
            return(0);
        }

        if (motionCodeMag > 16)
        {
            sign = -1;                      /*      negative delta  */
            motionCodeMag -= 16;
        }
        else
            sign = 1;

        /*      get residual    */
        if ((f_code != 1) && (motionCodeMag != 0))
            Get_Bits(&buf, f_code - 1);
        else
            buf = 0;

        if (motionCodeMag == 0)
            delta = 0;
```

```
        else
            delta = (motionCodeMag - 1) * f + buf + 1;

        MV[index] = PMV[index] + sign * delta;

        /*      adjust out of range MVs */

        if (MV[index] < low)
            MV[index] += range;
        else if (MV[index] > high)
            MV[index] -= range;

        PMV[index] = MV[index];
        if (fullPelV)
            MV[index] <<= 1;
    }

    return(1);
}
```

B.3.22 transfer.c

```
/* transfer.c  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   transfer.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:56  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include "consts.h"
#include "types.h"
#include "decode.h"
#include "decode.pro"
```

```
void    Copy_From_Buf(
    int     macroblock_type,
    short   coded_block_pattern,
    int     MB_row,
    int     MB_col,
    BYTE    *buf_Y_recon,
    BYTE    *buf_Cb_recon,
    BYTE    *buf_Cr_recon,
    int     currentMBDiff[],
    int     currentMBDiffU[],
    int     currentMBDiffV[])
{
    int     bufIndex, bufIndex1, currentIndex, temp;
    short   code, line, pix, row, col;

    if (macroblock_type & kMbIntra)  /*      intra   */
    {
        bufIndex = MB_row * kMBHeight * (16 * mb_width) + MB_col * kMBWidth;

        /*      lumin   */

        for (row = 0, currentIndex = 0; row < kMBHeight; row++,
                bufIndex += 16 * mb_width - kMBWidth)
        {
            for (col = 0; col < kMBWidth; col++, currentIndex++, bufIndex++)
            {
                buf_Y_recon[bufIndex] = (currentMBDiff[currentIndex] >= 255) ? 255 :
                        ((currentMBDiff[currentIndex] <= 0) ? 0 :
                        currentMBDiff[currentIndex]);
            }
        }

        /*      chrom   */

        bufIndex = MB_row * kBHeight * (8 * mb_width) + MB_col * kBWidth;

        for (row = 0, currentIndex = 0; row < kBHeight; row++,
                bufIndex += 8 * mb_width - kBWidth)
        {
            for (col = 0; col < kBWidth; col++, currentIndex++, bufIndex++)
            {
                buf_Cb_recon[bufIndex] = (currentMBDiffU[currentIndex] >= 255) ? 255 :
                        ((currentMBDiffU[currentIndex] <= 0) ? 0 :
                        currentMBDiffU[currentIndex]);
                buf_Cr_recon[bufIndex] = (currentMBDiffV[currentIndex] >= 255) ? 255 :
                        ((currentMBDiffV[currentIndex] <= 0) ? 0 :
                        currentMBDiffV[currentIndex]);
            }
        }
    }
    else    /*      inter   */
    {
        /*      lumin   */

        code = 0x20;

        for (line = 0; line < kMBHeight; line += kBHeight)
        {
            bufIndex1 = (MB_row * kMBHeight + line) * (16 * mb_width) +
                    MB_col * kMBWidth;

            for (pix = 0; pix < kMBWidth; pix += kBWidth)
            {
                if (! (coded_block_pattern & code))
                {
                    code >>= 1;
                    continue;
                }

                bufIndex = bufIndex1 + pix;

                code >>= 1;

                currentIndex = line * kMBWidth + pix;
```

```
                for (row = 0; row < kBHeight;
                        row++, currentIndex += kMBWidth - kBWidth,
                        bufIndex += 16 * mb_width - kBWidth)
                {
                    for (col = 0; col < kBWidth; col++, currentIndex++,
                            bufIndex++)
                    {
                        temp = buf_Y_recon[bufIndex] + currentMBDiff[currentIndex];
                        buf_Y_recon[bufIndex] =
                                (temp >= 255) ? 255 : ((temp <= 0) ? 0 : temp);
                    }
                }
            }
        }

        /*      chrom - U       */

        if (coded_block_pattern & 0x02)
        {
            bufIndex = MB_row * kBHeight * (8 * mb_width) + MB_col * kBWidth;

            for (row = 0, currentIndex = 0; row < kBHeight; row++,
                    bufIndex += 8 * mb_width - kBWidth)
            {
                for (col = 0; col < kBWidth; col++, currentIndex++, bufIndex++)
                {
                    temp = buf_Cb_recon[bufIndex] + currentMBDiffU[currentIndex];
                    buf_Cb_recon[bufIndex] =
                            (temp >= 255) ? 255 : ((temp <= 0) ? 0 : temp);
                }
            }
        }

        /*      chrom - V       */

        if (coded_block_pattern & 0x01)
        {
            bufIndex = MB_row * kBHeight * (8 * mb_width) + MB_col * kBWidth;

            for (row = 0, currentIndex = 0; row < kBHeight; row++,
                    bufIndex += 8 * mb_width - kBWidth)
            {
                for (col = 0; col < kBWidth; col++, currentIndex++, bufIndex++)
                {
                    temp = buf_Cr_recon[bufIndex] + currentMBDiffV[currentIndex];
                    buf_Cr_recon[bufIndex] =
                            (temp >= 255) ? 255 : ((temp <= 0) ? 0 : temp);
                }
            }
        }
    }
}
```

B.3.23 types.h

```
/* File: types.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
```

```
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *  $Log:   types.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.4  93/09/03  1:15:00  hana
 * VBV buffer operation
 *
 * Revision 1.1.1.1  93/03/29  11:27:53  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

typedef struct t_Data        /*  picture data */
{
    /*  working pointers */

    BYTE *firstPredictYR;    /*  reconstructed 1st prediction picture      */
    BYTE *firstPredictUR;    /*  reconstructed 1st prediction picture      */
    BYTE *firstPredictVR;    /*  reconstructed 1st prediction picture      */
    BYTE *secondPredictYR;   /*  reconstructed 2nd prediction picture      */
    BYTE *secondPredictUR;   /*  reconstructed 2nd prediction picture      */
    BYTE *secondPredictVR;   /*  reconstructed 2nd prediction picture      */
    BYTE *predictYR;     /*  reconstructed B picture */
    BYTE *predictUR;     /*  reconstructed B picture */
    BYTE *predictVR;     /*  reconstructed B picture */

    /*  storage arrays   */

    BYTE *luminPicture0R;    /*  reconstructed luminance of picture 0      */
    BYTE *luminPicture1R;    /*  reconstructed luminance of picture 1      */
    BYTE *luminPicture2R;    /*  reconstructed luminance of picture 2      */
    BYTE *chromUPicture0R;   /*  reconstructed chrominance of picture 0    */
    BYTE *chromUPicture1R;   /*  reconstructed chrominance of picture 1    */
    BYTE *chromUPicture2R;   /*  reconstructed chrominance of picture 2    */
    BYTE *chromVPicture0R;
    BYTE *chromVPicture1R;
    BYTE *chromVPicture2R;
} tData;

typedef struct t_Parameter       /*  user parameter */
{
    char inputSequence[kMaxCharBufLen],
            outputSequence[kMaxCharBufLen],
            statusFName[kMaxCharBufLen],
            ACVlc_FName[kMaxCharBufLen],
            bitStreamFName[kMaxCharBufLen];
    int     outputFormat;
} tParameter;
```

B.3.24 vlc.h

```
/* vlc.h */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
```

```
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   vlc.h,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:52  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:32:40  au
 * Initial revision
 *
 */

#define CBP_table_entries       64
#define DC_Y_table_entries      9
#define DC_C_table_entries      9
#define MBA_table_entries       35
#define MB_typeI_table_entries  2
#define MB_typeP_table_entries  7
#define MB_typeB_table_entries  11
#define MV_table_entries        33
#define AC_table_entries        112

#define AC_escape_value         1
#define AC_escape_length        6
#define AC_first_length         2       /*  dct_coeff_first bit count */
#define AC_first_value          2       /*  dct_coeff_first pattern */
```

B.3.25 writepic.c

```
/* writepic.c for version 1.5a of ISO MPEG-1 decoder  */
/* Copyright (C) 1994,ISO/IEC JTC1 SC29 WG11. All Rights Reserved. */
/*
By Peter Au (Hughes Aircraft)
Stefan Eckart (Technical University of Munich)
Chad Fogg (Cascade Design Automation)
Kinya Oosa (Nippon Steel)
Tsuyoshi Hanamura (Waseda University)
Hiroshi Watanabe (NTT).
 * Disclaimer of Warranty
 *
```

```
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  TheISO/IEC JTC1 SC29 WG1 disclaims
 * any and all warranties, whether express, implied, or statuary, including any
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * TheISO/IEC JTC1 SC29 WG1 does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 */

/*
 *      $Log:   writepic.c,v $
 * Revision 2.0  94/05/16  00:00:00  cfogg
 * Release version (publication ISO/IEC CD 11172-5)
 *
 * Revision 1.5  93/12/07  12:00:00  sE
 *
 * Revision 1.1.1.1  93/03/29  11:27:54  oosa
 * MPEG1 encoder/decoder initial revision
 *
 * Revision 1.1  1993/03/10  20:33:17  au
 * Initial revision
 *
 */

#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include "consts.h"
#include "types.h"
#include "decode.h"
#include "decode.pro"


int     Write_Picture(
    int                     aPictureNum,
    BYTE            *y,
    BYTE            *u,
    BYTE            *v,
    tParameter      *aParameter)
{
    int             ret;

    switch (aParameter->outputFormat)
    {
    case 0:
        ret = Write_SIF(aPictureNum, y, u, v, aParameter);
        break;
    case 1:
        ret = Write_TGA(aPictureNum, y, u, v, aParameter);
        break;
    case 2:
        ret = Write_PPM(aPictureNum, y, u, v, aParameter);
        break;
        case 3:
        ret = Write_YUV(aPictureNum, y, u, v, aParameter);
        break;
    }

    return(ret);
}


int     Write_SIF(
```

```
    int             aPictureNum,
    BYTE            *luminPictureR,
    BYTE            *chromUPictureR,
    BYTE            *chromVPictureR,
    tParameter      *aParameter)
{
    register        i,
                j;

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *buffer;
    BYTE    *b;
    BYTE    *y;
    BYTE    *u;
    BYTE    *v;
    BYTE    *tmpuR;
    BYTE    *tmpvR;
    int             numChromPixels, luminWidthHalf, luminWidth2;

    numChromPixels = (8 * mb_width) * (8 * mb_height);
    luminWidthHalf = horizontal_size / 2;
    luminWidth2 = horizontal_size * 2;
    tmpuR = (BYTE *) malloc(numChromPixels * sizeof(BYTE));
    tmpvR = (BYTE *) malloc(numChromPixels * sizeof(BYTE));
    buffer = (BYTE *) malloc(luminWidth2 * sizeof(BYTE));

    y = luminPictureR;
    u = chromUPictureR;
    v = chromVPictureR;

    for (i = 0; i < numChromPixels; i++)
    {
        tmpuR[i] = u[i];
        tmpvR[i] = v[i];
    }

    /* interpolate chrominance in Y direction */

    convert420to422(u, 8 * mb_width, 16* mb_height);
    convert420to422(v, 8 * mb_width, 16* mb_height);

    sprintf (fname, "%s%03d.SIF", aParameter->outputSequence, aPictureNum);

    printf("%s\n",fname);

    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (1);
    }

    /* store picture */

    y = luminPictureR;
    u = chromUPictureR;
    v = chromVPictureR;

    for (i = 0; i < vertical_size; i++)
    {
        for (j = 0, b = buffer; j < luminWidthHalf; j++)
        {
            *(b++) = *(u++);
            *(b++) = *(y++);
            *(b++) = *(v++);
            *(b++) = *(y++);
        }

        fwrite(buffer, sizeof(BYTE), luminWidth2, fptr);
        y += 16 * mb_width - 2 * luminWidthHalf;
        u += 8 * mb_width - luminWidthHalf;
        v += 8 * mb_width - luminWidthHalf;
    }

    fclose(fptr);
```

```
    u = chromUPictureR;
    v = chromVPictureR;

    for (i = 0; i < numChromPixels; i++)
    {
        u[i] = tmpuR[i];
        v[i] = tmpvR[i];
    }

    free(tmpuR);
    free(tmpvR);
    free(buffer);

    return (0);
}

int     Write_TGA(
    int                    aPictureNum,
    BYTE            *y,
    BYTE            *u,
    BYTE            *v,
    tParameter      *aParameter)
{
    int             i, j;
    static unsigned char    tga24[14]={0,0,2,0,0,0,0, 0,0,0,0,0,24,32};

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *u2;
    BYTE    *v2;
    int             r, g, b;

    u2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));
    v2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));

    /* interpolate chrominance in both directions */

    convert420to444(u, u2, 8 * mb_width, 8 * mb_height);
    convert420to444(v, v2, 8 * mb_width, 8 * mb_height);

    sprintf (fname, "%s%03d.tga", aParameter->outputSequence, aPictureNum);

    printf("%s\n",fname);

    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (1);
    }

    /* write Targa header */
    for (i = 0; i < 12; i++)
        putc(tga24[i], fptr);

    putc(horizontal_size, fptr);
    putc(horizontal_size >> 8, fptr);
    putc(vertical_size, fptr);
    putc(vertical_size >> 8,fptr);
    putc(tga24[12], fptr);
    putc(tga24[13], fptr);

    /* store picture */

    for (j = 0; j < vertical_size; j++)
    {
        for (i = 0; i < horizontal_size; i++)
        {

            yuvtorgb(*y++, *u2++, *v2++, &r, &g, &b);

            putc(b, fptr);
            putc(g, fptr);
            putc(r, fptr);
        }

        y  += 16 * mb_width - horizontal_size;
```

```
        u2 += 16 * mb_width - horizontal_size;
        v2 += 16 * mb_width - horizontal_size;
    }

    fclose(fptr);

    free(u2);
    free(v2);

    return (0);
}

int     Write_PPM(
    int                     aPictureNum,
    BYTE            *y,
    BYTE            *u,
    BYTE            *v,
    tParameter      *aParameter)
{
    int             i, j;

    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *u2;
    BYTE    *v2;
    int             r, g, b;

    u2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));
    v2 = (BYTE *) malloc((16*mb_width) * (16*mb_height) * sizeof(BYTE));

    /* interpolate chrominance in both directions */

    convert420to444(u, u2, 8 * mb_width, 8 * mb_height);
    convert420to444(v, v2, 8 * mb_width, 8 * mb_height);

    sprintf (fname, "%s%03d.ppm", aParameter->outputSequence, aPictureNum);

    printf("%s\n",fname);

    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (1);
    }

    /* write PPM header */
    fprintf(fptr,"P6\n%d %d\n255\n",horizontal_size,vertical_size);

    /* store picture */

    for (j = 0; j < vertical_size; j++)
    {
        for (i = 0; i < horizontal_size; i++)
        {

            yuvtorgb(*y++, *u2++, *v2++, &r, &g, &b);

            putc(r, fptr);
            putc(g, fptr);
            putc(b, fptr);
        }

        y  += 16 * mb_width - horizontal_size;
        u2 += 16 * mb_width - horizontal_size;
        v2 += 16 * mb_width - horizontal_size;
    }

    fclose(fptr);

    free(u2);
    free(v2);

    return (0);
}

void    yuvtorgb(
```

```
    int             y,
    int             u,
    int             v,
    int             *pr,
    int             *pg,
    int             *pb)
{
    int             r, g, b;
    double  yf, uf, vf, rmy, gmy, bmy, rf, gf, bf;

    /* scale to 0..1 (Y) / -1..+1 (U,V) */

    yf = (y - 16) / (double)(235-16);
    uf = (u - 128) / (double)(240-16);
    vf = (v - 128) / (double)(240-16);

    /* compute color differences R-Y, G-Y, B-Y */

    rmy = ((1.0-0.299)/0.5) * vf; /* 1.402*vf */
    bmy = ((1.0-0.114)/0.5) * uf; /* 1.772*uf */
    gmy = -(0.299*rmy + 0.114*bmy)/0.587;

    /* compute color primaries R, G, B */

    rf = yf + rmy;
    bf = yf + bmy;
    gf = yf + gmy;

    /* scale to 0..255 */

    r = floor(256.0 * rf + 0.5);
    if (r < 0)
        r = 0;
    if (r > 255)
        r = 255;
    g = floor(256.0 * gf + 0.5);
    if (g < 0)
        g = 0;
    if (g > 255)
        g = 255;
    b = floor(256.0 * bf + 0.5);
    if (b < 0)
        b = 0;
    if (b > 255)
        b = 255;

    *pr = r;
    *pg = g;
    *pb = b;

}

int     Write_YUV(
    int             pictureNum,
    BYTE            *y_recon,
    BYTE            *cb_recon,
    BYTE            *cr_recon,
    tParameter      *parameter)
{
    FILE    *fptr;
    char    fname[kMaxCharBufLen];
    BYTE    *buffer;
    BYTE    *b;
    BYTE    *y;
    BYTE    *u;
    BYTE    *v;
    register line, col;
    int     lw, lh, cw, ch;

    lw = horizontal_size;
    lh = vertical_size;
    cw = lw >> 1;   /* chroma width is always half luma for 4:2:0 */
    ch = lh >> 1;   /* ditto for height */

    /* allocate "line buffer" of greatest common multiple dimensions */
    buffer = (BYTE *) malloc(lw * sizeof(BYTE));
```

```
    sprintf (fname, "%s%03d.yuv", parameter->outputSequence, pictureNum);
    printf("%s\n",fname);

    if((fptr = fopen(fname, "wb")) == NULL)
    {
        printf("\nopen error filename: %s\n", fname);
        return (-1);
    }

    /* store picture */

    y = y_recon;

    for (line = 0; line <  lh; line++)
    {
        for (col = 0, b = buffer; col < lw; col++)
        {
            *(b++) = *(y++);
        }
        fwrite(buffer, sizeof(BYTE), lw, fptr);
        y += (16 * mb_width) - lw;

    }

    u = cb_recon;

    for (line = 0; line < ch; line++)
    {
        for (col = 0, b = buffer; col < cw; col++)
        {
            *(b++) = *(u++);
        }

        fwrite(buffer, sizeof(BYTE), cw, fptr);
        u += (8 * mb_width) - cw;

    }


    v = cr_recon;

    for (line = 0; line < ch; line++)
    {
        for (col = 0, b = buffer; col < cw; col++)
        {
            *(b++) = *(v++);
        }

        fwrite(buffer, sizeof(BYTE), cw, fptr);
        v += (8 * mb_width) - cw;

    }

    free(buffer);
    fclose(fptr);
    return (0);
}
```

Annex C
(informative)

Audio - code listings

C.1 Introduction

This Annex lists the C source code and associated data tables for the CD 11172-5 Audio codec.

Table C.1 List of audio files

filename	description
1cb0, 1cb1, 1cb2, 2cb1, 2cb2	Critical bound tables
1th0, 1th2, 2th0, 2th2	Global masking threshold tables
absthr_0 , absthr_1, absthr_2	Absolute threshold tables
alloc_0, alloc_1, alloc_2, alloc_3	Spectral bit allocation tables
enwindow	encoder subband synthesis filter bank coefficient data table
dewindow	decoder subband synthesis filter bank coefficient data table
huffdec	layer 3 Huffman coding data tables
common.c	global variable functions and definitions, read AIFF routine, CRC, and low-level I/O routines
common.h	global conditional compiler switches and defualts
decode.c	core decoder routines
decoder.h	external variables and prototypes used in decoding
encode.c	core of encoder program except psychoacoustic models.
encoder.h	encoder external variables and constants
huffman.h	definitions for layer 3 decoder Huffman coding routines
huffman.c	layer 3 decoder Huffman coding routines
musicin.c	prompts for encoder user input parameters and performs initlization..
musicout.c	decoder parameter initilization
psy.c	psychoaucostic model routines
subs.c	FFT subroutine
tonal.c	model I for layers I and II

C.2 Tables

C.2.1 1cb0

```
25
0 1
1 2
2 3
3 5
4 6
5 8
6 9
7 11
8 13
9 15
10 17
11 20
12 23
13 27
14 32
15 37
16 45
17 52
18 62
19 74
20 88
21 108
22 132
23 180
24 232
```

C.2.2 1cb1

```
26
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 9
8 10
9 12
10 14
11 16
12 19
13 21
14 25
15 29
16 35
17 41
18 50
19 58
20 68
21 82
22 100
23 124
24 164
25 216
```

C.2.3 1cb2

```
24
0 1
1 3
2 5
3 7
4 9
5 11
6 13
7 15
8 18
9 21
10 24
11 27
12 32
13 37
14 44
15 52
16 62
17 74
18 88
19 104
20 124
21 148
22 184
23 240
```

C.2.4 2cb0

```
27
0 1
1 2
2 3
3 5
4 7
5 10
6 13
7 16
8 19
9 22
10 26
11 30
12 35
13 40
14 46
15 54
16 64
17 76
18 90
19 104
20 124
21 148
22 176
23 216
24 264
25 360
26 464
```

C.2.5 2cb1

```
27
0 1
1 2
2 3
3 5
4 7
5 9
6 12
7 14
8 17
9 20
10 24
11 27
12 32
13 37
14 42
15 50
16 58
17 70
18 82
19 100
20 116
21 136
22 164
23 200
24 248
25 328
26 432
```

C.2.6 2cb2

```
25
0 1
1 3
2 6
3 10
4 13
5 17
6 21
7 25
8 30
9 35
10 41
11 47
12 54
13 64
14 74
15 88
16 104
17 124
18 148
19 176
20 208
21 248
22 296
23 368
24 480
```

C.2.7 1th0

```
107
1 1 0.850 25.87
2 2 1.694 14.85
3 3 2.525 10.72
4 4 3.337 8.50
5 5 4.124 7.10
6 6 4.882 6.11
7 7 5.608 5.37
8 8 6.301 4.79
9 9 6.959 4.32
10 10 7.581 3.92
11 11 8.169 3.57
12 12 8.723 3.25
13 13 9.244 2.95
14 14 9.734 2.67
15 15 10.195 2.39
16 16 10.629 2.11
17 17 11.037 1.83
18 18 11.421 1.53
19 19 11.783 1.23
20 20 12.125 0.90
21 21 12.448 0.56
22 22 12.753 0.21
23 23 13.042 -0.17
```

```
24 24 13.317 -0.56
25 25 13.578 -0.96
26 26 13.826 -1.38
27 27 14.062 -1.79
28 28 14.288 -2.21
29 29 14.504 -2.63
30 30 14.711 -3.03
31 31 14.909 -3.41
32 32 15.100 -3.77
33 33 15.284 -4.09
34 34 15.460 -4.37
35 35 15.631 -4.60
36 36 15.796 -4.78
37 37 15.955 -4.91
38 38 16.110 -4.97
39 39 16.260 -4.98
40 40 16.406 -4.92
41 41 16.547 -4.81
42 42 16.685 -4.65
43 43 16.820 -4.43
44 44 16.951 -4.17
45 45 17.079 -3.87
46 46 17.205 -3.54
47 47 17.327 -3.19
48 48 17.447 -2.82
49 50 17.680 -2.06
50 52 17.905 -1.32
51 54 18.121 -0.64
52 56 18.331 -0.04
53 58 18.534 0.47
54 60 18.731 0.89
55 62 18.922 1.23
56 64 19.108 1.51
57 66 19.289 1.74
58 68 19.464 1.93
59 70 19.635 2.11
60 72 19.801 2.28
61 74 19.963 2.46
62 76 20.120 2.63
63 78 20.273 2.82
64 80 20.421 3.03
65 82 20.565 3.25
66 84 20.705 3.49
67 86 20.840 3.74
68 88 20.972 4.02
69 90 21.099 4.32
70 92 21.222 4.64
71 94 21.342 4.98
72 96 21.457 5.35
73 100 21.677 6.15
74 104 21.882 7.07
75 108 22.074 8.10
76 112 22.253 9.25
77 116 22.420 10.54
78 120 22.576 11.97
79 124 22.721 13.56
80 128 22.857 15.31
81 132 22.984 17.23
82 136 23.102 19.34
83 140 23.213 21.64
84 144 23.317 24.15
85 148 23.415 26.88
86 152 23.506 29.84
87 156 23.592 33.05
88 160 23.673 36.52
89 164 23.749 40.25
90 168 23.821 44.27
91 172 23.888 48.59
92 176 23.952 53.22
93 180 24.013 58.18
94 184 24.070 63.49
95 188 24.125 68.00
96 192 24.176 68.00
97 196 24.225 68.00
98 200 24.271 68.00
99 204 24.316 68.00
100 208 24.358 68.00
101 212 24.398 68.00
102 216 24.436 68.00
103 220 24.473 68.00
104 224 24.508 68.00
105 228 24.542 68.00
106 232 24.574 68.00
```

C.2.8 1th1

```
103
1 1 0.925 24.17
2 2 1.842 13.87
3 3 2.742 10.01
4 4 3.618 7.94
5 5 4.463 6.62
6 6 5.272 5.70
7 7 6.041 5.00
8 8 6.770 4.45
9 9 7.457 4.00
10 10 8.103 3.61
11 11 8.708 3.26
12 12 9.275 2.93
13 13 9.805 2.63
14 14 10.301 2.32
15 15 10.765 2.02
16 16 11.199 1.71
17 17 11.606 1.38
18 18 11.988 1.04
19 19 12.347 0.67
20 20 12.684 0.29
21 21 13.002 -0.11
22 22 13.302 -0.54
23 23 13.586 -0.97
24 24 13.855 -1.43
25 25 14.111 -1.88
26 26 14.354 -2.34
27 27 14.585 -2.79
28 28 14.807 -3.22
29 29 15.018 -3.62
30 30 15.221 -3.98
31 31 15.415 -4.30
32 32 15.602 -4.57
33 33 15.783 -4.77
34 34 15.956 -4.91
35 35 16.124 -4.98
36 36 16.287 -4.97
37 37 16.445 -4.90
38 38 16.598 -4.76
39 39 16.746 -4.55
40 40 16.891 -4.29
41 41 17.032 -3.99
42 42 17.169 -3.64
43 43 17.303 -3.26
44 44 17.434 -2.86
45 45 17.563 -2.45
46 46 17.688 -2.04
47 47 17.811 -1.63
48 48 17.932 -1.24
49 50 18.166 -0.51
50 52 18.392 0.12
51 54 18.611 0.64
52 56 18.823 1.06
53 58 19.028 1.39
54 60 19.226 1.66
55 62 19.419 1.88
56 64 19.606 2.08
57 66 19.788 2.27
58 68 19.964 2.46
59 70 20.135 2.65
60 72 20.300 2.86
61 74 20.461 3.09
62 76 20.616 3.33
63 78 20.766 3.60
64 80 20.912 3.89
65 82 21.052 4.20
66 84 21.188 4.54
67 86 21.318 4.91
68 88 21.445 5.31
69 90 21.567 5.73
70 92 21.684 6.18
71 94 21.797 6.67
72 96 21.906 7.19
73 100 22.113 8.33
74 104 22.304 9.63
75 108 22.482 11.08
76 112 22.646 12.71
77 116 22.799 14.53
78 120 22.941 16.54
79 124 23.072 18.77
80 128 23.195 21.23
81 132 23.309 23.94
82 136 23.415 26.90
83 140 23.515 30.14
84 144 23.607 33.67
85 148 23.694 37.51
86 152 23.775 41.67
87 156 23.852 46.17
88 160 23.923 51.04
89 164 23.991 56.29
90 168 24.054 61.94
91 172 24.114 68.00
92 176 24.171 68.00
93 180 24.224 68.00
94 184 24.275 68.00
95 188 24.322 68.00
96 192 24.368 68.00
97 196 24.411 68.00
98 200 24.452 68.00
99 204 24.491 68.00
100 208 24.528 68.00
101 212 24.564 68.00
102 216 24.597 68.00
```

C.2.9 1th2

```
109
1 1 0.617 33.44
2 2 1.232 19.20
3 3 1.842 13.87
4 4 2.445 11.01
5 5 3.037 9.20
6 6 3.618 7.94
7 7 4.185 7.00
8 8 4.736 6.28
9 9 5.272 5.70
10 10 5.789 5.21
11 11 6.289 4.80
12 12 6.770 4.45
13 13 7.233 4.14
14 14 7.677 3.86
15 15 8.103 3.61
16 16 8.511 3.37
17 17 8.901 3.15
18 18 9.275 2.93
19 19 9.632 2.73
20 20 9.974 2.53
21 21 10.301 2.32
22 22 10.614 2.12
23 23 10.913 1.92
24 24 11.199 1.71
25 25 11.474 1.49
26 26 11.736 1.27
27 27 11.988 1.04
28 28 12.230 0.80
29 29 12.461 0.55
30 30 12.684 0.29
31 31 12.898 0.02
32 32 13.104 -0.25
33 33 13.302 -0.54
```

```
34 34 13.493 -0.83
35 35 13.678 -1.12
36 36 13.855 -1.43
37 37 14.027 -1.73
38 38 14.193 -2.04
39 39 14.354 -2.34
40 40 14.509 -2.64
41 41 14.660 -2.93
42 42 14.807 -3.22
43 43 14.949 -3.49
44 44 15.087 -3.74
45 45 15.221 -3.98
46 46 15.351 -4.20
47 47 15.478 -4.40
48 48 15.602 -4.57
49 50 15.841 -4.82
50 52 16.069 -4.96
51 54 16.287 -4.97
52 56 16.496 -4.86
53 58 16.697 -4.63
54 60 16.891 -4.29
55 62 17.078 -3.87
56 64 17.259 -3.39
57 66 17.434 -2.86
58 68 17.605 -2.31
59 70 17.770 -1.77
60 72 17.932 -1.24
61 74 18.089 -0.74
62 76 18.242 -0.29
63 78 18.392 0.12
64 80 18.539 0.48
65 82 18.682 0.79
66 84 18.823 1.06
67 86 18.960 1.29
68 88 19.095 1.49
69 90 19.226 1.66
70 92 19.356 1.81
71 94 19.482 1.95
72 96 19.606 2.08
73 100 19.847 2.33
74 104 20.079 2.59
75 108 20.300 2.86
76 112 20.513 3.17
77 116 20.717 3.51
78 120 20.912 3.89
79 124 21.098 4.31
80 128 21.275 4.79
81 132 21.445 5.31
82 136 21.606 5.88
83 140 21.760 6.50
84 144 21.906 7.19
85 148 22.046 7.93
86 152 22.178 8.75
87 156 22.304 9.63
88 160 22.424 10.58
89 164 22.538 11.60
90 168 22.646 12.71
91 172 22.749 13.90
92 176 22.847 15.18
93 180 22.941 16.54
94 184 23.030 18.01
95 188 23.114 19.57
96 192 23.195 21.23
97 196 23.272 23.01
98 200 23.345 24.90
99 204 23.415 26.90
100 208 23.482 29.03
101 212 23.546 31.28
102 216 23.607 33.67
103 220 23.666 36.19
104 224 23.722 38.86
105 228 23.775 41.67
106 232 23.827 44.63
107 236 23.876 47.76
108 240 23.923 51.04
```

C.2.10 2th0

```
131
1 1 0.425 45.05
2 2 0.850 25.87
3 3 1.273 18.70
4 4 1.694 14.85
5 5 2.112 12.41
6 6 2.525 10.72
7 7 2.934 9.47
8 8 3.337 8.50
9 9 3.733 7.73
10 10 4.124 7.10
11 11 4.507 6.56
12 12 4.882 6.11
13 13 5.249 5.72
14 14 5.608 5.37
15 15 5.959 5.07
16 16 6.301 4.79
17 17 6.634 4.55
18 18 6.959 4.32
19 19 7.274 4.11
20 20 7.581 3.92
21 21 7.879 3.74
22 22 8.169 3.57
23 23 8.450 3.40
24 24 8.723 3.25
25 25 8.987 3.10
26 26 9.244 2.95
27 27 9.493 2.81
28 28 9.734 2.67
29 29 9.968 2.53
30 30 10.195 2.39
31 31 10.416 2.25
32 32 10.629 2.11
33 33 10.836 1.97
34 34 11.037 1.83
35 35 11.232 1.68
36 36 11.421 1.53
37 37 11.605 1.38
38 38 11.783 1.23
39 39 11.957 1.07
40 40 12.125 0.90
41 41 12.289 0.74
42 42 12.448 0.56
43 43 12.603 0.39
44 44 12.753 0.21
45 45 12.900 0.02
46 46 13.042 -0.17
47 47 13.181 -0.36
48 48 13.317 -0.56
49 50 13.577 -0.96
50 52 13.826 -1.38
51 54 14.062 -1.79
52 56 14.288 -2.21
53 58 14.504 -2.63
54 60 14.711 -3.03
55 62 14.909 -3.41
56 64 15.100 -3.77
57 66 15.284 -4.09
58 68 15.460 -4.37
59 70 15.631 -4.60
60 72 15.796 -4.78
61 74 15.955 -4.91
62 76 16.110 -4.97
63 78 16.260 -4.98
64 80 16.406 -4.92
65 82 16.547 -4.81
66 84 16.685 -4.65
67 86 16.820 -4.43
68 88 16.951 -4.17
69 90 17.079 -3.87
70 92 17.205 -3.54
71 94 17.327 -3.19
72 96 17.447 -2.82
73 100 17.680 -2.06
74 104 17.905 -1.32
75 108 18.121 -0.64
76 112 18.331 -0.04
77 116 18.534 0.47
78 120 18.731 0.89
79 124 18.922 1.23
80 128 19.108 1.51
81 132 19.289 1.74
82 136 19.464 1.93
83 140 19.635 2.11
84 144 19.801 2.28
85 148 19.963 2.46
86 152 20.120 2.63
87 156 20.273 2.82
88 160 20.421 3.03
89 164 20.565 3.25
90 168 20.705 3.49
91 172 20.840 3.74
92 176 20.972 4.02
93 180 21.099 4.32
94 184 21.222 4.64
95 188 21.342 4.98
96 192 21.457 5.35
97 200 21.677 6.15
98 208 21.882 7.07
99 216 22.074 8.10
100 224 22.253 9.25
101 232 22.420 10.54
102 240 22.576 11.97
103 248 22.721 13.56
104 256 22.857 15.31
105 264 22.984 17.23
106 272 23.102 19.34
107 280 23.213 21.64
108 288 23.317 24.15
109 296 23.415 26.88
110 304 23.506 29.84
111 312 23.592 33.05
112 320 23.673 36.52
113 328 23.749 40.25
114 336 23.821 44.27
115 344 23.888 48.59
116 352 23.952 53.22
117 360 24.013 58.18
118 368 24.070 63.49
119 376 24.125 68.00
120 384 24.176 68.00
121 392 24.225 68.00
122 400 24.271 68.00
123 408 24.316 68.00
124 416 24.358 68.00
125 424 24.398 68.00
126 432 24.436 68.00
127 440 24.473 68.00
128 448 24.508 68.00
129 456 24.542 68.00
130 464 24.574 68.00
```

C.2.11 2th1

```
127
1 1 0.463 42.10
2 2 0.925 24.17
3 3 1.385 17.47
4 4 1.842 13.87
5 5 2.295 11.60
6 6 2.742 10.01
7 7 3.184 8.84
8 8 3.618 7.94
9 9 4.045 7.22
10 10 4.463 6.62
11 11 4.872 6.12
12 12 5.272 5.70
13 13 5.661 5.33
```

```
14 14 6.041 5.00
15 15 6.411 4.71
16 16 6.770 4.45
17 17 7.119 4.21
18 18 7.457 4.00
19 19 7.785 3.79
20 20 8.103 3.61
21 21 8.410 3.43
22 22 8.708 3.26
23 23 8.996 3.09
24 24 9.275 2.93
25 25 9.544 2.78
26 26 9.805 2.63
27 27 10.057 2.47
28 28 10.301 2.32
29 29 10.537 2.17
30 30 10.765 2.02
31 31 10.986 1.86
32 32 11.199 1.71
33 33 11.406 1.55
34 34 11.606 1.38
35 35 11.800 1.21
36 36 11.988 1.04
37 37 12.170 0.86
38 38 12.347 0.67
39 39 12.518 0.49
40 40 12.684 0.29
41 41 12.845 0.09
42 42 13.002 -0.11
43 43 13.154 -0.32
44 44 13.302 -0.54
45 45 13.446 -0.75
46 46 13.586 -0.97
47 47 13.723 -1.20
48 48 13.855 -1.43
49 50 14.111 -1.88
50 52 14.354 -2.34
51 54 14.585 -2.79
52 56 14.807 -3.22
53 58 15.018 -3.62
54 60 15.221 -3.98
55 62 15.415 -4.30
56 64 15.602 -4.57
57 66 15.783 -4.77
58 68 15.956 -4.91
59 70 16.124 -4.98
60 72 16.287 -4.97
61 74 16.445 -4.90
62 76 16.598 -4.76
63 78 16.746 -4.55
64 80 16.891 -4.29
65 82 17.032 -3.99
66 84 17.169 -3.64
67 86 17.303 -3.26
68 88 17.434 -2.86
69 90 17.563 -2.45
70 92 17.688 -2.04
71 94 17.811 -1.63
72 96 17.932 -1.24
73 100 18.166 -0.51
74 104 18.392 0.12
75 108 18.611 0.64
76 112 18.823 1.06
77 116 19.028 1.39
78 120 19.226 1.66
79 124 19.419 1.88
80 128 19.606 2.08
81 132 19.788 2.27
82 136 19.964 2.46
83 140 20.135 2.65
84 144 20.300 2.86
85 148 20.461 3.09
86 152 20.616 3.33
87 156 20.766 3.60
88 160 20.912 3.89
89 164 21.052 4.20
90 168 21.188 4.54
91 172 21.318 4.91
92 176 21.445 5.31
93 180 21.567 5.73
94 184 21.684 6.18
95 188 21.797 6.67
96 192 21.906 7.19
97 200 22.113 8.33
98 208 22.304 9.63
99 216 22.482 11.08
100 224 22.646 12.71
101 232 22.799 14.53
102 240 22.941 16.54
103 248 23.072 18.77
104 256 23.195 21.23
105 264 23.309 23.94
106 272 23.415 26.90
107 280 23.515 30.14
108 288 23.607 33.67
109 296 23.694 37.51
110 304 23.775 41.67
111 312 23.852 46.17
112 320 23.923 51.04
113 328 23.991 56.29
114 336 24.054 61.94
115 344 24.114 68.00
116 352 24.171 68.00
117 360 24.224 68.00
118 368 24.275 68.00
119 376 24.322 68.00
120 384 24.368 68.00
121 392 24.411 68.00
122 400 24.452 68.00
123 408 24.491 68.00
124 416 24.528 68.00
125 424 24.564 68.00
126 432 24.597 68.00
```

C.2.12 2th2

```
133
1 1 0.309 58.23
2 2 0.617 33.44
3 3 0.925 24.17
4 4 1.232 19.20
5 5 1.538 16.05
6 6 1.842 13.87
7 7 2.145 12.26
8 8 2.445 11.01
9 9 2.742 10.01
10 10 3.037 9.20
11 11 3.329 8.52
12 12 3.618 7.94
13 13 3.903 7.44
14 14 4.185 7.00
15 15 4.463 6.62
16 16 4.736 6.28
17 17 5.006 5.97
18 18 5.272 5.70
19 19 5.533 5.44
20 20 5.789 5.21
21 21 6.041 5.00
22 22 6.289 4.80
23 23 6.532 4.62
24 24 6.770 4.45
25 25 7.004 4.29
26 26 7.233 4.14
27 27 7.457 4.00
28 28 7.677 3.86
29 29 7.892 3.73
30 30 8.103 3.61
31 31 8.309 3.49
32 32 8.511 3.37
33 33 8.708 3.26
34 34 8.901 3.15
35 35 9.090 3.04
36 36 9.275 2.93
37 37 9.456 2.83
38 38 9.632 2.73
39 39 9.805 2.63
40 40 9.974 2.53
41 41 10.139 2.42
42 42 10.301 2.32
43 43 10.459 2.22
44 44 10.614 2.12
45 45 10.765 2.02
46 46 10.913 1.92
47 47 11.058 1.81
48 48 11.199 1.71
49 50 11.474 1.49
50 52 11.736 1.27
51 54 11.988 1.04
52 56 12.230 0.80
53 58 12.461 0.55
54 60 12.684 0.29
55 62 12.898 0.02
56 64 13.104 -0.25
57 66 13.302 -0.54
58 68 13.493 -0.83
59 70 13.678 -1.12
60 72 13.855 -1.43
61 74 14.027 -1.73
62 76 14.193 -2.04
63 78 14.354 -2.34
64 80 14.509 -2.64
65 82 14.660 -2.93
66 84 14.807 -3.22
67 86 14.949 -3.49
68 88 15.087 -3.74
69 90 15.221 -3.98
70 92 15.351 -4.20
71 94 15.478 -4.40
72 96 15.602 -4.57
73 100 15.841 -4.82
74 104 16.069 -4.96
75 108 16.287 -4.97
76 112 16.496 -4.86
77 116 16.697 -4.63
78 120 16.891 -4.29
79 124 17.078 -3.87
80 128 17.259 -3.39
81 132 17.434 -2.86
82 136 17.605 -2.31
83 140 17.770 -1.77
84 144 17.932 -1.24
85 148 18.089 -0.74
86 152 18.242 -0.29
87 156 18.392 0.12
88 160 18.539 0.48
89 164 18.682 0.79
90 168 18.823 1.06
91 172 18.960 1.29
92 176 19.095 1.49
93 180 19.226 1.66
94 184 19.356 1.81
95 188 19.482 1.95
96 192 19.606 2.08
97 200 19.847 2.33
98 208 20.079 2.59
99 216 20.300 2.86
100 224 20.513 3.17
101 232 20.717 3.51
102 240 20.912 3.89
103 248 21.098 4.31
104 256 21.275 4.79
105 264 21.445 5.31
106 272 21.606 5.88
107 280 21.760 6.50
108 288 21.906 7.19
109 296 22.046 7.93
110 304 22.178 8.75
```

```
111 312 22.304 9.63
112 320 22.424 10.58
113 328 22.538 11.60
114 336 22.646 12.71
115 344 22.749 13.90
116 352 22.847 15.18
117 360 22.941 16.54
118 368 23.030 18.01
119 376 23.114 19.57
120 384 23.195 21.23
121 392 23.272 23.01
122 400 23.345 24.90
123 408 23.415 26.90
124 416 23.482 29.03
125 424 23.546 31.28
126 432 23.607 33.67
127 440 23.666 36.19
128 448 23.722 38.86
129 456 23.775 41.67
130 464 23.827 44.63
131 472 23.876 47.76
132 480 23.923 51.04
```

C.2.13 absthr_0

```
table 0
10156347392.00
33708348.00
3987838.75
1269806.88
614805.19
372166.75
256884.33
192636.06
153016.27
126980.66
108576.95
95003.05
84671.56
76513.41
70103.16
64824.27
60358.38
56720.18
53424.14
50668.45
48276.70
46103.89
44232.10
42534.13
40995.63
39603.86
38347.55
37131.08
36036.08
35054.00
34098.68
33169.40
32339.82
31530.99
30742.39
29973.51
29291.23
28624.48
27972.91
27336.16
26652.48
26045.79
25452.92
24873.54
24307.35
23754.04
23159.95
22632.76
21514.81
21514.81
20452.08
20452.08
19397.13
19397.13
18354.29
18354.29
17327.57
17327.57
16320.66
16320.66
15336.90
15336.90
14412.44
14412.44
13481.48
13481.48
12610.65
12610.65
11796.08
11796.08
10983.42
10983.42
10250.32
10250.32
9544.16
9544.16
8907.13
8907.13
8312.61
8312.61
7775.67
7775.67
7273.40
7273.40
6834.99
6834.99
6452.64
6452.64
6105.73
6105.73
5804.14
5804.14
5542.91
5542.91
5330.13
5330.13
5031.96
5031.96
5031.96
5031.96
4872.34
4872.34
4872.34
4872.34
4861.13
4861.13
4861.13
4861.13
4985.83
4985.83
4985.83
4985.83
5257.00
5257.00
5257.00
5257.00
5685.09
5685.09
5685.09
5685.09
6262.35
6262.35
6262.35
6262.35
6994.19
6994.19
6994.19
6994.19
7902.01
7902.01
7902.01
7902.01
8968.87
8968.87
8968.87
8968.87
10156.35
10156.35
10156.35
10156.35
11474.60
11474.60
11474.60
11474.60
12874.71
12874.71
12874.71
12874.71
14280.31
14280.31
14280.31
14280.31
15694.14
15694.14
15694.14
15694.14
17050.52
17050.52
17050.52
17050.52
18312.08
18312.08
18312.08
18312.08
19486.67
19486.67
19486.67
19486.67
20546.49
20546.49
20546.49
20546.49
21514.81
21514.81
21514.81
21514.81
22373.69
22373.69
22373.69
22373.69
23159.95
23159.95
23159.95
23159.95
23918.70
23918.70
23918.70
23918.70
24645.50
24645.50
24645.50
24645.50
26105.83
26105.83
26105.83
26105.83
26105.83
26105.83
26105.83
26105.83
27716.45
```

```
27716.45
27716.45
27716.45
27716.45
27716.45
27716.45
27716.45
29494.26
29494.26
29494.26
29494.26
29494.26
29494.26
29494.26
29494.26
31676.53
31676.53
31676.53
31676.53
31676.53
31676.53
31676.53
31676.53
34256.07
34256.07
34256.07
34256.07
34256.07
34256.07
34256.07
34256.07
37388.46
37388.46
37388.46
37388.46
37388.46
37388.46
37388.46
37388.46
41184.86
41184.86
41184.86
41184.86
41184.86
41184.86
41184.86
41184.86
45997.86
45997.86
45997.86
45997.86
45997.86
45997.86
45997.86
45997.86
51848.66
51848.66
51848.66
51848.66
51848.66
51848.66
51848.66
51848.66
59120.43
59120.43
59120.43
59120.43
59120.43
59120.43
59120.43
59120.43
68192.65
68192.65
68192.65
68192.65
68192.65
68192.65
68192.65
68192.65
79935.11
79935.11
79935.11
79935.11
79935.11
79935.11
79935.11
79935.11
94784.55
94784.55
94784.55
94784.55
94784.55
94784.55
94784.55
94784.55
114482.10
114482.10
114482.10
114482.10
114482.10
114482.10
114482.10
114482.10
140196.64
140196.64
140196.64
140196.64
140196.64
140196.64
140196.64
140196.64
174476.75
174476.75
174476.75
174476.75
174476.75
174476.75
174476.75
174476.75
220667.11
220667.11
220667.11
220667.11
220667.11
220667.11
220667.11
220667.11
284929.63
284929.63
284929.63
284929.63
284929.63
284929.63
284929.63
284929.63
374746.47
374746.47
374746.47
374746.47
374746.47
374746.47
374746.47
374746.47
503196.50
503196.50
503196.50
503196.50
503196.50
503196.50
503196.50
503196.50
688236.44
688236.44
688236.44
688236.44
688236.44
688236.44
688236.44
688236.44
965467.38
965467.38
965467.38
965467.38
965467.38
965467.38
965467.38
965467.38
1382730.50
1382730.50
1382730.50
1382730.50
1382730.50
1382730.50
1382730.50
1382730.50
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
3053076.00
3053076.00
3053076.00
3053076.00
3053076.00
3053076.00
3053076.00
3053076.00
4717778.50
4717778.50
4717778.50
4717778.50
4717778.50
4717778.50
4717778.50
4717778.50
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
12210618.00
12210618.00
12210618.00
12210618.00
12210618.00
12210618.00
12210618.00
12210618.00
20499234.00
20499234.00
20499234.00
20499234.00
20499234.00
20499234.00
20499234.00
20499234.00
35541636.00
35541636.00
35541636.00
35541636.00
35541636.00
```

35541636.00
35541636.00
35541636.00
63494696.00
63494696.00
63494696.00
63494696.00
63494696.00
63494696.00
63494696.00
63494696.00
117418800.00
117418800.00
117418800.00
117418800.00
117418800.00
117418800.00
117418800.00
117418800.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
443340800.00
443340800.00
443340800.00
443340800.00
443340800.00
443340800.00
443340800.00
443340800.00
911459584.00
911459584.00
911459584.00
911459584.00
911459584.00
911459584.00
911459584.00
911459584.00
1935251712.00
1935251712.00
1935251712.00
1935251712.00
1935251712.00
1935251712.00
1935251712.00
1935251712.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00

C.2.14 absthr_1

table 1
488357088.00
5898447.00
1131716.50
466377.53
265911.78
180192.88
135125.59
108078.07
90518.55
78295.64
69141.30
62335.81
56981.98
52569.95
49061.14
45997.86
43524.88
41279.80
39331.23
37647.63
36119.16
34732.62
33399.32
32265.44
31170.05
30111.86
29156.65
28231.74
27336.16
26469.00
25629.35
24816.33
24029.10
23266.85
22476.96
21713.89
20976.72
20264.58
19531.59
18781.82
18102.46
17367.51
16700.81
16022.77
15336.90
14680.39
14051.98
13419.54
12238.76
12238.76
11136.21
11136.21
10109.68
10109.68
9177.78
9177.78
8331.78
8331.78
7598.67
7598.67
6962.06
6962.06
6408.22
6408.22
5953.02
5953.02
5581.33
5581.33
5293.44
5293.44
5078.52
5078.52
4928.76
4928.76
4861.13
4861.13
4849.95
4849.95
4917.42
4917.42
5043.56
5043.56
5232.84
5232.84
5504.75
5504.75
5844.37
5844.37
6262.35
6262.35
6756.75
6756.75
7323.82
7323.82
7975.13
7975.13
9500.31
9500.31
9500.31
9500.31
11239.26
11239.26
11239.26
11239.26
13174.60
13174.60
13174.60
13174.60
15126.47
15126.47
15126.47
15126.47
17011.30
17011.30
17011.30
17011.30
18738.62
18738.62
18738.62
18738.62
20264.58
20264.58
20264.58
20264.58
21614.12
21614.12
21614.12
21614.12
22789.65
22789.65
22789.65
22789.65
23808.80
23808.80
23808.80
23808.80
24816.33
24816.33
24816.33

24816.33
25807.00
25807.00
25807.00
25807.00
26837.22
26837.22
26837.22
26837.22
27972.91
27972.91
27972.91
27972.91
29223.86
29223.86
29223.86
29223.86
30671.68
30671.68
30671.68
30671.68
32265.44
32265.44
32265.44
32265.44
34098.68
34098.68
34098.68
34098.68
36119.16
36119.16
36119.16
36119.16
38524.55
38524.55
38524.55
38524.55
41279.80
41279.80
41279.80
41279.80
44436.27
44436.27
44436.27
44436.27
48054.89
48054.89
48054.89
48054.89
52328.41
52328.41
52328.41
52328.41
62912.59
62912.59
62912.59
62912.59
62912.59
62912.59
62912.59
62912.59
77756.66
77756.66
77756.66
77756.66
77756.66
77756.66
77756.66
77756.66
98568.38
98568.38
98568.38
98568.38
98568.38
98568.38
98568.38
98568.38
128451.02
128451.02
128451.02
128451.02
128451.02
128451.02
128451.02
128451.02
172877.14
172877.14
172877.14
172877.14
172877.14
172877.14
172877.14
172877.14
240291.06
240291.06
240291.06
240291.06
240291.06
240291.06
240291.06
240291.06
346527.44
346527.44
346527.44
346527.44
346527.44
346527.44
346527.44
346527.44
517294.19
517294.19
517294.19
517294.19
517294.19
517294.19
517294.19
517294.19
806747.31
806747.31
806747.31
806747.31
806747.31
806747.31
806747.31
806747.31
1308391.13
1308391.13
1308391.13
1308391.13
1308391.13
1308391.13
1308391.13
1308391.13
2227088.75
2227088.75
2227088.75
2227088.75
2227088.75
2227088.75
2227088.75
2227088.75
3969516.00
3969516.00
3969516.00
3969516.00
3969516.00
3969516.00
3969516.00
3969516.00
7442819.50
7442819.50
7442819.50
7442819.50
7442819.50
7442819.50
7442819.50
7442819.50
14714232.00
14714232.00
14714232.00
14714232.00
14714232.00
14714232.00
14714232.00
14714232.00
30742394.00
30742394.00
30742394.00
30742394.00
30742394.00
30742394.00
30742394.00
30742394.00
68349824.00
68349824.00
68349824.00
68349824.00
68349824.00
68349824.00
68349824.00
68349824.00
161338400.00
161338400.00
161338400.00
161338400.00
161338400.00
161338400.00
161338400.00
161338400.00
407134048.00
407134048.00
407134048.00
407134048.00
407134048.00
407134048.00
407134048.00
407134048.00
1100874368.00
1100874368.00
1100874368.00
1100874368.00
1100874368.00
1100874368.00
1100874368.00
1100874368.00
3196962048.00
3196962048.00
3196962048.00
3196962048.00
3196962048.00
3196962048.00
3196962048.00
3196962048.00
10016993280.00
10016993280.00
10016993280.00
10016993280.00
10016993280.00
10016993280.00
10016993280.00
10016993280.00
34020253696.00
34020253696.00
34020253696.00
34020253696.00
34020253696.00
34020253696.00
34020253696.00

```
34020253696.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
124950298624.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
```

C.2.15 absthr_2

table 2

```
247592464.00
3987838.75
852584.69
372166.75
220667.11
153016.27
116879.30
95003.05
80489.19
70103.16
62479.50
56720.18
52087.99
48276.70
45158.30
42534.13
40247.38
38347.55
36537.39
35054.00
33630.83
32339.82
31098.37
29973.51
28955.93
27972.91
26961.10
26045.79
25161.56
24307.35
23428.13
22632.76
21814.11
20976.72
20171.47
19397.13
18609.62
17813.02
17089.82
16320.66
15586.10
14884.61
14182.00
13481.48
12845.10
12210.62
11580.77
10983.42
9902.33
9902.33
8907.13
8907.13
8030.41
8030.41
7273.40
7273.40
6633.42
6633.42
6105.73
6105.73
5672.02
5672.02
5330.13
5330.13
5090.23
5090.23
4928.76
4928.76
4849.95
4849.95
4861.13
4861.13
4940.12
4940.12
5101.97
5101.97
```

5354.73
5354.73
5685.09
5685.09
6091.69
6091.69
6602.94
6602.94
7206.72
7206.72
7902.01
7902.01
8684.36
8684.36
9544.16
9544.16
10489.08
10489.08
11474.60
11474.60
13574.93
13574.93
13574.93
13574.93
15694.14
15694.14
15694.14
15694.14
17690.40
17690.40
17690.40
17690.40
19486.67
19486.67
19486.67
19486.67
21025.08
21025.08
21025.08
21025.08
22373.69
22373.69
22373.69
22373.69
23536.27
23536.27
23536.27
23536.27
24645.50
24645.50
24645.50
24645.50
25747.65
25747.65
25747.65
25747.65
26899.09
26899.09
26899.09
26899.09
28102.02
28102.02
28102.02
28102.02
29494.26
29494.26
29494.26
29494.26
31098.37
31098.37
31098.37
31098.37
32865.30
32865.30
32865.30
32865.30
34973.38
34973.38
34973.38
34973.38
37388.46
37388.46
37388.46
37388.46
40154.81
40154.81
40154.81
40154.81
43424.77
43424.77
43424.77
43424.77
47286.54
47286.54
47286.54
47286.54
51848.66
51848.66
51848.66
51848.66
57113.34
57113.34
57113.34
57113.34
63348.68
63348.68
63348.68
63348.68
70914.91
70914.91
70914.91
70914.91
79935.11
79935.11
79935.11
79935.11
103929.20
103929.20
103929.20
103929.20
103929.20
103929.20
103929.20
103929.20
140196.64
140196.64
140196.64
140196.64
140196.64
140196.64
140196.64
140196.64
195766.14
195766.14
195766.14
195766.14
195766.14
195766.14
195766.14
195766.14
284929.63
284929.63
284929.63
284929.63
284929.63
284929.63
284929.63
284929.63
433249.00
433249.00
433249.00
433249.00
433249.00
433249.00
433249.00
433249.00
688236.44
688236.44
688236.44
688236.44
688236.44
688236.44
688236.44
688236.44
1150105.38
1150105.38
1150105.38
1150105.38
1150105.38
1150105.38
1150105.38
1150105.38
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
2026457.63
3782140.25
3782140.25
3782140.25
3782140.25
3782140.25
3782140.25
3782140.25
3782140.25
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
7477175.00
15766581.00
15766581.00
15766581.00
15766581.00
15766581.00
15766581.00
15766581.00
15766581.00
35541636.00
35541636.00
35541636.00
35541636.00
35541636.00
35541636.00
35541636.00
35541636.00
86047336.00
86047336.00
86047336.00
86047336.00
86047336.00
86047336.00
86047336.00
86047336.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00
224252560.00

```
632029568.00
632029568.00
632029568.00
632029568.00
632029568.00
632029568.00
632029568.00
632029568.00
1939713920.00
1939713920.00
1939713920.00
1939713920.00
1939713920.00
1939713920.00
1939713920.00
1939713920.00
6497371136.00
6497371136.00
6497371136.00
6497371136.00
6497371136.00
6497371136.00
6497371136.00
6497371136.00
23863681024.00
23863681024.00
23863681024.00
23863681024.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
96324681728.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
60776765194240.00
```

C.2.16 alloc_0

```
27
0 0 0 4 0 0
0 1 3 5 1 0
0 2 7 3 3 2
0 3 15 4 3 4
0 4 31 5 3 5
0 5 63 6 3 6
0 6 127 7 3 7
0 7 255 8 3 8
0 8 511 9 3 9
0 9 1023 10 3 10
0 10 2047 11 3 11
0 11 4095 12 3 12
0 12 8191 13 3 13
0 13 16383 14 3 14
```

```
0 14 32767 15 3 15
0 15 65535 16 3 16
1 0 0 4 0 0
1 1 3 5 1 0
1 2 7 3 3 2
1 3 15 4 3 4
1 4 31 5 3 5
1 5 63 6 3 6
1 6 127 7 3 7
1 7 255 8 3 8
1 8 511 9 3 9
1 9 1023 10 3 10
1 10 2047 11 3 11
1 11 4095 12 3 12
1 12 8191 13 3 13
1 13 16383 14 3 14
1 14 32767 15 3 15
1 15 65535 16 3 16
2 0 0 4 0 0
2 1 3 5 1 0
2 2 7 3 3 2
2 3 15 4 3 4
2 4 31 5 3 5
2 5 63 6 3 6
2 6 127 7 3 7
2 7 255 8 3 8
2 8 511 9 3 9
2 9 1023 10 3 10
2 10 2047 11 3 11
2 11 4095 12 3 12
2 12 8191 13 3 13
2 13 16383 14 3 14
2 14 32767 15 3 15
2 15 65535 16 3 16
3 0 0 4 0 0
3 1 3 5 1 0
3 2 5 7 1 1
3 3 7 3 3 2
3 4 9 10 1 3
3 5 15 4 3 4
3 6 31 5 3 5
3 7 63 6 3 6
3 8 127 7 3 7
3 9 255 8 3 8
3 10 511 9 3 9
3 11 1023 10 3 10
3 12 2047 11 3 11
3 13 4095 12 3 12
3 14 8191 13 3 13
3 15 65535 16 3 16
4 0 0 4 0 0
4 1 3 5 1 0
4 2 5 7 1 1
4 3 7 3 3 2
4 4 9 10 1 3
4 5 15 4 3 4
4 6 31 5 3 5
4 7 63 6 3 6
4 8 127 7 3 7
4 9 255 8 3 8
4 10 511 9 3 9
4 11 1023 10 3 10
4 12 2047 11 3 11
4 13 4095 12 3 12
4 14 8191 13 3 13
4 15 65535 16 3 16
5 0 0 4 0 0
5 1 3 5 1 0
5 2 5 7 1 1
5 3 7 3 3 2
5 4 9 10 1 3
5 5 15 4 3 4
5 6 31 5 3 5
5 7 63 6 3 6
5 8 127 7 3 7
5 9 255 8 3 8
5 10 511 9 3 9
5 11 1023 10 3 10
5 12 2047 11 3 11
5 13 4095 12 3 12
5 14 8191 13 3 13
5 15 65535 16 3 16
6 0 0 4 0 0
6 1 3 5 1 0
6 2 5 7 1 1
6 3 7 3 3 2
6 4 9 10 1 3
6 5 15 4 3 4
6 6 31 5 3 5
6 7 63 6 3 6
6 8 127 7 3 7
6 9 255 8 3 8
6 10 511 9 3 9
6 11 1023 10 3 10
6 12 2047 11 3 11
6 13 4095 12 3 12
6 14 8191 13 3 13
6 15 65535 16 3 16
7 0 0 4 0 0
7 1 3 5 1 0
7 2 5 7 1 1
7 3 7 3 3 2
7 4 9 10 1 3
7 5 15 4 3 4
7 6 31 5 3 5
7 7 63 6 3 6
7 8 127 7 3 7
7 9 255 8 3 8
7 10 511 9 3 9
7 11 1023 10 3 10
7 12 2047 11 3 11
7 13 4095 12 3 12
7 14 8191 13 3 13
7 15 65535 16 3 16
8 0 0 4 0 0
8 1 3 5 1 0
8 2 5 7 1 1
8 3 7 3 3 2
8 4 9 10 1 3
8 5 15 4 3 4
8 6 31 5 3 5
8 7 63 6 3 6
8 8 127 7 3 7
8 9 255 8 3 8
8 10 511 9 3 9
8 11 1023 10 3 10
8 12 2047 11 3 11
8 13 4095 12 3 12
8 14 8191 13 3 13
8 15 65535 16 3 16
9 0 0 4 0 0
9 1 3 5 1 0
9 2 5 7 1 1
9 3 7 3 3 2
9 4 9 10 1 3
9 5 15 4 3 4
9 6 31 5 3 5
9 7 63 6 3 6
9 8 127 7 3 7
9 9 255 8 3 8
9 10 511 9 3 9
9 11 1023 10 3 10
9 12 2047 11 3 11
9 13 4095 12 3 12
9 14 8191 13 3 13
9 15 65535 16 3 16
10 0 0 4 0 0
10 1 3 5 1 0
10 2 5 7 1 1
10 3 7 3 3 2
10 4 9 10 1 3
10 5 15 4 3 4
10 6 31 5 3 5
10 7 63 6 3 6
10 8 127 7 3 7
10 9 255 8 3 8
10 10 511 9 3 9
10 11 1023 10 3 10
10 12 2047 11 3 11
10 13 4095 12 3 12
10 14 8191 13 3 13
10 15 65535 16 3 16
11 0 0 3 0 0
11 1 3 5 1 0
11 2 5 7 1 1
11 3 7 3 3 2
11 4 9 10 1 3
11 5 15 4 3 4
11 6 31 5 3 5
11 7 65535 16 3 16
12 0 0 3 0 0
12 1 3 5 1 0
12 2 5 7 1 1
12 3 7 3 3 2
12 4 9 10 1 3
12 5 15 4 3 4
12 6 31 5 3 5
12 7 65535 16 3 16
13 0 0 3 0 0
13 1 3 5 1 0
13 2 5 7 1 1
13 3 7 3 3 2
13 4 9 10 1 3
13 5 15 4 3 4
13 6 31 5 3 5
13 7 65535 16 3 16
14 0 0 3 0 0
14 1 3 5 1 0
14 2 5 7 1 1
14 3 7 3 3 2
14 4 9 10 1 3
14 5 15 4 3 4
14 6 31 5 3 5
14 7 65535 16 3 16
15 0 0 3 0 0
15 1 3 5 1 0
15 2 5 7 1 1
15 3 7 3 3 2
15 4 9 10 1 3
15 5 15 4 3 4
15 6 31 5 3 5
15 7 65535 16 3 16
16 0 0 3 0 0
16 1 3 5 1 0
16 2 5 7 1 1
16 3 7 3 3 2
16 4 9 10 1 3
16 5 15 4 3 4
16 6 31 5 3 5
16 7 65535 16 3 16
17 0 0 3 0 0
17 1 3 5 1 0
17 2 5 7 1 1
17 3 7 3 3 2
17 4 9 10 1 3
17 5 15 4 3 4
17 6 31 5 3 5
17 7 65535 16 3 16
18 0 0 3 0 0
18 1 3 5 1 0
18 2 5 7 1 1
18 3 7 3 3 2
18 4 9 10 1 3
18 5 15 4 3 4
18 6 31 5 3 5
18 7 65535 16 3 16
19 0 0 3 0 0
19 1 3 5 1 0
```

```
19 2 5 7 1 1
19 3 7 3 3 2
19 4 9 10 1 3
19 5 15 4 3 4
19 6 31 5 3 5
19 7 65535 16 3 16
20 0 0 3 0 0
20 1 3 5 1 0
20 2 5 7 1 1
20 3 7 3 3 2
20 4 9 10 1 3
20 5 15 4 3 4
20 6 31 5 3 5
20 7 65535 16 3 16
21 0 0 3 0 0
21 1 3 5 1 0
21 2 5 7 1 1
21 3 7 3 3 2
21 4 9 10 1 3
21 5 15 4 3 4
21 6 31 5 3 5
21 7 65535 16 3 16
22 0 0 3 0 0
22 1 3 5 1 0
22 2 5 7 1 1
22 3 7 3 3 2
22 4 9 10 1 3
22 5 15 4 3 4
22 6 31 5 3 5
22 7 65535 16 3 16
23 0 0 2 0 0
23 1 3 5 1 0
23 2 5 7 1 1
23 3 65535 16 3 16
24 0 0 2 0 0
24 1 3 5 1 0
24 2 5 7 1 1
24 3 65535 16 3 16
25 0 0 2 0 0
25 1 3 5 1 0
25 2 5 7 1 1
25 3 65535 16 3 16
26 0 0 2 0 0
26 1 3 5 1 0
26 2 5 7 1 1
26 3 65535 16 3 16
```

C.2.17 alloc_1

```
30
0 0 0 4 0 0
0 1 3 5 1 0
0 2 7 3 3 2
0 3 15 4 3 4
0 4 31 5 3 5
0 5 63 6 3 6
0 6 127 7 3 7
0 7 255 8 3 8
0 8 511 9 3 9
0 9 1023 10 3 10
0 10 2047 11 3 11
0 11 4095 12 3 12
0 12 8191 13 3 13
0 13 16383 14 3 14
0 14 32767 15 3 15
0 15 65535 16 3 16
1 0 0 4 0 0
1 1 3 5 1 0
1 2 7 3 3 2
1 3 15 4 3 4
1 4 31 5 3 5
1 5 63 6 3 6
1 6 127 7 3 7
1 7 255 8 3 8
1 8 511 9 3 9
1 9 1023 10 3 10
1 10 2047 11 3 11
1 11 4095 12 3 12
1 12 8191 13 3 13
1 13 16383 14 3 14
1 14 32767 15 3 15
1 15 65535 16 3 16
2 0 0 4 0 0
2 1 3 5 1 0
2 2 7 3 3 2
2 3 15 4 3 4
2 4 31 5 3 5
2 5 63 6 3 6
2 6 127 7 3 7
2 7 255 8 3 8
2 8 511 9 3 9
2 9 1023 10 3 10
2 10 2047 11 3 11
2 11 4095 12 3 12
2 12 8191 13 3 13
2 13 16383 14 3 14
2 14 32767 15 3 15
2 15 65535 16 3 16
3 0 0 4 0 0
3 1 3 5 1 0
3 2 5 7 1 1
3 3 7 3 3 2
3 4 9 10 1 3
3 5 15 4 3 4
3 6 31 5 3 5
3 7 63 6 3 6
3 8 127 7 3 7
3 9 255 8 3 8
3 10 511 9 3 9
3 11 1023 10 3 10
3 12 2047 11 3 11
3 13 4095 12 3 12
3 14 8191 13 3 13
3 15 65535 16 3 16
4 0 0 4 0 0
4 1 3 5 1 0
4 2 5 7 1 1
4 3 7 3 3 2
4 4 9 10 1 3
4 5 15 4 3 4
4 6 31 5 3 5
4 7 63 6 3 6
4 8 127 7 3 7
4 9 255 8 3 8
4 10 511 9 3 9
4 11 1023 10 3 10
4 12 2047 11 3 11
4 13 4095 12 3 12
4 14 8191 13 3 13
4 15 65535 16 3 16
5 0 0 4 0 0
5 1 3 5 1 0
5 2 5 7 1 1
5 3 7 3 3 2
5 4 9 10 1 3
5 5 15 4 3 4
5 6 31 5 3 5
5 7 63 6 3 6
5 8 127 7 3 7
5 9 255 8 3 8
5 10 511 9 3 9
5 11 1023 10 3 10
5 12 2047 11 3 11
5 13 4095 12 3 12
5 14 8191 13 3 13
5 15 65535 16 3 16
6 0 0 4 0 0
6 1 3 5 1 0
6 2 5 7 1 1
6 3 7 3 3 2
6 4 9 10 1 3
6 5 15 4 3 4
6 6 31 5 3 5
6 7 63 6 3 6
6 8 127 7 3 7
6 9 255 8 3 8
6 10 511 9 3 9
6 11 1023 10 3 10
6 12 2047 11 3 11
6 13 4095 12 3 12
6 14 8191 13 3 13
6 15 65535 16 3 16
7 0 0 4 0 0
7 1 3 5 1 0
7 2 5 7 1 1
7 3 7 3 3 2
7 4 9 10 1 3
7 5 15 4 3 4
7 6 31 5 3 5
7 7 63 6 3 6
7 8 127 7 3 7
7 9 255 8 3 8
7 10 511 9 3 9
7 11 1023 10 3 10
7 12 2047 11 3 11
7 13 4095 12 3 12
7 14 8191 13 3 13
7 15 65535 16 3 16
8 0 0 4 0 0
8 1 3 5 1 0
8 2 5 7 1 1
8 3 7 3 3 2
8 4 9 10 1 3
8 5 15 4 3 4
8 6 31 5 3 5
8 7 63 6 3 6
8 8 127 7 3 7
8 9 255 8 3 8
8 10 511 9 3 9
8 11 1023 10 3 10
8 12 2047 11 3 11
8 13 4095 12 3 12
8 14 8191 13 3 13
8 15 65535 16 3 16
9 0 0 4 0 0
9 1 3 5 1 0
9 2 5 7 1 1
9 3 7 3 3 2
9 4 9 10 1 3
9 5 15 4 3 4
9 6 31 5 3 5
9 7 63 6 3 6
9 8 127 7 3 7
9 9 255 8 3 8
9 10 511 9 3 9
9 11 1023 10 3 10
9 12 2047 11 3 11
9 13 4095 12 3 12
9 14 8191 13 3 13
9 15 65535 16 3 16
10 0 0 4 0 0
10 1 3 5 1 0
10 2 5 7 1 1
10 3 7 3 3 2
10 4 9 10 1 3
10 5 15 4 3 4
10 6 31 5 3 5
10 7 63 6 3 6
10 8 127 7 3 7
10 9 255 8 3 8
10 10 511 9 3 9
10 11 1023 10 3 10
10 12 2047 11 3 11
10 13 4095 12 3 12
10 14 8191 13 3 13
10 15 65535 16 3 16
11 0 0 3 0 0
```

```
11 1 3 5 1 0
11 2 5 7 1 1
11 3 7 3 3 2
11 4 9 10 1 3
11 5 15 4 3 4
11 6 31 5 3 5
11 7 65535 16 3 16
12 0 0 3 0 0
12 1 3 5 1 0
12 2 5 7 1 1
12 3 7 3 3 2
12 4 9 10 1 3
12 5 15 4 3 4
12 6 31 5 3 5
12 7 65535 16 3 16
13 0 0 3 0 0
13 1 3 5 1 0
13 2 5 7 1 1
13 3 7 3 3 2
13 4 9 10 1 3
13 5 15 4 3 4
13 6 31 5 3 5
13 7 65535 16 3 16
14 0 0 3 0 0
14 1 3 5 1 0
14 2 5 7 1 1
14 3 7 3 3 2
14 4 9 10 1 3
14 5 15 4 3 4
14 6 31 5 3 5
14 7 65535 16 3 16
15 0 0 3 0 0
15 1 3 5 1 0
15 2 5 7 1 1
15 3 7 3 3 2
15 4 9 10 1 3
15 5 15 4 3 4
15 6 31 5 3 5
15 7 65535 16 3 16
16 0 0 3 0 0
16 1 3 5 1 0
16 2 5 7 1 1
16 3 7 3 3 2
16 4 9 10 1 3
16 5 15 4 3 4
16 6 31 5 3 5
16 7 65535 16 3 16
17 0 0 3 0 0
17 1 3 5 1 0
17 2 5 7 1 1
17 3 7 3 3 2
17 4 9 10 1 3
17 5 15 4 3 4
17 6 31 5 3 5
17 7 65535 16 3 16
18 0 0 3 0 0
18 1 3 5 1 0
18 2 5 7 1 1
18 3 7 3 3 2
18 4 9 10 1 3
18 5 15 4 3 4
18 6 31 5 3 5
18 7 65535 16 3 16
19 0 0 3 0 0
19 1 3 5 1 0
19 2 5 7 1 1
19 3 7 3 3 2
19 4 9 10 1 3
19 5 15 4 3 4
19 6 31 5 3 5
19 7 65535 16 3 16
20 0 0 3 0 0
20 1 3 5 1 0
20 2 5 7 1 1
20 3 7 3 3 2
20 4 9 10 1 3
20 5 15 4 3 4
20 6 31 5 3 5
20 7 65535 16 3 16
21 0 0 3 0 0
21 1 3 5 1 0
21 2 5 7 1 1
21 3 7 3 3 2
21 4 9 10 1 3
21 5 15 4 3 4
21 6 31 5 3 5
21 7 65535 16 3 16
22 0 0 3 0 0
22 1 3 5 1 0
22 2 5 7 1 1
22 3 7 3 3 2
22 4 9 10 1 3
22 5 15 4 3 4
22 6 31 5 3 5
22 7 65535 16 3 16
23 0 0 2 0 0
23 1 3 5 1 0
23 2 5 7 1 1
23 3 65535 16 3 16
24 0 0 2 0 0
24 1 3 5 1 0
24 2 5 7 1 1
24 3 65535 16 3 16
25 0 0 2 0 0
25 1 3 5 1 0
25 2 5 7 1 1
25 3 65535 16 3 16
26 0 0 2 0 0
26 1 3 5 1 0
26 2 5 7 1 1
26 3 65535 16 3 16
27 0 0 2 0 0
27 1 3 5 1 0
27 2 5 7 1 1
27 3 65535 16 3 16
28 0 0 2 0 0
28 1 3 5 1 0
28 2 5 7 1 1
28 3 65535 16 3 16
29 0 0 2 0 0
29 1 3 5 1 0
29 2 5 7 1 1
29 3 65535 16 3 16
```

C.2.18 alloc_2

```
8
0 0 0 4 0 0
0 1 3 5 1 0
0 2 5 7 1 1
0 3 9 10 1 3
0 4 15 4 3 4
0 5 31 5 3 5
0 6 63 6 3 6
0 7 127 7 3 7
0 8 255 8 3 8
0 9 511 9 3 9
0 10 1023 10 3 10
0 11 2047 11 3 11
0 12 4095 12 3 12
0 13 8191 13 3 13
0 14 16383 14 3 14
0 15 32767 15 3 15
1 0 0 4 0 0
1 1 3 5 1 0
1 2 5 7 1 1
1 3 9 10 1 3
1 4 15 4 3 4
1 5 31 5 3 5
1 6 63 6 3 6
1 7 127 7 3 7
1 8 255 8 3 8
1 9 511 9 3 9
1 10 1023 10 3 10
1 11 2047 11 3 11
1 12 4095 12 3 12
1 13 8191 13 3 13
1 14 16383 14 3 14
1 15 32767 15 3 15
2 0 0 3 0 0
2 1 3 5 1 0
2 2 5 7 1 1
2 3 9 10 1 3
2 4 15 4 3 4
2 5 31 5 3 5
2 6 63 6 3 6
2 7 127 7 3 7
3 0 0 3 0 0
3 1 3 5 1 0
3 2 5 7 1 1
3 3 9 10 1 3
3 4 15 4 3 4
3 5 31 5 3 5
3 6 63 6 3 6
3 7 127 7 3 7
4 0 0 3 0 0
4 1 3 5 1 0
4 2 5 7 1 1
4 3 9 10 1 3
4 4 15 4 3 4
4 5 31 5 3 5
4 6 63 6 3 6
4 7 127 7 3 7
5 0 0 3 0 0
5 1 3 5 1 0
5 2 5 7 1 1
5 3 9 10 1 3
5 4 15 4 3 4
5 5 31 5 3 5
5 6 63 6 3 6
5 7 127 7 3 7
6 0 0 3 0 0
6 1 3 5 1 0
6 2 5 7 1 1
6 3 9 10 1 3
6 4 15 4 3 4
6 5 31 5 3 5
6 6 63 6 3 6
6 7 127 7 3 7
7 0 0 3 0 0
7 1 3 5 1 0
7 2 5 7 1 1
7 3 9 10 1 3
7 4 15 4 3 4
7 5 31 5 3 5
7 6 63 6 3 6
7 7 127 7 3 7
```

C.2.19 alloc_3

```
12
0 0 0 4 0 0
0 1 3 5 1 0
0 2 5 7 1 1
0 3 9 10 1 3
0 4 15 4 3 4
0 5 31 5 3 5
0 6 63 6 3 6
0 7 127 7 3 7
0 8 255 8 3 8
0 9 511 9 3 9
0 10 1023 10 3 10
0 11 2047 11 3 11
0 12 4095 12 3 12
0 13 8191 13 3 13
```

```
0 14 16383 14 3 14
0 15 32767 15 3 15
1 0 0 4 0 0
1 1 3 5 1 0
1 2 5 7 1 1
1 3 9 10 1 3
1 4 15 4 3 4
1 5 31 5 3 5
1 6 63 6 3 6
1 7 127 7 3 7
1 8 255 8 3 8
1 9 511 9 3 9
1 10 1023 10 3 10
1 11 2047 11 3 11
1 12 4095 12 3 12
1 13 8191 13 3 13
1 14 16383 14 3 14
1 15 32767 15 3 15
2 0 0 3 0 0
2 1 3 5 1 0
2 2 5 7 1 1
2 3 9 10 1 3
2 4 15 4 3 4
2 5 31 5 3 5
2 6 63 6 3 6
2 7 127 7 3 7
3 0 0 3 0 0
3 1 3 5 1 0
3 2 5 7 1 1
3 3 9 10 1 3
3 4 15 4 3 4
3 5 31 5 3 5
3 6 63 6 3 6
3 7 127 7 3 7
4 0 0 3 0 0
4 1 3 5 1 0
4 2 5 7 1 1
4 3 9 10 1 3
4 4 15 4 3 4
4 5 31 5 3 5
4 6 63 6 3 6
4 7 127 7 3 7
5 0 0 3 0 0
5 1 3 5 1 0
5 2 5 7 1 1
5 3 9 10 1 3
5 4 15 4 3 4
5 5 31 5 3 5
5 6 63 6 3 6
5 7 127 7 3 7
6 0 0 3 0 0
6 1 3 5 1 0
6 2 5 7 1 1
6 3 9 10 1 3
6 4 15 4 3 4
6 5 31 5 3 5
6 6 63 6 3 6
6 7 127 7 3 7
7 0 0 3 0 0
7 1 3 5 1 0
7 2 5 7 1 1
7 3 9 10 1 3
7 4 15 4 3 4
7 5 31 5 3 5
7 6 63 6 3 6
7 7 127 7 3 7
8 0 0 3 0 0
8 1 3 5 1 0
8 2 5 7 1 1
8 3 9 10 1 3
8 4 15 4 3 4
8 5 31 5 3 5
8 6 63 6 3 6
8 7 127 7 3 7
9 0 0 3 0 0
9 1 3 5 1 0
9 2 5 7 1 1
9 3 9 10 1 3
9 4 15 4 3 4
9 5 31 5 3 5
9 6 63 6 3 6
9 7 127 7 3 7
10 0 0 3 0 0
10 1 3 5 1 0
10 2 5 7 1 1
10 3 9 10 1 3
10 4 15 4 3 4
10 5 31 5 3 5
10 6 63 6 3 6
10 7 127 7 3 7
11 0 0 3 0 0
11 1 3 5 1 0
11 2 5 7 1 1
11 3 9 10 1 3
11 4 15 4 3 4
11 5 31 5 3 5
11 6 63 6 3 6
11 7 127 7 3 7
```

C.2.20 enwindow

```
C[  0]= 0.000000000 C[  1]=-0.000000477 C[  2]=-0.000000477 C[  3]=-0.000000477
C[  4]=-0.000000477 C[  5]=-0.000000477 C[  6]=-0.000000477 C[  7]=-0.000000954
C[  8]=-0.000000954 C[  9]=-0.000000954 C[ 10]=-0.000000954 C[ 11]=-0.000001431
C[ 12]=-0.000001431 C[ 13]=-0.000001907 C[ 14]=-0.000001907 C[ 15]=-0.000002384
C[ 16]=-0.000002384 C[ 17]=-0.000002861 C[ 18]=-0.000003338 C[ 19]=-0.000003338
C[ 20]=-0.000003815 C[ 21]=-0.000004292 C[ 22]=-0.000004768 C[ 23]=-0.000005245
C[ 24]=-0.000006199 C[ 25]=-0.000006676 C[ 26]=-0.000007629 C[ 27]=-0.000008106
C[ 28]=-0.000009060 C[ 29]=-0.000010014 C[ 30]=-0.000011444 C[ 31]=-0.000012398
C[ 32]=-0.000013828 C[ 33]=-0.000014782 C[ 34]=-0.000016689 C[ 35]=-0.000018120
C[ 36]=-0.000019550 C[ 37]=-0.000021458 C[ 38]=-0.000023365 C[ 39]=-0.000025272
C[ 40]=-0.000027657 C[ 41]=-0.000030041 C[ 42]=-0.000032425 C[ 43]=-0.000034809
C[ 44]=-0.000037670 C[ 45]=-0.000040531 C[ 46]=-0.000043392 C[ 47]=-0.000046253
C[ 48]=-0.000049591 C[ 49]=-0.000052929 C[ 50]=-0.000055790 C[ 51]=-0.000059605
C[ 52]=-0.000062943 C[ 53]=-0.000066280 C[ 54]=-0.000070095 C[ 55]=-0.000073433
C[ 56]=-0.000076771 C[ 57]=-0.000080585 C[ 58]=-0.000083923 C[ 59]=-0.000087261
C[ 60]=-0.000090599 C[ 61]=-0.000093460 C[ 62]=-0.000096321 C[ 63]=-0.000099182
C[ 64]= 0.000101566 C[ 65]= 0.000103951 C[ 66]= 0.000105858 C[ 67]= 0.000107288
C[ 68]= 0.000108242 C[ 69]= 0.000108719 C[ 70]= 0.000108719 C[ 71]= 0.000108242
C[ 72]= 0.000106812 C[ 73]= 0.000105381 C[ 74]= 0.000102520 C[ 75]= 0.000099182
C[ 76]= 0.000095367 C[ 77]= 0.000090122 C[ 78]= 0.000084400 C[ 79]= 0.000077724
C[ 80]= 0.000069618 C[ 81]= 0.000060558 C[ 82]= 0.000050545 C[ 83]= 0.000039577
C[ 84]= 0.000027180 C[ 85]= 0.000013828 C[ 86]=-0.000000954 C[ 87]=-0.000017166
C[ 88]=-0.000034332 C[ 89]=-0.000052929 C[ 90]=-0.000072956 C[ 91]=-0.000093937
C[ 92]=-0.000116348 C[ 93]=-0.000140190 C[ 94]=-0.000165462 C[ 95]=-0.000191212
C[ 96]=-0.000218868 C[ 97]=-0.000247478 C[ 98]=-0.000277042 C[ 99]=-0.000307560
C[100]=-0.000339031 C[101]=-0.000371456 C[102]=-0.000404358 C[103]=-0.000438213
C[104]=-0.000472546 C[105]=-0.000507355 C[106]=-0.000542164 C[107]=-0.000576973
C[108]=-0.000611782 C[109]=-0.000646591 C[110]=-0.000680923 C[111]=-0.000714302
C[112]=-0.000747204 C[113]=-0.000779152 C[114]=-0.000809669 C[115]=-0.000838757
C[116]=-0.000866413 C[117]=-0.000891685 C[118]=-0.000915051 C[119]=-0.000935555
C[120]=-0.000954151 C[121]=-0.000968933 C[122]=-0.000980854 C[123]=-0.000989437
C[124]=-0.000994205 C[125]=-0.000995159 C[126]=-0.000991821 C[127]=-0.000983715
C[128]= 0.000971317 C[129]= 0.000953674 C[130]= 0.000930786 C[131]= 0.000902653
C[132]= 0.000868797 C[133]= 0.000829220 C[134]= 0.000783920 C[135]= 0.000731945
C[136]= 0.000674248 C[137]= 0.000610352 C[138]= 0.000539303 C[139]= 0.000462532
C[140]= 0.000378609 C[141]= 0.000288486 C[142]= 0.000191689 C[143]= 0.000088215
C[144]=-0.000021458 C[145]=-0.000137329 C[146]=-0.000259876 C[147]=-0.000388145
C[148]=-0.000522137 C[149]=-0.000661850 C[150]=-0.000806808 C[151]=-0.000956535
C[152]=-0.001111031 C[153]=-0.001269817 C[154]=-0.001432419 C[155]=-0.001597881
```

```
C[156]=-0.001766682 C[157]=-0.001937389 C[158]=-0.002110004 C[159]=-0.002283096
C[160]=-0.002457142 C[161]=-0.002630711 C[162]=-0.002803326 C[163]=-0.002974033
C[164]=-0.003141880 C[165]=-0.003306866 C[166]=-0.003467083 C[167]=-0.003622532
C[168]=-0.003771782 C[169]=-0.003914356 C[170]=-0.004048824 C[171]=-0.004174709
C[172]=-0.004290581 C[173]=-0.004395962 C[174]=-0.004489899 C[175]=-0.004570484
C[176]=-0.004638195 C[177]=-0.004691124 C[178]=-0.004728317 C[179]=-0.004748821
C[180]=-0.004752159 C[181]=-0.004737377 C[182]=-0.004703045 C[183]=-0.004649162
C[184]=-0.004573822 C[185]=-0.004477024 C[186]=-0.004357815 C[187]=-0.004215240
C[188]=-0.004049301 C[189]=-0.003858566 C[190]=-0.003643036 C[191]=-0.003401756
C[192]= 0.003134727 C[193]= 0.002841473 C[194]= 0.002521515 C[195]= 0.002174854
C[196]= 0.001800537 C[197]= 0.001399517 C[198]= 0.000971317 C[199]= 0.000515938
C[200]= 0.000033379 C[201]=-0.000475883 C[202]=-0.001011848 C[203]=-0.001573563
C[204]=-0.002161503 C[205]=-0.002774239 C[206]=-0.003411293 C[207]=-0.004072189
C[208]=-0.004756451 C[209]=-0.005462170 C[210]=-0.006189346 C[211]=-0.006937027
C[212]=-0.007703304 C[213]=-0.008487225 C[214]=-0.009287834 C[215]=-0.010103703
C[216]=-0.010933399 C[217]=-0.011775017 C[218]=-0.012627602 C[219]=-0.013489246
C[220]=-0.014358521 C[221]=-0.015233517 C[222]=-0.016112804 C[223]=-0.016994476
C[224]=-0.017876148 C[225]=-0.018756866 C[226]=-0.019634247 C[227]=-0.020506859
C[228]=-0.021372318 C[229]=-0.022228718 C[230]=-0.023074150 C[231]=-0.023907185
C[232]=-0.024725437 C[233]=-0.025527000 C[234]=-0.026310921 C[235]=-0.027073860
C[236]=-0.027815342 C[237]=-0.028532982 C[238]=-0.029224873 C[239]=-0.029890060
C[240]=-0.030526638 C[241]=-0.031132698 C[242]=-0.031706810 C[243]=-0.032248020
C[244]=-0.032754898 C[245]=-0.033225536 C[246]=-0.033659935 C[247]=-0.034055710
C[248]=-0.034412861 C[249]=-0.034730434 C[250]=-0.035007000 C[251]=-0.035242081
C[252]=-0.035435200 C[253]=-0.035586357 C[254]=-0.035694122 C[255]=-0.035758972
C[256]= 0.035780907 C[257]= 0.035758972 C[258]= 0.035694122 C[259]= 0.035586357
C[260]= 0.035435200 C[261]= 0.035242081 C[262]= 0.035007000 C[263]= 0.034730434
C[264]= 0.034412861 C[265]= 0.034055710 C[266]= 0.033659935 C[267]= 0.033225536
C[268]= 0.032754898 C[269]= 0.032248020 C[270]= 0.031706810 C[271]= 0.031132698
C[272]= 0.030526638 C[273]= 0.029890060 C[274]= 0.029224873 C[275]= 0.028532982
C[276]= 0.027815342 C[277]= 0.027073860 C[278]= 0.026310921 C[279]= 0.025527000
C[280]= 0.024725437 C[281]= 0.023907185 C[282]= 0.023074150 C[283]= 0.022228718
C[284]= 0.021372318 C[285]= 0.020506859 C[286]= 0.019634247 C[287]= 0.018756866
C[288]= 0.017876148 C[289]= 0.016994476 C[290]= 0.016112804 C[291]= 0.015233517
C[292]= 0.014358521 C[293]= 0.013489246 C[294]= 0.012627602 C[295]= 0.011775017
C[296]= 0.010933399 C[297]= 0.010103703 C[298]= 0.009287834 C[299]= 0.008487225
C[300]= 0.007703304 C[301]= 0.006937027 C[302]= 0.006189346 C[303]= 0.005462170
C[304]= 0.004756451 C[305]= 0.004072189 C[306]= 0.003411293 C[307]= 0.002774239
C[308]= 0.002161503 C[309]= 0.001573563 C[310]= 0.001011848 C[311]= 0.000475883
C[312]=-0.000033379 C[313]=-0.000515938 C[314]=-0.000971317 C[315]=-0.001399517
C[316]=-0.001800537 C[317]=-0.002174854 C[318]=-0.002521515 C[319]=-0.002841473
C[320]= 0.003134727 C[321]= 0.003401756 C[322]= 0.003643036 C[323]= 0.003858566
C[324]= 0.004049301 C[325]= 0.004215240 C[326]= 0.004357815 C[327]= 0.004477024
C[328]= 0.004573822 C[329]= 0.004649162 C[330]= 0.004703045 C[331]= 0.004737377
C[332]= 0.004752159 C[333]= 0.004748821 C[334]= 0.004728317 C[335]= 0.004691124
C[336]= 0.004638195 C[337]= 0.004570484 C[338]= 0.004489899 C[339]= 0.004395962
C[340]= 0.004290581 C[341]= 0.004174709 C[342]= 0.004048824 C[343]= 0.003914356
C[344]= 0.003771782 C[345]= 0.003622532 C[346]= 0.003467083 C[347]= 0.003306866
C[348]= 0.003141880 C[349]= 0.002974033 C[350]= 0.002803326 C[351]= 0.002630711
C[352]= 0.002457142 C[353]= 0.002283096 C[354]= 0.002110004 C[355]= 0.001937389
C[356]= 0.001766682 C[357]= 0.001597881 C[358]= 0.001432419 C[359]= 0.001269817
C[360]= 0.001111031 C[361]= 0.000956535 C[362]= 0.000806808 C[363]= 0.000661850
C[364]= 0.000522137 C[365]= 0.000388145 C[366]= 0.000259876 C[367]= 0.000137329
C[368]= 0.000021458 C[369]=-0.000088215 C[370]=-0.000191689 C[371]=-0.000288486
C[372]=-0.000378609 C[373]=-0.000462532 C[374]=-0.000539303 C[375]=-0.000610352
C[376]=-0.000674248 C[377]=-0.000731945 C[378]=-0.000783920 C[379]=-0.000829220
C[380]=-0.000868797 C[381]=-0.000902653 C[382]=-0.000930786 C[383]=-0.000953674
C[384]= 0.000971317 C[385]= 0.000983715 C[386]= 0.000991821 C[387]= 0.000995159
C[388]= 0.000994205 C[389]= 0.000989437 C[390]= 0.000980854 C[391]= 0.000968933
C[392]= 0.000954151 C[393]= 0.000935555 C[394]= 0.000915051 C[395]= 0.000891685
C[396]= 0.000866413 C[397]= 0.000838757 C[398]= 0.000809669 C[399]= 0.000779152
C[400]= 0.000747204 C[401]= 0.000714302 C[402]= 0.000680923 C[403]= 0.000646591
C[404]= 0.000611782 C[405]= 0.000576973 C[406]= 0.000542164 C[407]= 0.000507355
C[408]= 0.000472546 C[409]= 0.000438213 C[410]= 0.000404358 C[411]= 0.000371456
C[412]= 0.000339031 C[413]= 0.000307560 C[414]= 0.000277042 C[415]= 0.000247478
C[416]= 0.000218868 C[417]= 0.000191212 C[418]= 0.000165462 C[419]= 0.000140190
C[420]= 0.000116348 C[421]= 0.000093937 C[422]= 0.000072956 C[423]= 0.000052929
C[424]= 0.000034332 C[425]= 0.000017166 C[426]= 0.000000954 C[427]=-0.000013828
C[428]=-0.000027180 C[429]=-0.000039577 C[430]=-0.000050545 C[431]=-0.000060558
C[432]=-0.000069618 C[433]=-0.000077724 C[434]=-0.000084400 C[435]=-0.000090122
C[436]=-0.000095367 C[437]=-0.000099182 C[438]=-0.000102520 C[439]=-0.000105381
C[440]=-0.000106812 C[441]=-0.000108242 C[442]=-0.000108719 C[443]=-0.000108719
C[444]=-0.000108242 C[445]=-0.000107288 C[446]=-0.000105858 C[447]=-0.000103951
C[448]= 0.000101566 C[449]= 0.000099182 C[450]= 0.000096321 C[451]= 0.000093460
C[452]= 0.000090599 C[453]= 0.000087261 C[454]= 0.000083923 C[455]= 0.000080585
C[456]= 0.000076771 C[457]= 0.000073433 C[458]= 0.000070095 C[459]= 0.000066280
```

```
C[460]= 0.000062943 C[461]= 0.000059605 C[462]= 0.000055790 C[463]= 0.000052929
C[464]= 0.000049591 C[465]= 0.000046253 C[466]= 0.000043392 C[467]= 0.000040531
C[468]= 0.000037670 C[469]= 0.000034809 C[470]= 0.000032425 C[471]= 0.000030041
C[472]= 0.000027657 C[473]= 0.000025272 C[474]= 0.000023365 C[475]= 0.000021458
C[476]= 0.000019550 C[477]= 0.000018120 C[478]= 0.000016689 C[479]= 0.000014782
C[480]= 0.000013828 C[481]= 0.000012398 C[482]= 0.000011444 C[483]= 0.000010014
C[484]= 0.000009060 C[485]= 0.000008106 C[486]= 0.000007629 C[487]= 0.000006676
C[488]= 0.000006199 C[489]= 0.000005245 C[490]= 0.000004768 C[491]= 0.000004292
C[492]= 0.000003815 C[493]= 0.000003338 C[494]= 0.000003338 C[495]= 0.000002861
C[496]= 0.000002384 C[497]= 0.000002384 C[498]= 0.000001907 C[499]= 0.000001907
C[500]= 0.000001431 C[501]= 0.000001431 C[502]= 0.000000954 C[503]= 0.000000954
C[504]= 0.000000954 C[505]= 0.000000954 C[506]= 0.000000477 C[507]= 0.000000477
C[508]= 0.000000477 C[509]= 0.000000477 C[510]= 0.000000477 C[511]= 0.000000477
```

C.2.21 dewindow

```
D[  0]= 0.000000000 D[  1]=-0.000015259 D[  2]=-0.000015259 D[  3]=-0.000015259
D[  4]=-0.000015259 D[  5]=-0.000015259 D[  6]=-0.000015259 D[  7]=-0.000030518
D[  8]=-0.000030518 D[  9]=-0.000030518 D[ 10]=-0.000030518 D[ 11]=-0.000045776
D[ 12]=-0.000045776 D[ 13]=-0.000061035 D[ 14]=-0.000061035 D[ 15]=-0.000076294
D[ 16]=-0.000076294 D[ 17]=-0.000091553 D[ 18]=-0.000106812 D[ 19]=-0.000106812
D[ 20]=-0.000122070 D[ 21]=-0.000137329 D[ 22]=-0.000152588 D[ 23]=-0.000167847
D[ 24]=-0.000198364 D[ 25]=-0.000213623 D[ 26]=-0.000244141 D[ 27]=-0.000259399
D[ 28]=-0.000289917 D[ 29]=-0.000320435 D[ 30]=-0.000366211 D[ 31]=-0.000396729
D[ 32]=-0.000442505 D[ 33]=-0.000473022 D[ 34]=-0.000534058 D[ 35]=-0.000579834
D[ 36]=-0.000625610 D[ 37]=-0.000686646 D[ 38]=-0.000747681 D[ 39]=-0.000808716
D[ 40]=-0.000885010 D[ 41]=-0.000961304 D[ 42]=-0.001037598 D[ 43]=-0.001113892
D[ 44]=-0.001205444 D[ 45]=-0.001296997 D[ 46]=-0.001388550 D[ 47]=-0.001480103
D[ 48]=-0.001586914 D[ 49]=-0.001693726 D[ 50]=-0.001785278 D[ 51]=-0.001907349
D[ 52]=-0.002014160 D[ 53]=-0.002120972 D[ 54]=-0.002243042 D[ 55]=-0.002349854
D[ 56]=-0.002456665 D[ 57]=-0.002578735 D[ 58]=-0.002685547 D[ 59]=-0.002792358
D[ 60]=-0.002899170 D[ 61]=-0.002990723 D[ 62]=-0.003082275 D[ 63]=-0.003173828
D[ 64]= 0.003250122 D[ 65]= 0.003326416 D[ 66]= 0.003387451 D[ 67]= 0.003433228
D[ 68]= 0.003463745 D[ 69]= 0.003479004 D[ 70]= 0.003479004 D[ 71]= 0.003463745
D[ 72]= 0.003417969 D[ 73]= 0.003372192 D[ 74]= 0.003280640 D[ 75]= 0.003173828
D[ 76]= 0.003051758 D[ 77]= 0.002883911 D[ 78]= 0.002700806 D[ 79]= 0.002487183
D[ 80]= 0.002227783 D[ 81]= 0.001937866 D[ 82]= 0.001617432 D[ 83]= 0.001266479
D[ 84]= 0.000869751 D[ 85]= 0.000442505 D[ 86]=-0.000030518 D[ 87]=-0.000549316
D[ 88]=-0.001098633 D[ 89]=-0.001693726 D[ 90]=-0.002334595 D[ 91]=-0.003005981
D[ 92]=-0.003723145 D[ 93]=-0.004486084 D[ 94]=-0.005294800 D[ 95]=-0.006118774
D[ 96]=-0.007003784 D[ 97]=-0.007919312 D[ 98]=-0.008865356 D[ 99]=-0.009841919
D[100]=-0.010848999 D[101]=-0.011886597 D[102]=-0.012939453 D[103]=-0.014022827
D[104]=-0.015121460 D[105]=-0.016235352 D[106]=-0.017349243 D[107]=-0.018463135
D[108]=-0.019577026 D[109]=-0.020690918 D[110]=-0.021789551 D[111]=-0.022857666
D[112]=-0.023910522 D[113]=-0.024932861 D[114]=-0.025909424 D[115]=-0.026840210
D[116]=-0.027725220 D[117]=-0.028533936 D[118]=-0.029281616 D[119]=-0.029937744
D[120]=-0.030532837 D[121]=-0.031005859 D[122]=-0.031387329 D[123]=-0.031661987
D[124]=-0.031814575 D[125]=-0.031845093 D[126]=-0.031738281 D[127]=-0.031478882
D[128]= 0.031082153 D[129]= 0.030517578 D[130]= 0.029785156 D[131]= 0.028884888
D[132]= 0.027801514 D[133]= 0.026535034 D[134]= 0.025085449 D[135]= 0.023422241
D[136]= 0.021575928 D[137]= 0.019531250 D[138]= 0.017257690 D[139]= 0.014801025
D[140]= 0.012115479 D[141]= 0.009231567 D[142]= 0.006134033 D[143]= 0.002822876
D[144]=-0.000686646 D[145]=-0.004394531 D[146]=-0.008316040 D[147]=-0.012420654
D[148]=-0.016708374 D[149]=-0.021179199 D[150]=-0.025817871 D[151]=-0.030609131
D[152]=-0.035552979 D[153]=-0.040634155 D[154]=-0.045837402 D[155]=-0.051132202
D[156]=-0.056533813 D[157]=-0.061996460 D[158]=-0.067520142 D[159]=-0.073059082
D[160]=-0.078628540 D[161]=-0.084182739 D[162]=-0.089706421 D[163]=-0.095169067
D[164]=-0.100540161 D[165]=-0.105819702 D[166]=-0.110946655 D[167]=-0.115921021
D[168]=-0.120697021 D[169]=-0.125259399 D[170]=-0.129562378 D[171]=-0.133590698
D[172]=-0.137298584 D[173]=-0.140670776 D[174]=-0.143676758 D[175]=-0.146255493
D[176]=-0.148422241 D[177]=-0.150115967 D[178]=-0.151306152 D[179]=-0.151962280
D[180]=-0.152069092 D[181]=-0.151596069 D[182]=-0.150497437 D[183]=-0.148773193
D[184]=-0.146362305 D[185]=-0.143264771 D[186]=-0.139450073 D[187]=-0.134887695
D[188]=-0.129577637 D[189]=-0.123474121 D[190]=-0.116577148 D[191]=-0.108856201
D[192]= 0.100311279 D[193]= 0.090927124 D[194]= 0.080688477 D[195]= 0.069595337
D[196]= 0.057617187 D[197]= 0.044784546 D[198]= 0.031082153 D[199]= 0.016510010
D[200]= 0.001068115 D[201]=-0.015228271 D[202]=-0.032379150 D[203]=-0.050354004
D[204]=-0.069168091 D[205]=-0.088775635 D[206]=-0.109161377 D[207]=-0.130310059
D[208]=-0.152206421 D[209]=-0.174789429 D[210]=-0.198059082 D[211]=-0.221984863
D[212]=-0.246505737 D[213]=-0.271591187 D[214]=-0.297210693 D[215]=-0.323318481
D[216]=-0.349868774 D[217]=-0.376800537 D[218]=-0.404083252 D[219]=-0.431655884
D[220]=-0.459472656 D[221]=-0.487472534 D[222]=-0.515609741 D[223]=-0.543823242
D[224]=-0.572036743 D[225]=-0.600219727 D[226]=-0.628295898 D[227]=-0.656219482
D[228]=-0.683914185 D[229]=-0.711318970 D[230]=-0.738372803 D[231]=-0.765029907
D[232]=-0.791213989 D[233]=-0.816864014 D[234]=-0.841949463 D[235]=-0.866363525
```

```
D[236]=-0.890090942 D[237]=-0.913055420 D[238]=-0.935195923 D[239]=-0.956481934
D[240]=-0.976852417 D[241]=-0.996246338 D[242]=-1.014617920 D[243]=-1.031936646
D[244]=-1.048156738 D[245]=-1.063217163 D[246]=-1.077117920 D[247]=-1.089782715
D[248]=-1.101211548 D[249]=-1.111373901 D[250]=-1.120223999 D[251]=-1.127746582
D[252]=-1.133926392 D[253]=-1.138763428 D[254]=-1.142211914 D[255]=-1.144287109
D[256]= 1.144989014 D[257]= 1.144287109 D[258]= 1.142211914 D[259]= 1.138763428
D[260]= 1.133926392 D[261]= 1.127746582 D[262]= 1.120223999 D[263]= 1.111373901
D[264]= 1.101211548 D[265]= 1.089782715 D[266]= 1.077117920 D[267]= 1.063217163
D[268]= 1.048156738 D[269]= 1.031936646 D[270]= 1.014617920 D[271]= 0.996246338
D[272]= 0.976852417 D[273]= 0.956481934 D[274]= 0.935195923 D[275]= 0.913055420
D[276]= 0.890090942 D[277]= 0.866363525 D[278]= 0.841949463 D[279]= 0.816864014
D[280]= 0.791213989 D[281]= 0.765029907 D[282]= 0.738372803 D[283]= 0.711318970
D[284]= 0.683914185 D[285]= 0.656219482 D[286]= 0.628295898 D[287]= 0.600219727
D[288]= 0.572036743 D[289]= 0.543823242 D[290]= 0.515609741 D[291]= 0.487472534
D[292]= 0.459472656 D[293]= 0.431655884 D[294]= 0.404083252 D[295]= 0.376800537
D[296]= 0.349868774 D[297]= 0.323318481 D[298]= 0.297210693 D[299]= 0.271591187
D[300]= 0.246505737 D[301]= 0.221984863 D[302]= 0.198059082 D[303]= 0.174789429
D[304]= 0.152206421 D[305]= 0.130310059 D[306]= 0.109161377 D[307]= 0.088775635
D[308]= 0.069168091 D[309]= 0.050354004 D[310]= 0.032379150 D[311]= 0.015228271
D[312]=-0.001068115 D[313]=-0.016510010 D[314]=-0.031082153 D[315]=-0.044784546
D[316]=-0.057617187 D[317]=-0.069595337 D[318]=-0.080688477 D[319]=-0.090927124
D[320]= 0.100311279 D[321]= 0.108856201 D[322]= 0.116577148 D[323]= 0.123474121
D[324]= 0.129577637 D[325]= 0.134887695 D[326]= 0.139450073 D[327]= 0.143264771
D[328]= 0.146362305 D[329]= 0.148773193 D[330]= 0.150497437 D[331]= 0.151596069
D[332]= 0.152069092 D[333]= 0.151962280 D[334]= 0.151306152 D[335]= 0.150115967
D[336]= 0.148422241 D[337]= 0.146255493 D[338]= 0.143676758 D[339]= 0.140670776
D[340]= 0.137298584 D[341]= 0.133590698 D[342]= 0.129562378 D[343]= 0.125259399
D[344]= 0.120697021 D[345]= 0.115921021 D[346]= 0.110946655 D[347]= 0.105819702
D[348]= 0.100540161 D[349]= 0.095169067 D[350]= 0.089706421 D[351]= 0.084182739
D[352]= 0.078628540 D[353]= 0.073059082 D[354]= 0.067520142 D[355]= 0.061996460
D[356]= 0.056533813 D[357]= 0.051132202 D[358]= 0.045837402 D[359]= 0.040634155
D[360]= 0.035552979 D[361]= 0.030609131 D[362]= 0.025817871 D[363]= 0.021179199
D[364]= 0.016708374 D[365]= 0.012420654 D[366]= 0.008316040 D[367]= 0.004394531
D[368]= 0.000686646 D[369]=-0.002822876 D[370]=-0.006134033 D[371]=-0.009231567
D[372]=-0.012115479 D[373]=-0.014801025 D[374]=-0.017257690 D[375]=-0.019531250
D[376]=-0.021575928 D[377]=-0.023422241 D[378]=-0.025085449 D[379]=-0.026535034
D[380]=-0.027801514 D[381]=-0.028884888 D[382]=-0.029785156 D[383]=-0.030517578
D[384]= 0.031082153 D[385]= 0.031478882 D[386]= 0.031738281 D[387]= 0.031845093
D[388]= 0.031814575 D[389]= 0.031661987 D[390]= 0.031387329 D[391]= 0.031005859
D[392]= 0.030532837 D[393]= 0.029937744 D[394]= 0.029281616 D[395]= 0.028533936
D[396]= 0.027725220 D[397]= 0.026840210 D[398]= 0.025909424 D[399]= 0.024932861
D[400]= 0.023910522 D[401]= 0.022857666 D[402]= 0.021789551 D[403]= 0.020690918
D[404]= 0.019577026 D[405]= 0.018463135 D[406]= 0.017349243 D[407]= 0.016235352
D[408]= 0.015121460 D[409]= 0.014022827 D[410]= 0.012939453 D[411]= 0.011886597
D[412]= 0.010848999 D[413]= 0.009841919 D[414]= 0.008865356 D[415]= 0.007919312
D[416]= 0.007003784 D[417]= 0.006118774 D[418]= 0.005294800 D[419]= 0.004486084
D[420]= 0.003723145 D[421]= 0.003005981 D[422]= 0.002334595 D[423]= 0.001693726
D[424]= 0.001098633 D[425]= 0.000549316 D[426]= 0.000030518 D[427]=-0.000442505
D[428]=-0.000869751 D[429]=-0.001266479 D[430]=-0.001617432 D[431]=-0.001937866
D[432]=-0.002227783 D[433]=-0.002487183 D[434]=-0.002700806 D[435]=-0.002883911
D[436]=-0.003051758 D[437]=-0.003173828 D[438]=-0.003280640 D[439]=-0.003372192
D[440]=-0.003417969 D[441]=-0.003463745 D[442]=-0.003479004 D[443]=-0.003479004
D[444]=-0.003463745 D[445]=-0.003433228 D[446]=-0.003387451 D[447]=-0.003326416
D[448]= 0.003250122 D[449]= 0.003173828 D[450]= 0.003082275 D[451]= 0.002990723
D[452]= 0.002899170 D[453]= 0.002792358 D[454]= 0.002685547 D[455]= 0.002578735
D[456]= 0.002456665 D[457]= 0.002349854 D[458]= 0.002243042 D[459]= 0.002120972
D[460]= 0.002014160 D[461]= 0.001907349 D[462]= 0.001785278 D[463]= 0.001693726
D[464]= 0.001586914 D[465]= 0.001480103 D[466]= 0.001388550 D[467]= 0.001296997
D[468]= 0.001205444 D[469]= 0.001113892 D[470]= 0.001037598 D[471]= 0.000961304
D[472]= 0.000885010 D[473]= 0.000808716 D[474]= 0.000747681 D[475]= 0.000686646
D[476]= 0.000625610 D[477]= 0.000579834 D[478]= 0.000534058 D[479]= 0.000473022
D[480]= 0.000442505 D[481]= 0.000396729 D[482]= 0.000366211 D[483]= 0.000320435
D[484]= 0.000289917 D[485]= 0.000259399 D[486]= 0.000244141 D[487]= 0.000213623
D[488]= 0.000198364 D[489]= 0.000167847 D[490]= 0.000152588 D[491]= 0.000137329
D[492]= 0.000122070 D[493]= 0.000106812 D[494]= 0.000106812 D[495]= 0.000091553
D[496]= 0.000076294 D[497]= 0.000076294 D[498]= 0.000061035 D[499]= 0.000061035
D[500]= 0.000045776 D[501]= 0.000045776 D[502]= 0.000030518 D[503]= 0.000030518
D[504]= 0.000030518 D[505]= 0.000030518 D[506]= 0.000015259 D[507]= 0.000015259
D[508]= 0.000015259 D[509]= 0.000015259 D[510]= 0.000015259 D[511]= 0.000015259
```

C.2.22 huffdec

```
##########################################################################
# huffman decode tree for ISO-MPEG Layer 3 Decoding                      #
# Syntax:                                                                #
```

```
# .table number treelen xlen ylen linbits:                              #
# .reference number of reference tree:                                  #
# or:                                                                   #
# .treedata                                                             #
# decodertree values in hex format:                                     #
#########################################################################

.table  0   0  0  0  0
.treedata

.table  1   7  2  2  0
.treedata
 2  1  0  0  2  1  0 10  2  1  0  1  0 11

.table  2  17  3  3  0
.treedata
 2  1  0  0  4  1  2  1  0 10  0  1  2  1  0 11  4  1  2  1  0 20  0 21
 2  1  0 12  2  1  0  2  0 22

.table  3  17  3  3  0
.treedata
 4  1  2  1  0  0  0  1  2  1  0 11  2  1  0 10  4  1  2  1  0 20  0 21
 2  1  0 12  2  1  0  2  0 22

.table  4   0  0  0  0
.treedata

.table  5  31  4  4  0
.treedata
 2  1  0  0  4  1  2  1  0 10  0  1  2  1  0 11  8  1  4  1  2  1  0 20
 0  2  2  1  0 21  0 12  8  1  4  1  2  1  0 22  0 30  2  1  0  3  0 13
 2  1  0 31  2  1  0 32  2  1  0 23  0 33

.table  6  31  4  4  0
.treedata
 6  1  4  1  2  1  0  0  0 10  0 11  6  1  2  1  0  1  2  1  0 20  0 21
 6  1  2  1  0 12  2  1  0  2  0 22  4  1  2  1  0 31  0 13  4  1  2  1
 0 30  0 32  2  1  0 23  2  1  0  3  0 33

.table  7  71  6  6  0
.treedata
 2  1  0  0  4  1  2  1  0 10  0  1  8  1  2  1  0 11  4  1  2  1  0 20
 0  2  0 21 12  1  6  1  2  1  0 12  2  1  0 22  0 30  4  1  2  1  0 31
 0 13  4  1  2  1  0  3  0 32  2  1  0 23  0  4  a  1  4  1  2  1  0 40
 0 41  2  1  0 14  2  1  0 42  0 24  c  1  6  1  4  1  2  1  0 33  0 43
 0 50  4  1  2  1  0 34  0  5  0 51  6  1  2  1  0 15  2  1  0 52  0 25
 4  1  2  1  0 44  0 35  4  1  2  1  0 53  0 54  2  1  0 45  0 55

.table  8  71  6  6  0
.treedata
 6  1  2  1  0  0  2  1  0 10  0  1  2  1  0 11  4  1  2  1  0 21  0 12
 e  1  4  1  2  1  0 20  0  2  2  1  0 22  4  1  2  1  0 30  0  3  2  1
 0 31  0 13  e  1  8  1  4  1  2  1  0 32  0 23  2  1  0 40  0  4  2  1
 0 41  2  1  0 14  0 42  c  1  6  1  2  1  0 24  2  1  0 33  0 50  4  1
 2  1  0 43  0 34  0 51  6  1  2  1  0 15  2  1  0  5  0 52  6  1  2  1
 0 25  2  1  0 44  0 35  2  1  0 53  2  1  0 45  2  1  0 54  0 55

.table  9  71  6  6  0
.treedata
 8  1  4  1  2  1  0  0  0 10  2  1  0  1  0 11  a  1  4  1  2  1  0 20
 0 21  2  1  0 12  2  1  0  2  0 22  c  1  6  1  4  1  2  1  0 30  0  3
 0 31  2  1  0 13  2  1  0 32  0 23  c  1  4  1  2  1  0 41  0 14  4  1
 2  1  0 40  0 33  2  1  0 42  0 24  a  1  6  1  4  1  2  1  0  4  0 50
 0 43  2  1  0 34  0 51  8  1  4  1  2  1  0 15  0 52  2  1  0 25  0 44
 6  1  4  1  2  1  0  5  0 54  0 53  2  1  0 35  2  1  0 45  0 55

.table 10 127  8  8  0
.treedata
 2  1  0  0  4  1  2  1  0 10  0  1  a  1  2  1  0 11  4  1  2  1  0 20
 0  2  2  1  0 21  0 12 1c  1  8  1  4  1  2  1  0 22  0 30  2  1  0 31
 0 13  8  1  4  1  2  1  0  3  0 32  2  1  0 23  0 40  4  1  2  1  0 41
 0 14  4  1  2  1  0  4  0 33  2  1  0 42  0 24 1c  1  a  1  6  1  4  1
 2  1  0 50  0  5  0 60  2  1  0 61  0 16  c  1  6  1  4  1  2  1  0 43
 0 34  0 51  2  1  0 15  2  1  0 52  0 25  4  1  2  1  0 26  0 36  0 71
14  1  8  1  2  1  0 17  4  1  2  1  0 44  0 53  0  6  6  1  4  1  2  1
 0 35  0 45  0 62  2  1  0 70  2  1  0  7  0 64  e  1  4  1  2  1  0 72
```

```
 0 27  6  1  2  1  0 63  2  1  0 54  0 55  2  1  0 46  0 73  8  1  4  1
 2  1  0 37  0 65  2  1  0 56  0 74  6  1  2  1  0 47  2  1  0 66  0 75
 4  1  2  1  0 57  0 76  2  1  0 67  0 77

.table 11 127  8  8  0
.treedata
 6  1  2  1  0  0  2  1  0 10  0  1  8  1  2  1  0 11  4  1  2  1  0 20
 0  2  0 12 18  1  8  1  2  1  0 21  2  1  0 22  2  1  0 30  0  3  4  1
 2  1  0 31  0 13  4  1  2  1  0 32  0 23  4  1  2  1  0 40  0  4  2  1
 0 41  0 14 1e  1 10  1  a  1  4  1  2  1  0 42  0 24  4  1  2  1  0 33
 0 43  0 50  4  1  2  1  0 34  0 51  0 61  6  1  2  1  0 16  2  1  0  6
 0 26  2  1  0 62  2  1  0 15  2  1  0  5  0 52 10  1  a  1  6  1  4  1
 2  1  0 25  0 44  0 60  2  1  0 63  0 36  4  1  2  1  0 70  0 17  0 71
10  1  6  1  4  1  2  1  0  7  0 64  0 72  2  1  0 27  4  1  2  1  0 53
 0 35  2  1  0 54  0 45  a  1  4  1  2  1  0 46  0 73  2  1  0 37  2  1
 0 65  0 56  a  1  6  1  4  1  2  1  0 55  0 57  0 74  2  1  0 47  0 66
 4  1  2  1  0 75  0 76  2  1  0 67  0 77

.table 12 127  8  8  0
.treedata
 c  1  4  1  2  1  0 10  0  1  2  1  0 11  2  1  0  0  2  1  0 20  0  2
10  1  4  1  2  1  0 21  0 12  4  1  2  1  0 22  0 31  2  1  0 13  2  1
 0 30  2  1  0  3  0 40 1a  1  8  1  4  1  2  1  0 32  0 23  2  1  0 41
 0 33  a  1  4  1  2  1  0 14  0 42  2  1  0 24  2  1  0  4  0 50  4  1
 2  1  0 43  0 34  2  1  0 51  0 15 1c  1  e  1  8  1  4  1  2  1  0 52
 0 25  2  1  0 53  0 35  4  1  2  1  0 60  0 16  0 61  4  1  2  1  0 62
 0 26  6  1  4  1  2  1  0  5  0  6  0 44  2  1  0 54  0 45 12  1  a  1
 4  1  2  1  0 63  0 36  4  1  2  1  0 70  0  7  0 71  4  1  2  1  0 17
 0 64  2  1  0 46  0 72  a  1  6  1  2  1  0 27  2  1  0 55  0 73  2  1
 0 37  0 56  8  1  4  1  2  1  0 65  0 74  2  1  0 47  0 66  4  1  2  1
 0 75  0 57  2  1  0 76  2  1  0 67  0 77

.table 13 511 16 16  0
.treedata
 2  1  0  0  6  1  2  1  0 10  2  1  0  1  0 11 1c  1  8  1  4  1  2  1
 0 20  0  2  2  1  0 21  0 12  8  1  4  1  2  1  0 22  0 30  2  1  0  3
 0 31  6  1  2  1  0 13  2  1  0 32  0 23  4  1  2  1  0 40  0  4  0 41
46  1 1c  1  e  1  6  1  2  1  0 14  2  1  0 33  0 42  4  1  2  1  0 24
 0 50  2  1  0 43  0 34  4  1  2  1  0 51  0 15  4  1  2  1  0  5  0 52
 2  1  0 25  2  1  0 44  0 53  e  1  8  1  4  1  2  1  0 60  0  6  2  1
 0 61  0 16  4  1  2  1  0 80  0  8  0 81 10  1  8  1  4  1  2  1  0 35
 0 62  2  1  0 26  0 54  4  1  2  1  0 45  0 63  2  1  0 36  0 70  6  1
 4  1  2  1  0  7  0 55  0 71  2  1  0 17  2  1  0 27  0 37 48  1 18  1
 c  1  4  1  2  1  0 18  0 82  2  1  0 28  4  1  2  1  0 64  0 46  0 72
 8  1  4  1  2  1  0 84  0 48  2  1  0 90  0  9  2  1  0 91  0 19 18  1
 e  1  8  1  4  1  2  1  0 73  0 65  2  1  0 56  0 74  4  1  2  1  0 47
 0 66  0 83  6  1  2  1  0 38  2  1  0 75  0 57  2  1  0 92  0 29  e  1
 8  1  4  1  2  1  0 67  0 85  2  1  0 58  0 39  2  1  0 93  2  1  0 49
 0 86  6  1  2  1  0 a0  2  1  0 68  0  a  2  1  0 a1  0 1a 44  1 18  1
 c  1  4  1  2  1  0 a2  0 2a  4  1  2  1  0 95  0 59  2  1  0 a3  0 3a
 8  1  4  1  2  1  0 4a  0 96  2  1  0 b0  0  b  2  1  0 b1  0 1b 14  1
 8  1  2  1  0 b2  4  1  2  1  0 76  0 77  0 94  6  1  4  1  2  1  0 87
 0 78  0 a4  4  1  2  1  0 69  0 a5  0 2b  c  1  6  1  4  1  2  1  0 5a
 0 88  0 b3  2  1  0 3b  2  1  0 79  0 a6  6  1  4  1  2  1  0 6a  0 b4
 0 c0  4  1  2  1  0  c  0 98  0 c1 3c  1 16  1  a  1  6  1  2  1  0 1c
 2  1  0 89  0 b5  2  1  0 5b  0 c2  4  1  2  1  0 2c  0 3c  4  1  2  1
 0 b6  0 6b  2  1  0 c4  0 4c 10  1  8  1  4  1  2  1  0 a8  0 8a  2  1
 0 d0  0  d  2  1  0 d1  2  1  0 4b  2  1  0 97  0 a7  c  1  6  1  2  1
 0 c3  2  1  0 7a  0 99  4  1  2  1  0 c5  0 5c  0 b7  4  1  2  1  0 1d
 0 d2  2  1  0 2d  2  1  0 7b  0 d3 34  1 1c  1  c  1  4  1  2  1  0 3d
 0 c6  4  1  2  1  0 6c  0 a9  2  1  0 9a  0 d4  8  1  4  1  2  1  0 b8
 0 8b  2  1  0 4d  0 c7  4  1  2  1  0 7c  0 d5  2  1  0 5d  0 e0  a  1
 4  1  2  1  0 e1  0 1e  4  1  2  1  0  e  0 2e  0 e2  8  1  4  1  2  1
 0 e3  0 6d  2  1  0 8c  0 e4  4  1  2  1  0 e5  0 ba  0 f0 26  1 10  1
 4  1  2  1  0 f1  0 1f  6  1  4  1  2  1  0 aa  0 9b  0 b9  2  1  0 3e
 2  1  0 d6  0 c8  c  1  6  1  2  1  0 4e  2  1  0 d7  0 7d  2  1  0 ab
 2  1  0 5e  0 c9  6  1  2  1  0  f  2  1  0 9c  0 6e  2  1  0 f2  0 2f
20  1 10  1  6  1  4  1  2  1  0 d8  0 8d  0 3f  6  1  2  1  0 f3  2  1
 0 e6  0 ca  2  1  0 f4  0 4f  8  1  4  1  2  1  0 bb  0 ac  2  1  0 e7
 0 f5  4  1  2  1  0 d9  0 9d  2  1  0 5f  0 e8 1e  1  c  1  6  1  2  1
 0 6f  2  1  0 f6  0 cb  4  1  2  1  0 bc  0 ad  0 da  8  1  2  1  0 f7
 4  1  2  1  0 7e  0 7f  0 8e  6  1  4  1  2  1  0 9e  0 ae  0 cc  2  1
 0 f8  0 8f 12  1  8  1  4  1  2  1  0 db  0 bd  2  1  0 ea  0 f9  4  1
 2  1  0 9f  0 eb  2  1  0 be  2  1  0 cd  0 fa  e  1  4  1  2  1  0 dd
 0 ec  6  1  4  1  2  1  0 e9  0 af  0 dc  2  1  0 ce  0 fb  8  1  4  1
 2  1  0 bf  0 de  2  1  0 cf  0 ee  4  1  2  1  0 df  0 ef  2  1  0 ff
```

```
 2  1  0 ed  2  1  0 fd  2  1  0 fc  0 fe

.table 14   0  0  0  0
.treedata

.table 15 511 16 16  0
.treedata
10  1  6  1  2  1  0  0  2  1  0 10  0  1  2  1  0 11  4  1  2  1  0 20
 0  2  2  1  0 21  0 12 32  1 10  1  6  1  2  1  0 22  2  1  0 30  0 31
 6  1  2  1  0 13  2  1  0  3  0 40  2  1  0 32  0 23  e  1  6  1  4  1
 2  1  0  4  0 14  0 41  4  1  2  1  0 33  0 42  2  1  0 24  0 43  a  1
 6  1  2  1  0 34  2  1  0 50  0  5  2  1  0 51  0 15  4  1  2  1  0 52
 0 25  4  1  2  1  0 44  0 53  0 61 5a  1 24  1 12  1  a  1  6  1  2  1
 0 35  2  1  0 60  0  6  2  1  0 16  0 62  4  1  2  1  0 26  0 54  2  1
 0 45  0 63  a  1  6  1  2  1  0 36  2  1  0 70  0  7  2  1  0 71  0 55
 4  1  2  1  0 17  0 64  2  1  0 72  0 27 18  1 10  1  8  1  4  1  2  1
 0 46  0 73  2  1  0 37  0 65  4  1  2  1  0 56  0 80  2  1  0  8  0 74
 4  1  2  1  0 81  0 18  2  1  0 82  0 28 10  1  8  1  4  1  2  1  0 47
 0 66  2  1  0 83  0 38  4  1  2  1  0 75  0 57  2  1  0 84  0 48  6  1
 4  1  2  1  0 90  0 19  0 91  4  1  2  1  0 92  0 76  2  1  0 67  0 29
5c  1 24  1 12  1  a  1  4  1  2  1  0 85  0 58  4  1  2  1  0  9  0 77
 0 93  4  1  2  1  0 39  0 94  2  1  0 49  0 86  a  1  6  1  2  1  0 68
 2  1  0 a0  0  a  2  1  0 a1  0 1a  4  1  2  1  0 a2  0 2a  2  1  0 95
 0 59 1a  1  e  1  6  1  2  1  0 a3  2  1  0 3a  0 87  4  1  2  1  0 78
 0 a4  2  1  0 4a  0 96  6  1  4  1  2  1  0 69  0 b0  0 b1  4  1  2  1
 0 1b  0 a5  0 b2  e  1  8  1  4  1  2  1  0 5a  0 2b  2  1  0 88  0 97
 2  1  0 b3  2  1  0 79  0 3b  8  1  4  1  2  1  0 6a  0 b4  2  1  0 4b
 0 c1  4  1  2  1  0 98  0 89  2  1  0 1c  0 b5 50  1 22  1 10  1  6  1
 4  1  2  1  0 5b  0 2c  0 c2  6  1  4  1  2  1  0  b  0 c0  0 a6  2  1
 0 a7  0 7a  a  1  4  1  2  1  0 c3  0 3c  4  1  2  1  0  c  0 99  0 b6
 4  1  2  1  0 6b  0 c4  2  1  0 4c  0 a8 14  1  a  1  4  1  2  1  0 8a
 0 c5  4  1  2  1  0 d0  0 5c  0 d1  4  1  2  1  0 b7  0 7b  2  1  0 1d
 2  1  0  d  0 2d  c  1  4  1  2  1  0 d2  0 d3  4  1  2  1  0 3d  0 c6
 2  1  0 6c  0 a9  6  1  4  1  2  1  0 9a  0 b8  0 d4  4  1  2  1  0 8b
 0 4d  2  1  0 c7  0 7c 44  1 22  1 12  1  a  1  4  1  2  1  0 d5  0 5d
 4  1  2  1  0 e0  0  e  0 e1  4  1  2  1  0 1e  0 e2  2  1  0 aa  0 2e
 8  1  4  1  2  1  0 b9  0 9b  2  1  0 e3  0 d6  4  1  2  1  0 6d  0 3e
 2  1  0 c8  0 8c 10  1  8  1  4  1  2  1  0 e4  0 4e  2  1  0 d7  0 7d
 4  1  2  1  0 e5  0 ba  2  1  0 ab  0 5e  8  1  4  1  2  1  0 c9  0 9c
 2  1  0 f1  0 1f  6  1  4  1  2  1  0 f0  0 6e  0 f2  2  1  0 2f  0 e6
26  1 12  1  8  1  4  1  2  1  0 d8  0 f3  2  1  0 3f  0 f4  6  1  2  1
 0 4f  2  1  0 8d  0 d9  2  1  0 bb  0 ca  8  1  4  1  2  1  0 ac  0 e7
 2  1  0 7e  0 f5  8  1  4  1  2  1  0 9d  0 5f  2  1  0 e8  0 8e  2  1
 0 f6  0 cb 22  1 12  1  a  1  6  1  4  1  2  1  0  f  0 ae  0 6f  2  1
 0 bc  0 da  4  1  2  1  0 ad  0 f7  2  1  0 7f  0 e9  8  1  4  1  2  1
 0 9e  0 cc  2  1  0 f8  0 8f  4  1  2  1  0 db  0 bd  2  1  0 ea  0 f9
10  1  8  1  4  1  2  1  0 9f  0 dc  2  1  0 cd  0 eb  4  1  2  1  0 be
 0 fa  2  1  0 af  0 dd  e  1  6  1  4  1  2  1  0 ec  0 ce  0 fb  4  1
 2  1  0 bf  0 ed  2  1  0 de  0 fc  6  1  4  1  2  1  0 cf  0 fd  0 ee
 4  1  2  1  0 df  0 fe  2  1  0 ef  0 ff

.table 16 511 16 16  1
.treedata
 2  1  0  0  6  1  2  1  0 10  2  1  0  1  0 11 2a  1  8  1  4  1  2  1
 0 20  0  2  2  1  0 21  0 12  a  1  6  1  2  1  0 22  2  1  0 30  0  3
 2  1  0 31  0 13  a  1  4  1  2  1  0 32  0 23  4  1  2  1  0 40  0  4
 0 41  6  1  2  1  0 14  2  1  0 33  0 42  4  1  2  1  0 24  0 50  2  1
 0 43  0 34 8a  1 28  1 10  1  6  1  4  1  2  1  0  5  0 15  0 51  4  1
 2  1  0 52  0 25  4  1  2  1  0 44  0 35  0 53  a  1  6  1  4  1  2  1
 0 60  0  6  0 61  2  1  0 16  0 62  8  1  4  1  2  1  0 26  0 54  2  1
 0 45  0 63  4  1  2  1  0 36  0 70  0 71 28  1 12  1  8  1  2  1  0 17
 2  1  0  7  2  1  0 55  0 64  4  1  2  1  0 72  0 27  4  1  2  1  0 46
 0 65  0 73  a  1  6  1  2  1  0 37  2  1  0 56  0  8  2  1  0 80  0 81
 6  1  2  1  0 18  2  1  0 74  0 47  2  1  0 82  2  1  0 28  0 66 18  1
 e  1  8  1  4  1  2  1  0 83  0 38  2  1  0 75  0 84  4  1  2  1  0 48
 0 90  0 91  6  1  2  1  0 19  2  1  0  9  0 76  2  1  0 92  0 29  e  1
 8  1  4  1  2  1  0 85  0 58  2  1  0 93  0 39  4  1  2  1  0 a0  0  a
 0 1a  8  1  2  1  0 a2  2  1  0 67  2  1  0 57  0 49  6  1  2  1  0 94
 2  1  0 77  0 86  2  1  0 a1  2  1  0 68  0 95 dc  1 7e  1 32  1 1a  1
 c  1  6  1  2  1  0 2a  2  1  0 59  0 3a  2  1  0 a3  2  1  0 87  0 78
 8  1  4  1  2  1  0 a4  0 4a  2  1  0 96  0 69  4  1  2  1  0 b0  0  b
 0 b1  a  1  4  1  2  1  0 1b  0 b2  2  1  0 2b  2  1  0 a5  0 5a  6  1
 2  1  0 b3  2  1  0 a6  0 6a  4  1  2  1  0 b4  0 4b  2  1  0  c  0 c1
1e  1  e  1  6  1  4  1  2  1  0 b5  0 c2  0 2c  4  1  2  1  0 a7  0 c3
 2  1  0 6b  0 c4  8  1  2  1  0 1d  4  1  2  1  0 88  0 97  0 3b  4  1
 2  1  0 d1  0 d2  2  1  0 2d  0 d3 12  1  6  1  4  1  2  1  0 1e  0 2e
```

```
 0 e2  6  1  4  1  2  1  0 79  0 98  0 c0  2  1  0 1c  2  1  0 89  0 5b
 e  1  6  1  2  1  0 3c  2  1  0 7a  0 b6  4  1  2  1  0 4c  0 99  2  1
 0 a8  0 8a  6  1  2  1  0  d  2  1  0 c5  0 5c  4  1  2  1  0 3d  0 c6
 2  1  0 6c  0 9a 58  1 56  1 24  1 10  1  8  1  4  1  2  1  0 8b  0 4d
 2  1  0 c7  0 7c  4  1  2  1  0 d5  0 5d  2  1  0 e0  0  e  8  1  2  1
 0 e3  4  1  2  1  0 d0  0 b7  0 7b  6  1  4  1  2  1  0 a9  0 b8  0 d4
 2  1  0 e1  2  1  0 aa  0 b9 18  1  a  1  6  1  4  1  2  1  0 9b  0 d6
 0 6d  2  1  0 3e  0 c8  6  1  4  1  2  1  0 8c  0 e4  0 4e  4  1  2  1
 0 d7  0 e5  2  1  0 ba  0 ab  c  1  4  1  2  1  0 9c  0 e6  4  1  2  1
 0 6e  0 d8  2  1  0 8d  0 bb  8  1  4  1  2  1  0 e7  0 9d  2  1  0 e8
 0 8e  4  1  2  1  0 cb  0 bc  0 9e  0 f1  2  1  0 1f  2  1  0  f  0 2f
42  1 38  1  2  1  0 f2 34  1 32  1 14  1  8  1  2  1  0 bd  2  1  0 5e
 2  1  0 7d  0 c9  6  1  2  1  0 ca  2  1  0 ac  0 7e  4  1  2  1  0 da
 0 ad  0 cc  a  1  6  1  2  1  0 ae  2  1  0 db  0 dc  2  1  0 cd  0 be
 6  1  4  1  2  1  0 eb  0 ed  0 ee  6  1  4  1  2  1  0 d9  0 ea  0 e9
 2  1  0 de  4  1  2  1  0 dd  0 ec  0 ce  0 3f  0 f0  4  1  2  1  0 f3
 0 f4  2  1  0 4f  2  1  0 f5  0 5f  a  1  2  1  0 ff  4  1  2  1  0 f6
 0 6f  2  1  0 f7  0 7f  c  1  6  1  2  1  0 8f  2  1  0 f8  0 f9  4  1
 2  1  0 9f  0 fa  0 af  8  1  4  1  2  1  0 fb  0 bf  2  1  0 fc  0 cf
 4  1  2  1  0 fd  0 df  2  1  0 fe  0 ef

.table 17 511 16 16  2
.reference 16

.table 18 511 16 16  3
.reference 16

.table 19 511 16 16  4
.reference 16

.table 20 511 16 16  6
.reference 16

.table 21 511 16 16  8
.reference 16

.table 22 511 16 16 10
.reference 16

.table 23 511 16 16 13
.reference 16

.table 24 512 16 16  4
.treedata
3c  1  8  1  4  1  2  1  0  0  0 10  2  1  0  1  0 11  e  1  6  1  4  1
 2  1  0 20  0  2  0 21  2  1  0 12  2  1  0 22  2  1  0 30  0  3  e  1
 4  1  2  1  0 31  0 13  4  1  2  1  0 32  0 23  4  1  2  1  0 40  0  4
 0 41  8  1  4  1  2  1  0 14  0 33  2  1  0 42  0 24  6  1  4  1  2  1
 0 43  0 34  0 51  6  1  4  1  2  1  0 50  0  5  0 15  2  1  0 52  0 25
fa  1 62  1 22  1 12  1  a  1  4  1  2  1  0 44  0 53  2  1  0 35  2  1
 0 60  0  6  4  1  2  1  0 61  0 16  2  1  0 62  0 26  8  1  4  1  2  1
 0 54  0 45  2  1  0 63  0 36  4  1  2  1  0 71  0 55  2  1  0 64  0 46
20  1  e  1  6  1  2  1  0 72  2  1  0 27  0 37  2  1  0 73  4  1  2  1
 0 70  0  7  0 17  a  1  4  1  2  1  0 65  0 56  4  1  2  1  0 80  0  8
 0 81  4  1  2  1  0 74  0 47  2  1  0 18  0 82 10  1  8  1  4  1  2  1
 0 28  0 66  2  1  0 83  0 38  4  1  2  1  0 75  0 57  2  1  0 84  0 48
 8  1  4  1  2  1  0 91  0 19  2  1  0 92  0 76  4  1  2  1  0 67  0 29
 2  1  0 85  0 58 5c  1 22  1 10  1  8  1  4  1  2  1  0 93  0 39  2  1
 0 94  0 49  4  1  2  1  0 77  0 86  2  1  0 68  0 a1  8  1  4  1  2  1
 0 a2  0 2a  2  1  0 95  0 59  4  1  2  1  0 a3  0 3a  2  1  0 87  2  1
 0 78  0 4a 16  1  c  1  4  1  2  1  0 a4  0 96  4  1  2  1  0 69  0 b1
 2  1  0 1b  0 a5  6  1  2  1  0 b2  2  1  0 5a  0 2b  2  1  0 88  0 b3
10  1  a  1  6  1  2  1  0 90  2  1  0  9  0 a0  2  1  0 97  0 79  4  1
 2  1  0 a6  0 6a  0 b4  c  1  6  1  2  1  0 1a  2  1  0  a  0 b0  2  1
 0 3b  2  1  0  b  0 c0  4  1  2  1  0 4b  0 c1  2  1  0 98  0 89 43  1
22  1 10  1  8  1  4  1  2  1  0 1c  0 b5  2  1  0 5b  0 c2  4  1  2  1
 0 2c  0 a7  2  1  0 7a  0 c3  a  1  6  1  2  1  0 3c  2  1  0  c  0 d0
 2  1  0 b6  0 6b  4  1  2  1  0 c4  0 4c  2  1  0 99  0 a8 10  1  8  1
 4  1  2  1  0 8a  0 c5  2  1  0 5c  0 d1  4  1  2  1  0 b7  0 7b  2  1
 0 1d  0 d2  9  1  4  1  2  1  0 2d  0 d3  2  1  0 3d  0 c6 55 fa  4  1
 2  1  0 6c  0 a9  2  1  0 9a  0 d4 20  1 10  1  8  1  4  1  2  1  0 b8
 0 8b  2  1  0 4d  0 c7  4  1  2  1  0 7c  0 d5  2  1  0 5d  0 e1  8  1
 4  1  2  1  0 1e  0 e2  2  1  0 aa  0 b9  4  1  2  1  0 9b  0 e3  2  1
 0 d6  0 6d 14  1  a  1  6  1  2  1  0 3e  2  1  0 2e  0 4e  2  1  0 c8
 0 8c  4  1  2  1  0 e4  0 d7  4  1  2  1  0 7d  0 ab  0 e5  a  1  4  1
 2  1  0 ba  0 5e  2  1  0 c9  2  1  0 9c  0 6e  8  1  2  1  0 e6  2  1
```

```
 0  d  2  1  0 e0  0  e  4  1  2  1  0 d8  0 8d  2  1  0 bb  0 ca 4a  1
 2  1  0 ff 40  1 3a  1 20  1 10  1  8  1  4  1  2  1  0 ac  0 e7  2  1
 0 7e  0 d9  4  1  2  1  0 9d  0 e8  2  1  0 8e  0 cb  8  1  4  1  2  1
 0 bc  0 da  2  1  0 ad  0 e9  4  1  2  1  0 9e  0 cc  2  1  0 db  0 bd
10  1  8  1  4  1  2  1  0 ea  0 ae  2  1  0 dc  0 cd  4  1  2  1  0 eb
 0 be  2  1  0 dd  0 ec  8  1  4  1  2  1  0 ce  0 ed  2  1  0 de  0 ee
 0  f  4  1  2  1  0 f0  0 1f  0 f1  4  1  2  1  0 f2  0 2f  2  1  0 f3
 0 3f 12  1  8  1  4  1  2  1  0 f4  0 4f  2  1  0 f5  0 5f  4  1  2  1
 0 f6  0 6f  2  1  0 f7  2  1  0 7f  0 8f  a  1  4  1  2  1  0 f8  0 f9
 4  1  2  1  0 9f  0 af  0 fa  8  1  4  1  2  1  0 fb  0 bf  2  1  0 fc
 0 cf  4  1  2  1  0 fd  0 df  2  1  0 fe  0 ef

.table 25 512 16 16  5
.reference 24

.table 26 512 16 16  6
.reference 24

.table 27 512 16 16  7
.reference 24

.table 28 512 16 16  8
.reference 24

.table 29 512 16 16  9
.reference 24

.table 30 512 16 16 11
.reference 24

.table 31 512 16 16 13
.reference 24

.table 32  31  1 16  0
.treedata
 2  1  0  0  8  1  4  1  2  1  0  8  0  4  2  1  0  1  0  2  8  1  4  1
 2  1  0  c  0  a  2  1  0  3  0  6  6  1  2  1  0  9  2  1  0  5  0  7
 4  1  2  1  0  e  0  d  2  1  0  f  0  b

.table 33  31  1 16  0
.treedata
10  1  8  1  4  1  2  1  0  0  0  1  2  1  0  2  0  3  4  1  2  1  0  4
 0  5  2  1  0  6  0  7  8  1  4  1  2  1  0  8  0  9  2  1  0  a  0  b
 4  1  2  1  0  c  0  d  2  1  0  e  0  f

.end
```

C.3 Encoder and decoder

C.3.1 common.c

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
common.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Doulas Wong,        start of version 1.0 records          *
 *          Davis Pan                                                 *
 * 5/10/91  W. Joseph Carter    Created this file for all common      *
 *                              functions and global variables.       *
 *                              Ported to Macintosh and Unix.         *
 *                              Added Jean-Georges Fritsch's          *
 *                              "bitstream.c" package.                *
 *                              Added routines to handle AIFF PCM     *
```

```
 *                              sound files.                      *
 *                              Added "mem_alloc()" and "mem_free()"  *
 *                              routines for memory allocation    *
 *                              portability.                      *
 *                              Added routines to convert between *
 *                              Apple SANE extended floating point *
 *                              format and IEEE double precision  *
 *                              floating point format.  For AIFF. *
 * 02jul91 dpwe (Aware Inc)     Moved allocation table input here; *
 *                              Tables read from subdir TABLES_PATH. *
 *                              Added some debug printout fns (Write*)*
 * 7/10/91 Earle Jennings       replacement of the one float by FLOAT *
 *                              port to MsDos from MacIntosh version *
 * 8/ 5/91 Jean-Georges Fritsch fixed bug in open_bit_stream_r()  *
 *10/ 1/91 S.I. Sudharsanan,    Ported to IBM AIX platform.       *
 *         Don H. Lee,                                            *
 *         Peter W. Farrett                                       *
 *10/3/91  Don H. Lee           implemented CRC-16 error protection *
 *                              newly introduced functions are    *
 *                              I_CRC_calc, II_CRC_calc and       *
 *                              update_CRC. Additions and revisions *
 *                              are marked with dhl for clarity   *
 *10/18/91 Jean-Georges Fritsch fixed bug in update_CRC(),        *
 *                              II_CRC_calc() and I_CRC_calc()    *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most *
 *                              important fixes involved changing *
 *                              16-bit ints to long or unsigned in *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args. *
 *                              Removed "Other Joint Stereo" option *
 *                              and made bitrate be total channel *
 *                              bitrate, irrespective of the mode. *
 *                              Fixed many small bugs & reorganized. *
 * 3/20/92 Jean-Georges Fritsch  fixed bug in start-of-frame search *
 * 6/15/92 Juan Pineda          added refill_buffer(bs) "n"       *
 *                              initialization                    *
 * 7/08/92 Susanne Ritscher     MS-DOS, MSC6.0 port fixes         *
 * 7/27/92 Mike Li               (re-)Port to MS-DOS              *
 * 8/19/92 Soren H. Nielsen     Fixed bug in I_CRC_calc and in    *
 *                              II_CRC_calc.  Added function: new_ext *
 *                              for better MS-DOS compatability   *
 * 3/10/93 Kevin Peterson       changed aiff_read_headers to handle *
 *                              chunks in any order.  now returns *
 *                              position of sound data in file.   *
 * 3/31/93 Jens Spille          changed IFF_* string compares to use *
 *                              strcmp()                          *
 * 5/30/93 Masahiro Iwadare     removed the previous modification *
 *                              for UNIX.                         *
 * 8/27/93 Seymour Shlien,      Fixes in Unix and MSDOS ports,    *
 *         Daniel Lauzon, and                                     *
 *         Bill Truerniet                                         *
 *----------------------------------------------------------------*
 *  8/24/93 Masahiro Iwadare    Included IS modification in Layer III.*
 *                              Changed for 1 pass decoding.      *
 *  9/07/93 Toshiyuki Ishino    Integrated Layer III with Ver 3.9. *
 *----------------------------------------------------------------*
 * 11/20/93 Masahiro Iwadare    Integrated Layer III with Ver 4.0. *
 *----------------------------------------------------------------*
 *  7/14/94 Juergen Koller      rewind of bitbuffer added         *
 ******************************************************************/

/*******************************************************************
*
*  Global Include Files
*
*******************************************************************/

#include        "common.h"

#ifdef  MACINTOSH

#include        <SANE.h>
#include        <pascal.h>

#endif
```

```
#include <ctype.h>

/***********************************************************************
*
*  Global Variable Definitions
*
***********************************************************************/

char *mode_names[4] = { "stereo", "j-stereo", "dual-ch", "single-ch" };
char *layer_names[3] = { "I", "II", "III" };

double  s_freq[4] = {44.1, 48, 32, 0};

int     bitrate[3][15] = {
          {0,32,64,96,128,160,192,224,256,288,320,352,384,416,448},
          {0,32,48,56,64,80,96,112,128,160,192,224,256,320,384},
          {0,32,40,48,56,64,80,96,112,128,160,192,224,256,320}
        };

double FAR multiple[64] = {
2.00000000000000, 1.58740105196820, 1.25992104989487,
1.00000000000000, 0.79370052598410, 0.62996052494744, 0.50000000000000,
0.39685026299205, 0.31498026247372, 0.25000000000000, 0.19842513149602,
0.15749013123686, 0.12500000000000, 0.09921256574801, 0.07874506561843,
0.06250000000000, 0.04960628287401, 0.03937253280921, 0.03125000000000,
0.02480314143700, 0.01968626640461, 0.01562500000000, 0.01240157071850,
0.00984313320230, 0.00781250000000, 0.00620078535925, 0.00492156660115,
0.00390625000000, 0.00310039267963, 0.00246078330058, 0.00195312500000,
0.00155019633981, 0.00123039165029, 0.00097656250000, 0.00077509816991,
0.00061519582514, 0.00048828125000, 0.00038754908495, 0.00030759791257,
0.00024414062500, 0.00019377454248, 0.00015379895629, 0.00012207031250,
0.00009688727124, 0.00007689947814, 0.00006103515625, 0.00004844363562,
0.00003844973907, 0.00003051757813, 0.00002422181781, 0.00001922486954,
0.00001525878906, 0.00001211090890, 0.00000961243477, 0.00000762939453,
0.00000605545445, 0.00000480621738, 0.00000381469727, 0.00000302772723,
0.00000240310869, 0.00000190734863, 0.00000151386361, 0.00000120155435,
1E-20
};

/***********************************************************************
*
*  Global Function Definitions
*
***********************************************************************/

/* The system uses a variety of data files.  By opening them via this
   function, we can accommodate various locations. */

FILE *OpenTableFile(name)
char *name;
{
char fulname[80];
char *envdir;
FILE *f;

     fulname[0] = '\0';

#ifdef TABLES_PATH
       strcpy(fulname, TABLES_PATH);   /* default relative path for tables */
#endif /* TABLES_PATH */          /* (includes terminal path seperator */

#ifdef UNIX                       /* envir. variables for UNIX only */
       {
        char *getenv();

        envdir = getenv(MPEGTABENV);   /* check for environment */
        if(envdir != NULL)
            strcpy(fulname, envdir);
        strcat(fulname, PATH_SEPARATOR);  /* add a "/" on the end */
      }
#endif /* UNIX */

    strcat(fulname, name);
    if( (f=fopen(fulname,"r"))==NULL ) {
        fprintf(stderr,"OpenTable: could not find %s\n", fulname);
```

```
#ifdef UNIX
          if(envdir != NULL)
            fprintf(stderr,"Check %s directory '%s'\n",MPEGTABENV, envdir);
          else
            fprintf(stderr,"Check local directory './%s' or setenv %s\n",
                    TABLES_PATH, MPEGTABENV);
#else /* not unix : no environment variables */

#ifdef TABLES_PATH
            fprintf(stderr,"Check local directory './%s'\n",TABLES_PATH);
#endif /* TABLES_PATH */

#endif /* UNIX */

    }
    return f;
}

/************************************************************************
/*
/* Read one of the data files ("alloc_*") specifying the bit allocation/
/* quatization parameters for each subband in layer II encoding
/*
/************************************************************************/

int read_bit_alloc(table, alloc)        /* read in table, return # subbands */
int table;
al_table *alloc;
{
        unsigned int a, b, c, d, i, j;
        FILE *fp;
        char name[16], t[80];
        int sblim;

        strcpy(name, "alloc_0");

        switch (table) {
                case 0 : name[6] = '0';         break;
                case 1 : name[6] = '1';         break;
                case 2 : name[6] = '2';         break;
                case 3 : name[6] = '3';         break;
                default : name[6] = '0';
        }

        if (!(fp = OpenTableFile(name))) {
                printf("Please check bit allocation table %s\n", name);
                exit(1);
        }

        printf("using bit allocation table %s\n", name);

        fgets(t, 80, fp);
        sscanf(t, "%d\n", &sblim);
        while (!feof(fp)) {
                fgets(t, 80, fp);
                sscanf(t, "%d %d %d %d %d %d\n", &i, &j, &a, &b, &c, &d);
                        (*alloc)[i][j].steps = a;
                        (*alloc)[i][j].bits  = b;
                        (*alloc)[i][j].group = c;
                        (*alloc)[i][j].quant = d;
        }
        fclose(fp);
        return sblim;
}

/************************************************************************
/*
/* Using the decoded info the appropriate possible quantization per
/* subband table is loaded
/*
/************************************************************************/

int pick_table(fr_ps)   /* choose table, load if necess, return # sb's */
frame_params *fr_ps;
{
        int table, lay, ws, bsp, br_per_ch, sfrq;
```

```
        int sblim = fr_ps->sblimit;     /* return current value if no load */

        lay = fr_ps->header->lay - 1;
        bsp = fr_ps->header->bitrate_index;
        br_per_ch = bitrate[lay][bsp] / fr_ps->stereo;
        ws = fr_ps->header->sampling_frequency;
        sfrq = s_freq[ws];
        /* decision rules refer to per-channel bitrates (kbits/sec/chan) */
        if ((sfrq == 48 && br_per_ch >= 56) ||
            (br_per_ch >= 56 && br_per_ch <= 80)) table = 0;
        else if (sfrq != 48 && br_per_ch >= 96) table = 1;
        else if (sfrq != 32 && br_per_ch <= 48) table = 2;
        else table = 3;
        if (fr_ps->tab_num != table) {
           if (fr_ps->tab_num >= 0)
              mem_free((void **)&(fr_ps->alloc));
           fr_ps->alloc = (al_table FAR *) mem_alloc(sizeof(al_table),
                                                        "alloc");
           sblim = read_bit_alloc(fr_ps->tab_num = table, fr_ps->alloc);
        }
        return sblim;
}

int js_bound(lay, m_ext)
int lay, m_ext;
{
static int jsb_table[3][4] =  { { 4, 8, 12, 16 }, { 4, 8, 12, 16},
                                { 0, 4, 8, 16} };  /* lay+m_e -> jsbound */

    if(lay<1 || lay >3 || m_ext<0 || m_ext>3) {
        fprintf(stderr, "js_bound bad layer/modext (%d/%d)\n", lay, m_ext);
        exit(1);
    }
    return(jsb_table[lay-1][m_ext]);
}

void hdr_to_frps(fr_ps) /* interpret data in hdr str to fields in fr_ps */
frame_params *fr_ps;
{
layer *hdr = fr_ps->header;     /* (or pass in as arg?) */

    fr_ps->actual_mode = hdr->mode;
    fr_ps->stereo = (hdr->mode == MPG_MD_MONO) ? 1 : 2;
    if (hdr->lay == 2)          fr_ps->sblimit = pick_table(fr_ps);
    else                        fr_ps->sblimit = SBLIMIT;
    if(hdr->mode == MPG_MD_JOINT_STEREO)
        fr_ps->jsbound = js_bound(hdr->lay, hdr->mode_ext);
    else
        fr_ps->jsbound = fr_ps->sblimit;
    /* alloc, tab_num set in pick_table */
}

void WriteHdr(fr_ps, s)
frame_params *fr_ps;
FILE *s;
{
layer *info = fr_ps->header;

   fprintf(s, "HDR:  s=FFF, id=%X, l=%X, ep=%X, br=%X, sf=%X, pd=%X, ",
           info->version, info->lay, !info->error_protection,
           info->bitrate_index, info->sampling_frequency, info->padding);
   fprintf(s, "pr=%X, m=%X, js=%X, c=%X, o=%X, e=%X\n",
           info->extension, info->mode, info->mode_ext,
           info->copyright, info->original, info->emphasis);
   fprintf(s, "layer=%s, tot bitrate=%d, sfrq=%.1f, mode=%s, ",
           layer_names[info->lay-1], bitrate[info->lay-1][info->bitrate_index],
           s_freq[info->sampling_frequency], mode_names[info->mode]);
   fprintf(s, "sblim=%d, jsbd=%d, ch=%d\n",
           fr_ps->sblimit, fr_ps->jsbound, fr_ps->stereo);
   fflush(s);
}

void WriteBitAlloc(bit_alloc, f_p, s)
unsigned int bit_alloc[2][SBLIMIT];
frame_params *f_p;
FILE *s;
```

```
{
int i,j;
int st = f_p->stereo;
int sbl = f_p->sblimit;
int jsb = f_p->jsbound;

    fprintf(s, "BITA ");
    for(i=0; i<sbl; ++i) {
        if(i == jsb) fprintf(s,"-");
        for(j=0; j<st; ++j)
            fprintf(s, "%1x", bit_alloc[j][i]);
    }
    fprintf(s, "\n");   fflush(s);
}

void WriteScale(bit_alloc, scfsi, scalar, fr_ps, s)
unsigned int bit_alloc[2][SBLIMIT], scfsi[2][SBLIMIT], scalar[2][3][SBLIMIT];
frame_params *fr_ps;
FILE *s;
{
int stereo  = fr_ps->stereo;
int sblimit = fr_ps->sblimit;
int lay     = fr_ps->header->lay;
int i,j,k;

        if(lay == 2) {
            fprintf(s, "SFSI ");
            for (i=0;i<sblimit;i++) for (k=0;k<stereo;k++)
                if (bit_alloc[k][i])  fprintf(s,"%d",scfsi[k][i]);
            fprintf(s, "\nSCFs ");
            for (k=0;k<stereo;k++) {
                for (i=0;i<sblimit;i++)
                    if (bit_alloc[k][i])
                        switch (scfsi[k][i]) {
                          case 0: for (j=0;j<3;j++)
                                  fprintf(s,"%2d%c",scalar[k][j][i],
                                          (j==2)?';':'-');
                                  break;
                          case 1:
                          case 3: fprintf(s,"%2d-",scalar[k][0][i]);
                                  fprintf(s,"%2d;",scalar[k][2][i]);
                                  break;
                          case 2: fprintf(s,"%2d;",scalar[k][0][i]);
                        }
                fprintf(s, "\n");
            }
        }
        else{   /* lay == 1 */
            fprintf(s, "SCFs ");
            for (i=0;i<sblimit;i++) for (k=0;k<stereo;k++)
                if (bit_alloc[k][i])  fprintf(s,"%2d;",scalar[k][0][i]);
            fprintf(s, "\n");
        }
}

void WriteSamples(ch, sample, bit_alloc, fr_ps, s)
int ch;
unsigned int FAR sample[SBLIMIT];
unsigned int bit_alloc[SBLIMIT];
frame_params *fr_ps;
FILE *s;
{
int i;
int stereo = fr_ps->stereo;
int sblimit = fr_ps->sblimit;

        fprintf(s, "SMPL ");
        for (i=0;i<sblimit;i++)
                if ( bit_alloc[i] != 0)
                    fprintf(s, "%d:", sample[i]);
        if(ch==(stereo-1) )     fprintf(s, "\n");
        else                    fprintf(s, "\t");
}

int NumericQ(s) /* see if a string lookd like a numeric argument */
char *s;
```

```
{
char    c;

    while( (c = *s++)!='\0' && isspace((int)c)) /* strip leading ws */
        ;
    if( c == '+' || c == '-' )
        c = *s++;               /* perhaps skip leading + or - */
    return isdigit((int)c);
}

int BitrateIndex(layr, bRate)   /* convert bitrate in kbps to index */
int     layr;           /* 1 or 2 */
int     bRate;          /* legal rates from 32 to 448 */
{
int     index = 0;
int     found = 0;

    while(!found && index<15)   {
        if(bitrate[layr-1][index] == bRate)
            found = 1;
        else
            ++index;
    }
    if(found)
        return(index);
    else {
        fprintf(stderr, "BitrateIndex: %d (layer %d) is not a legal bitrate\n",
                bRate, layr);
        return(-1);     /* Error! */
    }
}

int SmpFrqIndex(sRate)  /* convert samp frq in Hz to index */
long sRate;             /* legal rates 32000, 44100, 48000 */
{
    if(sRate == 44100L)
        return(0);
    else if(sRate == 48000L)
        return(1);
    else if(sRate == 32000L)
        return(2);
    else {
        fprintf(stderr, "SmpFrqIndex: %ld is not a legal sample rate\n", sRate);
        return(-1);      /* Error! */
    }
}

/*******************************************************************************
*
*  Allocate number of bytes of memory equal to "block".
*
*******************************************************************************/

void  FAR *mem_alloc(block, item)
unsigned long   block;
char            *item;
{

    void    *ptr;

#ifdef  MACINTOSH
    ptr = NewPtr(block);
#endif

#ifdef MSC60
    /*ptr = (void FAR *) _fmalloc((unsigned int)block);*/ /* far memory, 92-07-08 sr */
    ptr = (void FAR *) malloc((unsigned int)block); /* far memory, 93-08-24 ss */
#endif

#if ! defined (MACINTOSH) && ! defined (MSC60)
    ptr = (void FAR *) malloc(block);
#endif

    if (ptr != NULL){
#ifdef  MSC60
        _fmemset(ptr, 0, (unsigned int)block); /* far memory, 92-07-08 sr */
```

```
#else
        memset(ptr, 0, block);
#endif
    }
    else{
        printf("Unable to allocate %s\n", item);
        exit(0);
    }
    return(ptr);
}

/*****************************************************************************
*
*  Free memory pointed to by "*ptr_addr".
*
*****************************************************************************/

void    mem_free(ptr_addr)
void    **ptr_addr;
{

    if (*ptr_addr != NULL){
#ifdef  MACINTOSH
        DisposPtr(*ptr_addr);
#else
        free(*ptr_addr);
#endif
        *ptr_addr = NULL;
    }

}

/*******************************************************************************
*
*  Check block of memory all equal to a single byte, else return FALSE
*
*******************************************************************************/

int memcheck(array, test, num)
char *array;
int test;       /* but only tested as a char (bottom 8 bits) */
int num;
{
 int i=0;

   while (array[i] == test && i<num) i++;
   if (i==num) return TRUE;
   else return FALSE;
}

/*****************************************************************************
*
*  Routines to convert between the Apple SANE extended floating point format
*  and the IEEE double precision floating point format.  These routines are
*  called from within the Audio Interchange File Format (AIFF) routines.
*
*****************************************************************************/

/*
*** Apple's 80-bit SANE extended has the following format:

 1       15      1            63
+-+-------------+-+-----------------------------+
|s|       e     |i|            f                |
+-+-------------+-+-----------------------------+
  msb        lsb   msb                       lsb

The value v of the number is determined by these fields as follows:
If 0 <= e < 32767,              then v = (-1)^s * 2^(e-16383) * (i.f).
If e == 32767 and f == 0,       then v = (-1)^s * (infinity), regardless of i.
If e == 32767 and f != 0,       then v is a NaN, regardless of i.

*** IEEE Draft Standard 754 Double Precision has the following format:

MSB
```

```
+-+---------+-----------------------------+
|1| 11 Bits |           52 Bits           |
+-+---------+-----------------------------+
 ^     ^                ^
 |     |                |
 Sign  Exponent         Mantissa
*/

/****************************************************************************
*
*  double_to_extended()
*
*  Purpose:     Convert from IEEE double precision format to SANE extended
*               format.
*
*  Passed:      Pointer to the double precision number and a pointer to what
*               will hold the Apple SANE extended format value.
*
*  Outputs:     The SANE extended format pointer will be filled with the
*               converted value.
*
*  Returned:    Nothing.
*
*****************************************************************************/

void    double_to_extended(pd, ps)
double  *pd;
char    ps[10];
{

#ifdef  MACINTOSH

        x96tox80(pd, (extended *) ps);

#else

register unsigned long  top2bits;

register unsigned short *ps2;
register IEEE_DBL       *p_dbl;
register SANE_EXT       *p_ext;

   p_dbl = (IEEE_DBL *) pd;
   p_ext = (SANE_EXT *) ps;
   top2bits = p_dbl->hi & 0xc0000000;
   p_ext->l1 = ((p_dbl->hi >> 4) & 0x3ff0000) | top2bits;
   p_ext->l1 |= ((p_dbl->hi >> 5) & 0x7fff) | 0x8000;
   p_ext->l2 = (p_dbl->hi << 27) & 0xf8000000;
   p_ext->l2 |= ((p_dbl->lo >> 5) & 0x07ffffff);
   ps2 = (unsigned short *) & (p_dbl->lo);
   ps2++;
   p_ext->s1 = (*ps2 << 11) & 0xf800;

#endif

}

/****************************************************************************
*
*  extended_to_double()
*
*  Purpose:     Convert from SANE extended format to IEEE double precision
*               format.
*
*  Passed:      Pointer to the Apple SANE extended format value and a pointer
*               to what will hold the the IEEE double precision number.
*
*  Outputs:     The IEEE double precision format pointer will be filled with
*               the converted value.
*
*  Returned:    Nothing.
*
*****************************************************************************/

void    extended_to_double(ps, pd)
char    ps[10];
```

```
double  *pd;
{

#ifdef  MACINTOSH

   x80tox96((extended *) ps, pd);

#else

register unsigned long  top2bits;

register IEEE_DBL       *p_dbl;
register SANE_EXT       *p_ext;

   p_dbl = (IEEE_DBL *) pd;
   p_ext = (SANE_EXT *) ps;
   top2bits = p_ext->l1 & 0xc0000000;
   p_dbl->hi = ((p_ext->l1 << 4) & 0x3ff00000) | top2bits;
   p_dbl->hi |= (p_ext->l1 << 5) & 0xffff0;
   p_dbl->hi |= (p_ext->l2 >> 27) & 0x1f;
   p_dbl->lo = (p_ext->l2 << 5) & 0xffffffe0;
   p_dbl->lo |= (unsigned long) ((p_ext->s1 >> 11) & 0x1f);

#endif

}

/*****************************************************************************
*
*  Read Audio Interchange File Format (AIFF) headers.
*
*****************************************************************************/

int             aiff_read_headers(file_ptr, aiff_ptr)
FILE            *file_ptr;
IFF_AIFF        *aiff_ptr;
{

register char   i;
register long   seek_offset;
register long   sound_position;

char            temp_sampleRate[10];

ChunkHeader     Header;
Chunk           FormChunk;
CommonChunk     CommChunk;
SoundDataChunk  SndDChunk;

   if (fseek(file_ptr, 0, SEEK_SET) != 0)
      return(-1);

   if (fread(&FormChunk, sizeof(Chunk), 1, file_ptr) != 1)
      return(-1);

#ifdef IFF_LONG
   if (*(unsigned long *) FormChunk.ckID != IFF_ID_FORM ||
       *(unsigned long *) FormChunk.formType != IFF_ID_AIFF)
      return(-1);
#else
   if (strncmp(FormChunk.ckID,IFF_ID_FORM,4) ||
       strncmp(FormChunk.formType,IFF_ID_AIFF,4))
      return(-1);
#endif

   /*
    * chunks need not be in any particular order
    */

   while (fread(&Header, sizeof(ChunkHeader), 1, file_ptr) == 1) {

#ifdef IFF_LONG
      if (*(unsigned long *)Header.ckID == IFF_ID_COMM) {
#else
      if (strncmp(Header.ckID,IFF_ID_COMM,4) == 0) {
#endif
```

```
      /*
       * read comm chunk
       */
         if (fread(&CommChunk.numChannels, sizeof(short), 1, file_ptr) != 1)
            return(-1);

         if (fread(&CommChunk.numSampleFrames, sizeof(unsigned long), 1,
                   file_ptr) != 1)
            return(-1);

         if (fread(&CommChunk.sampleSize, sizeof(short), 1, file_ptr) != 1)
            return(-1);

         if (fread(CommChunk.sampleRate, sizeof(char[10]), 1, file_ptr) != 1)
            return(-1);

         for (i = 0; i < sizeof(char[10]); i++)
            temp_sampleRate[i] = CommChunk.sampleRate[i];

         extended_to_double(temp_sampleRate, &aiff_ptr->sampleRate);

      aiff_ptr->numChannels = CommChunk.numChannels;
      aiff_ptr->numSampleFrames = CommChunk.numSampleFrames;
      aiff_ptr->sampleSize = CommChunk.sampleSize;

#ifdef IFF_LONG
      } else if (*(unsigned long *)Header.ckID == IFF_ID_SSND) {
#else
      } else if (strncmp(Header.ckID,IFF_ID_SSND,4) == 0) {
#endif
      /*
       * read ssnd chunk
       */

      if (fread(&SndDChunk.offset, sizeof(long), 1, file_ptr) != 1)
         return(-1);

      if (fread(&SndDChunk.blockSize, sizeof(long), 1, file_ptr) != 1)
         return(-1);

      aiff_ptr->blkAlgn.offset = SndDChunk.offset;
      aiff_ptr->blkAlgn.blockSize = SndDChunk.blockSize;
      aiff_ptr->sampleType = *(unsigned long *)Header.ckID;

      /*
       * record position of sound data
       */

      sound_position = ftell(file_ptr);

      /*
       * skip over sound data to look at remaining chunks
       */

         seek_offset = Header.ckSize - sizeof(SoundDataChunk) +
            sizeof(ChunkHeader);

         if (fseek(file_ptr, seek_offset, SEEK_CUR) != 0)
            return(-1);

      } else {

      /*
       * skip unknown chunk
       */

      seek_offset = Header.ckSize;

         if (fseek(file_ptr, seek_offset, SEEK_CUR) != 0)
            return(-1);

      }

   }
```

```
   return(sound_position);

}

/*****************************************************************************
*
*  Seek past some Audio Interchange File Format (AIFF) headers to sound data.
*
*****************************************************************************/

int   aiff_seek_to_sound_data(file_ptr)
FILE  *file_ptr;
{

   if (fseek(file_ptr, sizeof(Chunk) + sizeof(SoundDataChunk), SEEK_SET) != 0)
      return(-1);

   return(0);

}

/******************************************************************************
*
*  Write Audio Interchange File Format (AIFF) headers.
*
******************************************************************************/

int             aiff_write_headers(file_ptr, aiff_ptr)
FILE            *file_ptr;
IFF_AIFF        *aiff_ptr;
{

register char   i;
register long   seek_offset;

char            temp_sampleRate[10];

Chunk           FormChunk;
CommonChunk     CommChunk;
SoundDataChunk  SndDChunk;

#ifdef IFF_LONG
   *(unsigned long *) FormChunk.ckID     = IFF_ID_FORM;
   *(unsigned long *) FormChunk.formType = IFF_ID_AIFF;
   *(unsigned long *) CommChunk.ckID     = IFF_ID_COMM;
#else
   strncpy(FormChunk.ckID,IFF_ID_FORM,4);
   strncpy(FormChunk.formType,IFF_ID_AIFF,4);
   strncpy(CommChunk.ckID,IFF_ID_COMM,4);
#endif

   double_to_extended(&aiff_ptr->sampleRate, temp_sampleRate);

   for (i = 0; i < sizeof(char[10]); i++)
      CommChunk.sampleRate[i] = temp_sampleRate[i];

   CommChunk.numChannels             = aiff_ptr->numChannels;
   CommChunk.numSampleFrames         = aiff_ptr->numSampleFrames;
   CommChunk.sampleSize              = aiff_ptr->sampleSize;
   SndDChunk.offset                  = aiff_ptr->blkAlgn.offset;
   SndDChunk.blockSize               = aiff_ptr->blkAlgn.blockSize;
   *(unsigned long *) SndDChunk.ckID = aiff_ptr->sampleType;

   CommChunk.ckSize = sizeof(CommChunk.numChannels) +
      sizeof(CommChunk.numSampleFrames) + sizeof(CommChunk.sampleSize) +
      sizeof(CommChunk.sampleRate);

   SndDChunk.ckSize = sizeof(SoundDataChunk) - sizeof(ChunkHeader) +
      (CommChunk.sampleSize + BITS_IN_A_BYTE - 1) / BITS_IN_A_BYTE *
      CommChunk.numChannels * CommChunk.numSampleFrames;

   FormChunk.ckSize = sizeof(Chunk) + SndDChunk.ckSize + sizeof(ChunkHeader) +
      CommChunk.ckSize;

   if (fseek(file_ptr, 0, SEEK_SET) != 0)
      return(-1);
```

```
   if (fwrite(&FormChunk, sizeof(Chunk), 1, file_ptr) != 1)
      return(-1);

   if (fwrite(&SndDChunk, sizeof(SoundDataChunk), 1, file_ptr) != 1)
      return(-1);

   seek_offset = SndDChunk.ckSize - sizeof(SoundDataChunk) +
      sizeof(ChunkHeader);

   if (fseek(file_ptr, seek_offset, SEEK_CUR) != 0)
      return(-1);

   if (fwrite(CommChunk.ckID, sizeof(ID), 1, file_ptr) != 1)
      return(-1);

   if (fwrite(&CommChunk.ckSize, sizeof(long), 1, file_ptr) != 1)
      return(-1);

   if (fwrite(&CommChunk.numChannels, sizeof(short), 1, file_ptr) != 1)
      return(-1);

   if (fwrite(&CommChunk.numSampleFrames, sizeof(unsigned long), 1,
              file_ptr) != 1)
      return(-1);

   if (fwrite(&CommChunk.sampleSize, sizeof(short), 1, file_ptr) != 1)
      return(-1);

   if (fwrite(CommChunk.sampleRate, sizeof(char[10]), 1, file_ptr) != 1)
      return(-1);

   return(0);

}

/*****************************************************************************
*
*  bit_stream.c package
*  Author:  Jean-Georges Fritsch, C-Cube Microsystems
*
*****************************************************************************/

/********************************************************************
  This package provides functions to write (exclusive or read)
  information from (exclusive or to) the bit stream.

  If the bit stream is opened in read mode only the get functions are
  available. If the bit stream is opened in write mode only the put
  functions are available.
********************************************************************/

/*open_bit_stream_w(); open the device to write the bit stream into it    */
/*open_bit_stream_r(); open the device to read the bit stream from it     */
/*close_bit_stream();  close the device containing the bit stream         */
/*alloc_buffer();      open and initialize the buffer;                    */
/*desalloc_buffer();   empty and close the buffer                         */
/*back_track_buffer();     goes back N bits in the buffer                 */
/*unsigned int get1bit();  read 1 bit from the bit stream                 */
/*unsigned long getbits(); read N bits from the bit stream                */
/*unsigned long byte_ali_getbits();   read the next byte aligned N bits from*/
/*                                    the bit stream                        */
/*unsigned long look_ahead(); grep the next N bits in the bit stream without*/
/*                            changing the buffer pointer                   */
/*put1bit(); write 1 bit from the bit stream  */
/*put1bit(); write 1 bit from the bit stream  */
/*putbits(); write N bits from the bit stream */
/*byte_ali_putbits(); write byte aligned the next N bits into the bit stream*/
/*unsigned long sstell(); return the current bit stream length (in bits)    */
/*int end_bs(); return 1 if the end of bit stream reached otherwise 0       */
/*int seek_sync(); return 1 if a sync word was found in the bit stream      */
/*                 otherwise returns 0                                      */

/* refill the buffer from the input device when the buffer becomes empty    */
int refill_buffer(bs)
Bit_stream_struc *bs;   /* bit stream structure */
```

```
{
   register int i=bs->buf_size-2-bs->buf_byte_idx;
   register unsigned long n=1;
   register int index=0;
   char val[2];

   while ((i>=0) && (!bs->eob)) {

      if (bs->format == BINARY)
         n = fread(&bs->buf[i--], sizeof(unsigned char), 1, bs->pt);

      else {
         while((index < 2) && n) {
            n = fread(&val[index], sizeof(char), 1, bs->pt);
            switch (val[index]) {
                  case 0x30:
                  case 0x31:
                  case 0x32:
                  case 0x33:
                  case 0x34:
                  case 0x35:
                  case 0x36:
                  case 0x37:
                  case 0x38:
                  case 0x39:
                  case 0x41:
                  case 0x42:
                  case 0x43:
                  case 0x44:
                  case 0x45:
                  case 0x46:
                  index++;
                  break;
                  default: break;
            }
         }

         if (val[0] <= 0x39)   bs->buf[i] = (val[0] - 0x30) << 4;
                 else  bs->buf[i] = (val[0] - 0x37) << 4;
         if (val[1] <= 0x39)   bs->buf[i--] |= (val[1] - 0x30);
                 else  bs->buf[i--] |= (val[1] - 0x37);
         index = 0;
      }

      if (!n) {
         bs->eob= i+1;
      }

    }
}

static char *he = "0123456789ABCDEF";

/* empty the buffer to the output device when the buffer becomes full */
void empty_buffer(bs, minimum)
Bit_stream_struc *bs;   /* bit stream structure */
int minimum;            /* end of the buffer to empty */
{
   register int i;

#if BS_FORMAT == BINARY
   for (i=bs->buf_size-1;i>=minimum;i--)
      fwrite(&bs->buf[i], sizeof(unsigned char), 1, bs->pt);
#else
   for (i=bs->buf_size-1;i>=minimum;i--) {
       char val[2];
       val[0] = he[((bs->buf[i] >> 4) & 0x0F)];
       val[1] = he[(bs->buf[i] & 0x0F)];
       fwrite(val, sizeof(char), 2, bs->pt);
   }
#endif

   for (i=minimum-1; i>=0; i--)
       bs->buf[bs->buf_size - minimum + i] = bs->buf[i];

   bs->buf_byte_idx = bs->buf_size -1 - minimum;
```

```
   bs->buf_bit_idx = 8;
}

/* open the device to write the bit stream into it */
void open_bit_stream_w(bs, bs_filenam, size)
Bit_stream_struc *bs;   /* bit stream structure */
char *bs_filenam;       /* name of the bit stream file */
int size;               /* size of the buffer */
{
   if ((bs->pt = fopen(bs_filenam, "wb")) == NULL) {
      printf("Could not create \"%s\".\n", bs_filenam);
      exit(1);
   }
   alloc_buffer(bs, size);
   bs->buf_byte_idx = size-1;
   bs->buf_bit_idx=8;
   bs->totbit=0;
   bs->mode = WRITE_MODE;
   bs->eob = FALSE;
   bs->eobs = FALSE;
}

/* open the device to read the bit stream from it */
void open_bit_stream_r(bs, bs_filenam, size)
Bit_stream_struc *bs;   /* bit stream structure */
char *bs_filenam;       /* name of the bit stream file */
int size;               /* size of the buffer */
{
   register unsigned long n;
   register unsigned char flag = 1;
   unsigned char val;

   if ((bs->pt = fopen(bs_filenam, "rb")) == NULL) {
      printf("Could not find \"%s\".\n", bs_filenam);
      exit(1);
   }

   do {
     n = fread(&val, sizeof(unsigned char), 1, bs->pt);
     switch (val) {
      case 0x30:
      case 0x31:
      case 0x32:
      case 0x33:
      case 0x34:
      case 0x35:
      case 0x36:
      case 0x37:
      case 0x38:
      case 0x39:
      case 0x41:
      case 0x42:
      case 0x43:
      case 0x44:
      case 0x45:
      case 0x46:
      case 0xa:  /* \n */
      case 0xd:  /* cr */
      case 0x1a:  /* sub */
          break;

      default: /* detection of an binary character */
          flag--;
          break;
     }

   } while (flag & n);

   if (flag) {
      printf ("the bit stream file %s is an ASCII file\n", bs_filenam);
      bs->format = ASCII;
   }
   else {
      bs->format = BINARY;
      printf ("the bit stream file %s is a BINARY file\n", bs_filenam);
   }
```

```
   fclose(bs->pt);

   if ((bs->pt = fopen(bs_filenam, "rb")) == NULL) {
      printf("Could not find \"%s\".\n", bs_filenam);
      exit(1);
   }

   alloc_buffer(bs, size);
   bs->buf_byte_idx=0;
   bs->buf_bit_idx=0;
   bs->totbit=0;
   bs->mode = READ_MODE;
   bs->eob = FALSE;
   bs->eobs = FALSE;
}

/*close the device containing the bit stream after a read process*/
void close_bit_stream_r(bs)
Bit_stream_struc *bs;   /* bit stream structure */
{
   fclose(bs->pt);
   desalloc_buffer(bs);
}

/*close the device containing the bit stream after a write process*/
void close_bit_stream_w(bs)
Bit_stream_struc *bs;   /* bit stream structure */
{
   empty_buffer(bs, bs->buf_byte_idx);
   fclose(bs->pt);
   desalloc_buffer(bs);
}

/*open and initialize the buffer; */
void alloc_buffer(bs, size)
Bit_stream_struc *bs;   /* bit stream structure */
int size;
{
   bs->buf = (unsigned char FAR *) mem_alloc(size*sizeof(unsigned
              char), "buffer");
   bs->buf_size = size;
}

/*empty and close the buffer */
void desalloc_buffer(bs)
Bit_stream_struc *bs;   /* bit stream structure */
{
   free(bs->buf);
}

int putmask[9]={0x0, 0x1, 0x3, 0x7, 0xf, 0x1f, 0x3f, 0x7f, 0xff};
int clearmask[9]={0xff, 0xfe, 0xfc, 0xf8, 0xf0, 0xe0, 0xc0, 0x80, 0x0};

void back_track_buffer(bs, N) /* goes back N bits in the buffer */
Bit_stream_struc *bs;   /* bit stream structure */
int N;
{
   int tmp = N - (N/8)*8;
   register int i;

   bs->totbit -= N;
   for (i=bs->buf_byte_idx;i< bs->buf_byte_idx+N/8-1;i++) bs->buf[i] = 0;
   bs->buf_byte_idx += N/8;
   if ( (tmp + bs->buf_bit_idx) <= 8) {
      bs->buf_bit_idx += tmp;
   }
   else {
      bs->buf_byte_idx ++;
      bs->buf_bit_idx += (tmp - 8);
   }
   bs->buf[bs->buf_byte_idx] &= clearmask[bs->buf_bit_idx];
}

int mask[8]={0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80};
```

```
/*read 1 bit from the bit stream */
unsigned int get1bit(bs)
Bit_stream_struc *bs;   /* bit stream structure */
{
   unsigned int bit;
   register int i;

   bs->totbit++;

   if (!bs->buf_bit_idx) {
        bs->buf_bit_idx = 8;
        bs->buf_byte_idx--;
        if ((bs->buf_byte_idx < MINIMUM) || (bs->buf_byte_idx < bs->eob)) {
             if (bs->eob)
                bs->eobs = TRUE;
             else {
                for (i=bs->buf_byte_idx; i>=0;i--)
                  bs->buf[bs->buf_size-1-bs->buf_byte_idx+i] = bs->buf[i];
                refill_buffer(bs);
                bs->buf_byte_idx = bs->buf_size-1;
             }
        }
   }
   bit = bs->buf[bs->buf_byte_idx]&mask[bs->buf_bit_idx-1];
   bit = bit >> (bs->buf_bit_idx-1);
   bs->buf_bit_idx--;
   return(bit);
}

/*write 1 bit from the bit stream */
void put1bit(bs, bit)
Bit_stream_struc *bs;   /* bit stream structure */
int bit;                /* bit to write into the buffer */
{
   bs->totbit++;

   bs->buf[bs->buf_byte_idx] |= (bit&0x1) << (bs->buf_bit_idx-1);
   bs->buf_bit_idx--;
   if (!bs->buf_bit_idx) {
       bs->buf_bit_idx = 8;
       bs->buf_byte_idx--;
       if (bs->buf_byte_idx < 0)
          empty_buffer(bs, MINIMUM);
       bs->buf[bs->buf_byte_idx] = 0;
   }
}

/*look ahead for the next N bits from the bit stream */
unsigned long look_ahead(bs, N)
Bit_stream_struc *bs;   /* bit stream structure */
int N;                  /* number of bits to read from the bit stream */
{
 unsigned long val=0;
 register int j = N;
 register int k, tmp;
 register int bit_idx = bs->buf_bit_idx;
 register int byte_idx = bs->buf_byte_idx;

 if (N > MAX_LENGTH)
    printf("Cannot read or write more than %d bits at a time.\n", MAX_LENGTH);

 while (j > 0) {
    if (!bit_idx) {
        bit_idx = 8;
        byte_idx--;
    }
    k = MIN (j, bit_idx);
    tmp = bs->buf[byte_idx]&putmask[bit_idx];
    tmp = tmp >> (bit_idx-k);
    val |= tmp << (j-k);
    bit_idx -= k;
    j -= k;
 }
 return(val);
}
```

```
/*read N bit from the bit stream */
unsigned long getbits(bs, N)
Bit_stream_struc *bs;   /* bit stream structure */
int N;                  /* number of bits to read from the bit stream */
{
 unsigned long val=0;
 register int i;
 register int j = N;
 register int k, tmp;

 if (N > MAX_LENGTH)
    printf("Cannot read or write more than %d bits at a time.\n", MAX_LENGTH);

 bs->totbit += N;
 while (j > 0) {
   if (!bs->buf_bit_idx) {
        bs->buf_bit_idx = 8;
        bs->buf_byte_idx--;
        if ((bs->buf_byte_idx < MINIMUM) || (bs->buf_byte_idx < bs->eob)) {
             if (bs->eob)
                bs->eobs = TRUE;
             else {
                for (i=bs->buf_byte_idx; i>=0;i--)
                   bs->buf[bs->buf_size-1-bs->buf_byte_idx+i] = bs->buf[i];
                refill_buffer(bs);
                bs->buf_byte_idx = bs->buf_size-1;
             }
        }
   }
   k = MIN (j, bs->buf_bit_idx);
   tmp = bs->buf[bs->buf_byte_idx]&putmask[bs->buf_bit_idx];
   tmp = tmp >> (bs->buf_bit_idx-k);
   val |= tmp << (j-k);
   bs->buf_bit_idx -= k;
   j -= k;
 }
 return(val);
}

/*write N bits into the bit stream */
void putbits(bs, val, N)
Bit_stream_struc *bs;   /* bit stream structure */
unsigned int val;       /* val to write into the buffer */
int N;                  /* number of bits of val */
{
 register int j = N;
 register int k, tmp;

 if (N > MAX_LENGTH)
    printf("Cannot read or write more than %d bits at a time.\n", MAX_LENGTH);

 bs->totbit += N;
 while (j > 0) {
   k = MIN(j, bs->buf_bit_idx);
   tmp = val >> (j-k);
   bs->buf[bs->buf_byte_idx] |= (tmp&putmask[k]) << (bs->buf_bit_idx-k);
   bs->buf_bit_idx -= k;
   if (!bs->buf_bit_idx) {
       bs->buf_bit_idx = 8;
       bs->buf_byte_idx--;
       if (bs->buf_byte_idx < 0)
          empty_buffer(bs, MINIMUM);
       bs->buf[bs->buf_byte_idx] = 0;
   }
   j -= k;
 }
}

/*write N bits byte aligned into the bit stream */
void byte_ali_putbits(bs, val, N)
Bit_stream_struc *bs;   /* bit stream structure */
unsigned int val;       /* val to write into the buffer */
int N;                  /* number of bits of val */
{
 unsigned long aligning, sstell();
```

```
 if (N > MAX_LENGTH)
    printf("Cannot read or write more than %d bits at a time.\n", MAX_LENGTH);
 aligning = sstell(bs)%8;
 if (aligning)
     putbits(bs, (unsigned int)0, (int)(8-aligning));

 putbits(bs, val, N);
}

/*read the next bute aligned N bits from the bit stream */
unsigned long byte_ali_getbits(bs, N)
Bit_stream_struc *bs;   /* bit stream structure */
int N;                  /* number of bits of val */
{
 unsigned long aligning, sstell();

 if (N > MAX_LENGTH)
    printf("Cannot read or write more than %d bits at a time.\n", MAX_LENGTH);
 aligning = sstell(bs)%8;
 if (aligning)
    getbits(bs, (int)(8-aligning));

 return(getbits(bs, N));
}

/*return the current bit stream length (in bits)*/
unsigned long sstell(bs)
Bit_stream_struc *bs;   /* bit stream structure */
{
  return(bs->totbit);
}

/*return the status of the bit stream*/
/* returns 1 if end of bit stream was reached */
/* returns 0 if end of bit stream was not reached */
int end_bs(bs)
Bit_stream_struc *bs;   /* bit stream structure */
{
  return(bs->eobs);
}

/*this function seeks for a byte aligned sync word in the bit stream and
  places the bit stream pointer right after the sync.
  This function returns 1 if the sync was found otherwise it returns 0  */
int seek_sync(bs, sync, N)
Bit_stream_struc *bs;   /* bit stream structure */
long sync;      /* sync word maximum 32 bits */
int N;          /* sync word length */
{
 double pow();
 unsigned long aligning, stell();
 unsigned long val;
 long maxi = (int)pow(2.0, (FLOAT)N) - 1;

 aligning = sstell(bs)%ALIGNING;
 if (aligning)
    getbits(bs, (int)(ALIGNING-aligning));

  val = getbits(bs, N);
  while (((val&maxi) != sync) && (!end_bs(bs))) {
        val <<= ALIGNING;
        val |= getbits(bs, ALIGNING);
  }

 if (end_bs(bs)) return(0);
 else return(1);
}
/*****************************************************************************
*
*  End of bit_stream.c package
*
*****************************************************************************/

/*****************************************************************************
*
*  CRC error protection package
```

```
*
***************************************************************************/

void I_CRC_calc(fr_ps, bit_alloc, crc)
frame_params *fr_ps;
unsigned int bit_alloc[2][SBLIMIT];
unsigned int *crc;
{
        int i, k;
        layer *info = fr_ps->header;
        int stereo  = fr_ps->stereo;
        int jsbound = fr_ps->jsbound;

        *crc = 0xffff; /* changed from '0' 92-08-11 shn */
        update_CRC(info->bitrate_index, 4, crc);
        update_CRC(info->sampling_frequency, 2, crc);
        update_CRC(info->padding, 1, crc);
        update_CRC(info->extension, 1, crc);
        update_CRC(info->mode, 2, crc);
        update_CRC(info->mode_ext, 2, crc);
        update_CRC(info->copyright, 1, crc);
        update_CRC(info->original, 1, crc);
        update_CRC(info->emphasis, 2, crc);

        for (i=0;i<SBLIMIT;i++)
                for (k=0;k<((i<jsbound)?stereo:1);k++)
                        update_CRC(bit_alloc[k][i], 4, crc);
}

void II_CRC_calc(fr_ps, bit_alloc, scfsi, crc)
frame_params *fr_ps;
unsigned int bit_alloc[2][SBLIMIT], scfsi[2][SBLIMIT];
unsigned int *crc;
{
        int i, k;
        layer *info = fr_ps->header;
        int stereo  = fr_ps->stereo;
        int sblimit = fr_ps->sblimit;
        int jsbound = fr_ps->jsbound;
        al_table *alloc = fr_ps->alloc;

        *crc = 0xffff; /* changed from '0' 92-08-11 shn */
        update_CRC(info->bitrate_index, 4, crc);
        update_CRC(info->sampling_frequency, 2, crc);
        update_CRC(info->padding, 1, crc);
        update_CRC(info->extension, 1, crc);
        update_CRC(info->mode, 2, crc);
        update_CRC(info->mode_ext, 2, crc);
        update_CRC(info->copyright, 1, crc);
        update_CRC(info->original, 1, crc);
        update_CRC(info->emphasis, 2, crc);

        for (i=0;i<sblimit;i++)
                for (k=0;k<((i<jsbound)?stereo:1);k++)
                        update_CRC(bit_alloc[k][i], (*alloc)[i][0].bits, crc);

        for (i=0;i<sblimit;i++)
                for (k=0;k<stereo;k++)
                        if (bit_alloc[k][i])
                                update_CRC(scfsi[k][i], 2, crc);
}

void update_CRC(data, length, crc)
unsigned int data, length, *crc;
{
        unsigned int  masking, carry;

        masking = 1 << length;

        while((masking >>= 1)){
                carry = *crc & 0x8000;
                *crc <<= 1;
                if (!carry ^ !(data & masking))
                        *crc ^= CRC16_POLYNOMIAL;
        }
        *crc &= 0xffff;
```

```
}

/*****************************************************************************
*
*  End of CRC error protection package
*
*****************************************************************************/

#ifdef  MACINTOSH
/*****************************************************************************
*
*  Set Macintosh file attributes.
*
*****************************************************************************/

void    set_mac_file_attr(fileName, vRefNum, creator, fileType)
char    fileName[MAX_NAME_SIZE];
short   vRefNum;
OsType  creator;
OsType  fileType;
{

short   theFile;
char    pascal_fileName[MAX_NAME_SIZE];
FInfo   fndrInfo;

        CtoPstr(strcpy(pascal_fileName, fileName));

        FSOpen(pascal_fileName, vRefNum, &theFile);
        GetFInfo(pascal_fileName, vRefNum, &fndrInfo);
        fndrInfo.fdCreator = creator;
        fndrInfo.fdType = fileType;
        SetFInfo(pascal_fileName, vRefNum, &fndrInfo);
        FSClose(theFile);

}
#endif


#ifdef  MS_DOS
/* ------------------------------------------------------------------------
new_ext.
Puts a new extension name on a file name <filename>.
Removes the last extension name, if any.
92-08-19 shn
------------------------------------------------------------------------ */
char *new_ext(char *filename, char *extname)
{
  int found, dotpos;
  char newname[80];

  /* First, strip the extension */
  dotpos=strlen(filename); found=0;
  do
  {
    switch (filename[dotpos])
    {
      case '.' : found=1; break;
      case '\\':                  /* used by MS-DOS */
      case '/' :                  /* used by UNIX */
      case ':' : found=-1; break; /* used by MS-DOS in drive designation */
      default  : dotpos--; if (dotpos<0) found=-1; break;
    }
  } while (found==0);
  if (found==-1) strcpy(newname,filename);
  if (found== 1) strncpy(newname,filename,dotpos); newname[dotpos]='\0';
  strcat(newname,extname);
  return(newname);
}
#endif



#define BUFSIZE 4096
static unsigned long offset,totbit=0, buf_byte_idx=0;
static unsigned int buf[BUFSIZE];
```

```
static unsigned int buf_bit_idx=8;

/*return the current bit stream length (in bits)*/
unsigned long hsstell()
{
  return(totbit);
}

/* int putmask[9]={0x0, 0x1, 0x3, 0x7, 0xf, 0x1f, 0x3f, 0x7f, 0xff}; */
extern int putmask[9];

/*read N bit from the bit stream */
unsigned long hgetbits(N)
int N;                  /* number of bits to read from the bit stream */
{
 unsigned long val=0;
 register int j = N;
 register int k, tmp;

/*
 if (N > MAX_LENGTH)
     printf("Cannot read or write more than %d bits at a time.\n", MAX_LENGTH);
*/
 totbit += N;
 while (j > 0) {
   if (!buf_bit_idx) {
        buf_bit_idx = 8;
        buf_byte_idx++;
    if (buf_byte_idx > offset)
      { printf("Buffer overflow !!\n");exit(3); }
   }
   k = MIN (j, buf_bit_idx);
   tmp = buf[buf_byte_idx%BUFSIZE]&putmask[buf_bit_idx];
   tmp = tmp >> (buf_bit_idx-k);
   val |= tmp << (j-k);
   buf_bit_idx -= k;
   j -= k;
 }
 return(val);
}

unsigned int hget1bit()
{
return(hgetbits(1));
}

/*write N bits into the bit stream */
void hputbuf(val, N)
unsigned int val;       /* val to write into the buffer */
int N;                  /* number of bits of val */
{
  if (N != 8) { printf("Not Supported yet!!\n"); exit(-3); }
  buf[offset % BUFSIZE] = val;
  offset++;
}

void rewindNbits( N )
int N;
{
   totbit -= N;
   buf_bit_idx += N;
   while( buf_bit_idx >= 8 )
   {  buf_bit_idx -= 8;
      buf_byte_idx--;
   }
}

void rewindNbytes( N )
int N;
{
   totbit -= N*8;
   buf_byte_idx -= N;
}
```

C.3.2 common.h

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
common.h
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Doulas Wong,        start of version 1.0 records          *
 *          Davis Pan                                                 *
 * 5/10/91  W. Joseph Carter    Reorganized & renamed all ".h" files  *
 *                              into "common.h" and "encoder.h".      *
 *                              Ported to Macintosh and Unix.         *
 *                              Added additional type definitions for *
 *                              AIFF, double/SANE and "bitstream.c".  *
 *                              Added function prototypes for more    *
 *                              rigorous type checking.               *
 * 27jun91  dpwe (Aware)        Added "alloc_*" defs & prototypes     *
 *                              Defined new struct 'frame_params'.    *
 *                              Changed info.stereo to info.mode_ext  *
 *                              #define constants for mode types      *
 *                              Prototype arguments if PROTO_ARGS     *
 * 5/28/91  Earle Jennings      added MS_DOS definition               *
 *                              MsDos function prototype declarations *
 * 7/10/91  Earle Jennings      added FLOAT definition as double      *
 *10/ 3/91  Don H. Lee          implemented CRC-16 error protection   *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most   *
 *                              important fixes involved changing     *
 *                              16-bit ints to long or unsigned in    *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args.     *
 *                              Removed "Other Joint Stereo" option   *
 *                              and made bitrate be total channel     *
 *                              bitrate, irrespective of the mode.    *
 *                              Fixed many small bugs & reorganized.  *
 *                              Modified some function prototypes.    *
 *                              Changed BUFFER_SIZE back to 4096.     *
 * 7/27/92  Michael Li          (re-)Ported to MS-DOS                 *
 * 7/27/92  Masahiro Iwadare    Ported to Convex                      *
 * 8/07/92  mc@tv.tek.com                                             *
 * 8/10/92  Amit Gulati         Ported to the AIX Platform (RS6000)   *
 *                              AIFF string constants redefined       *
 * 8/27/93 Seymour Shlien,      Fixes in Unix and MSDOS ports,        *
 *         Daniel Lauzon, and                                         *
 *         Bill Truerniet                                             *
 *--------------------------------------------------------------------*
 * 4/23/92  J. Pineda           Added code for Layer III.             *
 * 11/9/92  Amit Gulati         Added defines for layerIII stereo     *
 *                              modes.                                *
 *  8/24/93 Masahiro Iwadare    Included IS modification in Layer III.*
 *                              Changed for 1 pass decoding.          *
 *  9/07/93 Toshiyuki Ishino    Integrated Layer III with Ver 3.9.    *
 *--------------------------------------------------------------------*
 * 11/20/93 Masahiro Iwadare    Integrated Layer III with Ver 4.0.    *
 *--------------------------------------------------------------------*
 *  7/14/94 Juergen Koller      Fix for HPUX an IRIX in AIFF-Strings  *
 *********************************************************************/

/***********************************************************************
*
*  Global Conditional Compile Switches
*
***********************************************************************/

/* #define      UNIX            /* Unix conditional compile switch */
/* #define      MACINTOSH       /* Macintosh conditional compile switch */
/* #define      MS_DOS          /* IBM PC conditional compile switch */
/* #define      MSC60           /* Compiled for MS_DOS with MSC v6.0 */
/* #define      AIX             /* AIX conditional compile switch    */
/* #define      CONVEX          /* CONVEX conditional compile switch */
```

```
#if defined(MSC60)
#ifndef MS_DOS
#define MS_DOS
#endif
#ifndef PROTO_ARGS
#define PROTO_ARGS
#endif
#endif

#ifdef  UNIX
#define         TABLES_PATH     "tables"  /* to find data files */
/* name of environment variable holding path of table files */
#define         MPEGTABENV      "MPEGTABLES"
#define         PATH_SEPARATOR  "/"        /* how to build paths */
#endif  /* UNIX */

#ifdef  MACINTOSH
/* #define      TABLES_PATH ":tables:"  /* where to find data files */
#endif  /* MACINTOSH */

/*
 * Don't define FAR to far unless you're willing to clean up the
 * prototypes
 */
#define FAR /*far*/

#ifdef __STDC__
#ifndef PROTO_ARGS
#define PROTO_ARGS
#endif
#endif

#ifdef CONVEX
#define SEEK_SET        0
#define SEEK_CUR        1
#define SEEK_END        2
#endif

/* MS_DOS and VMS do not define TABLES_PATH, so OpenTableFile will default
   to finding the data files in the default directory */

/***********************************************************************
*
*  Global Include Files
*
***********************************************************************/

#include        <stdio.h>
#include        <string.h>
#include        <math.h>

#ifdef  UNIX
#include        <unistd.h>
#endif  /* UNIX */

#ifdef  MACINTOSH
#include        <stdlib.h>
#include        <console.h>
#endif  /* MACINTOSH */

#ifdef  MS_DOS
#include        <stdlib.h>
#ifdef MSC60
#include        <memory.h>
#else
#include        <alloc.h>
#include        <mem.h>
#endif  /* MSC60 */
#endif  /* MS_DOS */

/***********************************************************************
*
*  Global Definitions
*
***********************************************************************/
```

```
/* General Definitions */

#ifdef  MS_DOS
#define         FLOAT                   double
#else
#define         FLOAT                   float
#endif

#define         FALSE                   0
#define         TRUE                    1
#define         NULL_CHAR               '\0'

#define         MAX_U_32_NUM            0xFFFFFFFF
#define         PI                      3.14159265358979
#define         PI4                     PI/4
#define         PI64                    PI/64
#define         LN_TO_LOG10             0.2302585093

#define         VOL_REF_NUM             0
#define         MPEG_AUDIO_ID           1
#define         MAC_WINDOW_SIZE         24

#define         MONO                    1
#define         STEREO                  2
#define         BITS_IN_A_BYTE          8
#define         WORD                    16
#define         MAX_NAME_SIZE           81
#define         SBLIMIT                 32
#define         SSLIMIT                 18
#define         FFT_SIZE                1024
#define         HAN_SIZE                512
#define         SCALE_BLOCK             12
#define         SCALE_RANGE             64
#define         SCALE                   32768
#define         CRC16_POLYNOMIAL        0x8005

/* MPEG Header Definitions - Mode Values */

#define         MPG_MD_STEREO           0
#define         MPG_MD_JOINT_STEREO     1
#define         MPG_MD_DUAL_CHANNEL     2
#define         MPG_MD_MONO             3

/* Mode Extention */

#define         MPG_MD_LR_LR             0
#define         MPG_MD_LR_I              1
#define         MPG_MD_MS_LR             2
#define         MPG_MD_MS_I              3

/* AIFF Definitions */

/*
 * Note:  The value of a multi-character constant
 *        is implementation-defined.
 */
#if !defined(MS_DOS) && !defined(AIX) && !defined(__hpux) && !defined(sgi)
#define         IFF_LONG
#define         IFF_ID_FORM             'FORM'
#define         IFF_ID_AIFF             'AIFF'
#define         IFF_ID_COMM             'COMM'
#define         IFF_ID_SSND             'SSND'
#define         IFF_ID_MPEG             'MPEG'
#else
#define         IFF_ID_FORM             "FORM"
#define         IFF_ID_AIFF             "AIFF"
#define         IFF_ID_COMM             "COMM"
#define         IFF_ID_SSND             "SSND"
#define         IFF_ID_MPEG             "MPEG"
#endif

/* "bit_stream.h" Definitions */

#define         MINIMUM         4    /* Minimum size of the buffer in bytes */
#define         MAX_LENGTH      32   /* Maximum length of word written or
```

```
                                        read from bit stream */
#define         READ_MODE       0
#define         WRITE_MODE      1
#define         ALIGNING        8
#define         BINARY          0
#define         ASCII           1
#define         BS_FORMAT       ASCII /* BINARY or ASCII = 2x bytes */
#define         BUFFER_SIZE     4096

#define         MIN(A, B)       ((A) < (B) ? (A) : (B))
#define         MAX(A, B)       ((A) > (B) ? (A) : (B))

/***********************************************************************
*
*  Global Type Definitions
*
***********************************************************************/

/* Structure for Reading Layer II Allocation Tables from File */

typedef struct {
    unsigned int    steps;
    unsigned int    bits;
    unsigned int    group;
    unsigned int    quant;
} sb_alloc, *alloc_ptr;

typedef sb_alloc        al_table[SBLIMIT][16];

/* Header Information Structure */

typedef struct {
    int version;
    int lay;
    int error_protection;
    int bitrate_index;
    int sampling_frequency;
    int padding;
    int extension;
    int mode;
    int mode_ext;
    int copyright;
    int original;
    int emphasis;
} layer, *the_layer;

/* Parent Structure Interpreting some Frame Parameters in Header */

typedef struct {
    layer       *header;        /* raw header information */
    int         actual_mode;    /* when writing IS, may forget if 0 chs */
    al_table    *alloc;         /* bit allocation table read in */
    int         tab_num;        /* number of table as loaded */
    int         stereo;         /* 1 for mono, 2 for stereo */
    int         jsbound;        /* first band of joint stereo coding */
    int         sblimit;        /* total number of sub bands */
} frame_params;

/* Double and SANE Floating Point Type Definitions */

typedef struct  IEEE_DBL_struct {
    unsigned long   hi;
    unsigned long   lo;
} IEEE_DBL;

typedef struct  SANE_EXT_struct {
    unsigned long   l1;
    unsigned long   l2;
    unsigned short  s1;
} SANE_EXT;

/* AIFF Type Definitions */

typedef char    ID[4];

typedef struct  ChunkHeader_struct {
```

```
    ID      ckID;
    long    ckSize;
} ChunkHeader;

typedef struct  Chunk_struct {
    ID      ckID;
    long    ckSize;
    ID      formType;
} Chunk;

typedef struct  CommonChunk_struct {
    ID              ckID;
    long            ckSize;
    short           numChannels;
    unsigned long   numSampleFrames;
    short           sampleSize;
    char            sampleRate[10];
} CommonChunk;

typedef struct  SoundDataChunk_struct {
    ID              ckID;
    long            ckSize;
    unsigned long   offset;
    unsigned long   blockSize;
} SoundDataChunk;

typedef struct  blockAlign_struct {
    unsigned long   offset;
    unsigned long   blockSize;
} blockAlign;

typedef struct  IFF_AIFF_struct {
    short           numChannels;
    unsigned long   numSampleFrames;
    short           sampleSize;
    double          sampleRate;
    unsigned long   sampleType;
    blockAlign      blkAlgn;
} IFF_AIFF;

/* "bit_stream.h" Type Definitions */

typedef struct  bit_stream_struc {
    FILE        *pt;            /* pointer to bit stream device */
    unsigned char *buf;         /* bit stream buffer */
    int         buf_size;       /* size of buffer (in number of bytes) */
    long        totbit;         /* bit counter of bit stream */
    int         buf_byte_idx;   /* pointer to top byte in buffer */
    int         buf_bit_idx;    /* pointer to top bit of top byte in buffer */
    int         mode;           /* bit stream open in read or write mode */
    int         eob;            /* end of buffer index */
    int         eobs;           /* end of bit stream flag */
    char        format;

    /* format of file in rd mode (BINARY/ASCII) */
} Bit_stream_struc;

/* Layer III side information. */

typedef struct {
    unsigned main_data_begin;
    unsigned private_bits;
    struct {
        unsigned scfsi[4];
        struct gr_info_s {
            unsigned part2_3_length;
            unsigned big_values;
            unsigned global_gain;
            unsigned scalefac_compress;
            unsigned window_switching_flag;
            unsigned block_type;
            unsigned mixed_block_flag;
            unsigned table_select[3];
            unsigned subblock_gain[3];
            unsigned region0_count;
            unsigned region1_count;
```

```
            unsigned preflag;
            unsigned scalefac_scale;
            unsigned count1table_select;
            } gr[2];
        } ch[2];
    } III_side_info_t;

/* Layer III scale factors. */

typedef struct {
    int l[23];            /* [cb] */
    int s[3][13];         /* [window][cb] */
    } III_scalefac_t[2];  /* [ch] */

/***********************************************************************
*
*  Global Variable External Declarations
*
***********************************************************************/

extern char     *mode_names[4];
extern char     *layer_names[3];
extern double   s_freq[4];
extern int      bitrate[3][15];
extern double FAR multiple[64];

/***********************************************************************
*
*  Global Function Prototype Declarations
*
***********************************************************************/

/* The following functions are in the file "common.c" */

#ifdef  PROTO_ARGS
extern FILE           *OpenTableFile(char*);
extern int            read_bit_alloc(int, al_table*);
extern int            pick_table(frame_params*);
extern int            js_bound(int, int);
extern void           hdr_to_frps(frame_params*);
extern void           WriteHdr(frame_params*, FILE*);
extern void           WriteBitAlloc(unsigned int[2][SBLIMIT], frame_params*,
                        FILE*);
extern void           WriteScale(unsigned int[2][SBLIMIT],
                        unsigned int[2][SBLIMIT], unsigned int[2][3][SBLIMIT],
                        frame_params*, FILE*);
extern void           WriteSamples(int, unsigned int FAR [SBLIMIT],
                        unsigned int[SBLIMIT], frame_params*, FILE*);
extern int            NumericQ(char*);
extern int            BitrateIndex(int, int);
extern int            SmpFrqIndex(long);
extern int            memcheck(char*, int, int);
extern void           FAR *mem_alloc(unsigned long, char*);
extern void           mem_free(void**);
extern void           double_to_extended(double*, char[10]);
extern void           extended_to_double(char[10], double*);
extern int            aiff_read_headers(FILE*, IFF_AIFF*);
extern int            aiff_seek_to_sound_data(FILE*);
extern int            aiff_write_headers(FILE*, IFF_AIFF*);
extern int            refill_buffer(Bit_stream_struc*);
extern void           empty_buffer(Bit_stream_struc*, int);
extern void           open_bit_stream_w(Bit_stream_struc*, char*, int);
extern void           open_bit_stream_r(Bit_stream_struc*, char*, int);
extern void           close_bit_stream_r(Bit_stream_struc*);
extern void           close_bit_stream_w(Bit_stream_struc*);
extern void           alloc_buffer(Bit_stream_struc*, int);
extern void           desalloc_buffer(Bit_stream_struc*);
extern void           back_track_buffer(Bit_stream_struc*, int);
extern unsigned int   get1bit(Bit_stream_struc*);
extern void           put1bit(Bit_stream_struc*, int);
extern unsigned long  look_ahead(Bit_stream_struc*, int);
extern unsigned long  getbits(Bit_stream_struc*, int);
extern void           putbits(Bit_stream_struc*, unsigned int, int);
extern void           byte_ali_putbits(Bit_stream_struc*, unsigned int, int);
extern unsigned long  byte_ali_getbits(Bit_stream_struc*, int);
extern unsigned long  sstell(Bit_stream_struc*);
```

```
extern int            end_bs(Bit_stream_struc*);
extern int            seek_sync(Bit_stream_struc*, long, int);
extern void           I_CRC_calc(frame_params*, unsigned int[2][SBLIMIT],
                        unsigned int*);
extern void           II_CRC_calc(frame_params*, unsigned int[2][SBLIMIT],
                        unsigned int[2][SBLIMIT], unsigned int*);
extern void           update_CRC(unsigned int, unsigned int, unsigned int*);
extern void           read_absthr(FLOAT*, int);
extern unsigned int   hget1bit(); /* MI */
extern unsigned long  hgetbits(int);
extern unsigned long  hsstell();
extern void           hputbuf(unsigned int,int);



#ifdef  MACINTOSH
extern void           set_mac_file_attr(char[MAX_NAME_SIZE], short, OsType,
                        OsType);
#endif
#ifdef MS_DOS
extern char           *new_ext(char *filename, char *extname);
#endif

#else
extern FILE           *OpenTableFile();
extern int            read_bit_alloc();
extern int            pick_table();
extern int            js_bound();
extern void           hdr_to_frps();
extern void           WriteHdr();
extern void           WriteBitAlloc();
extern void           WriteScale();
extern void           WriteSamples();
extern int            NumericQ();
extern int            BitrateIndex();
extern int            SmpFrqIndex();
extern int            memcheck();
extern void           FAR *mem_alloc();
extern void           mem_free();
extern void           double_to_extended();
extern void           extended_to_double();
extern int            aiff_read_headers();
extern int            aiff_seek_to_sound_data();
extern int            aiff_write_headers();
extern int            refill_buffer();
extern void           empty_buffer();
extern void           open_bit_stream_w();
extern void           open_bit_stream_r();
extern void           close_bit_stream_r();
extern void           close_bit_stream_w();
extern void           alloc_buffer();
extern void           desalloc_buffer();
extern void           back_track_buffer();
extern unsigned int   get1bit();
extern void           put1bit();
extern unsigned long  look_ahead();
extern unsigned long  getbits();
extern void           putbits();
extern void           byte_ali_putbits();
extern unsigned long  byte_ali_getbits();
extern unsigned long  sstell();
extern int            end_bs();
extern int            seek_sync();
extern void           I_CRC_calc();
extern void           II_CRC_calc();
extern void           update_CRC();
extern void           read_absthr();

extern unsigned int   hget1bit();
extern unsigned long  hgetbits();
extern unsigned long  hsstell();
extern void           hputbuf();

#ifdef MS_DOS
extern char           *new_ext();
#endif
```

```
#endif
```

C.3.3 decode.c

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
decode.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress             *
 *   NOT for public distribution until verified and approved by the  *
 *   MPEG/audio committee.  For further information, please contact  *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com       *
 *                                                                   *
 * VERSION 4.3                                                       *
 *   changes made since last update:                                 *
 *   date   programmers         comment                              *
 * 2/25/91  Douglas Wong,       start of version 1.0 records         *
 *          Davis Pan                                                *
 * 3/06/91  Douglas Wong        rename: setup.h to dedef.h           *
 *                                      dfilter to defilter          *
 *                                      dwindow to dewindow          *
 *                              integrated "quantizer", "scalefactor" *
 *                              combined window_samples routine into  *
 *                              filter samples                       *
 * 3/31/91  Bill Aspromonte     replaced read_filter by              *
 *                              create_syn_filter and introduced a   *
 *                              new Sub-Band Synthesis routine called *
 *                              SubBandSynthesis()                   *
 * 5/10/91  Vish (PRISM)        Ported to Macintosh and Unix.        *
 *                              Changed "out_fifo()" so that last    *
 *                              unfilled block is also written out.  *
 *                              "create_syn_filter()" was modified so *
 *                              that calculation precision is same as *
 *                              in specification tables.             *
 *                              Changed "decode_scale()" to reflect  *
 *                              specifications.                      *
 *                              Removed all routines used by         *
 *                              "synchronize_buffer()".  This is now  *
 *                              replaced by "seek_sync()".           *
 *                              Incorporated Jean-Georges Fritsch's  *
 *                              "bitstream.c" package.               *
 *                              Deleted "reconstruct_sample()".      *
 * 27jun91  dpwe (Aware)        Passed outFile and &sampFrames as    *
 *                              args to out_fifo() - were global.    *
 *                              Moved "alloc_*" reader to common.c.  *
 *                              alloc, sblimit, stereo passed via new *
 *                              'frame_params struct (were globals).  *
 *                              Added JOINT STEREO decoding, lyrs I&II*
 *                              Affects: decode_bitalloc,buffer_samps *
 *                              Plus a few other cleanups.           *
 * 6/10/91   Earle Jennings     conditional expansion added in       *
 *                              II_dequantize_sample to handle range  *
 *                              problems in MSDOS version            *
 * 8/8/91    Jens Spille        Change for MS-C6.00                  *
 *10/1/91    S.I. Sudharsanan,  Ported to IBM AIX platform.          *
 *           Don H. Lee,                                             *
 *           Peter W. Farrett                                        *
 *10/3/91    Don H. Lee         implemented CRC-16 error protection  *
 *                              newly introduced functions are       *
 *                              buffer_CRC and recover_CRC_error.    *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most  *
 *                              important fixes involved changing    *
 *                              16-bit ints to long or unsigned in   *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args.    *
 *                              Removed "Other Joint Stereo" option  *
 *                              and made bitrate be total channel    *
 *                              bitrate, irrespective of the mode.   *
 *                              Fixed many small bugs & reorganized.  *
 * 7/27/92  Juan Pineda         Bug fix in SubBandSynthesis()        *
 *-------------------------------------------------------------------*
 * 6/14/92  Juan Pineda         Layer III decoding routines added.   *
 *          Amit Gulati         Follows CD 3-11172 rev2.  Contains   *
 *                              hacks deal with evolving available   *
```

```
 *                               layerIII bitstreams.  Some (minor)    *
 *                               modification of prior LI&II code.     *
 * 10/25/92 Amit Gulati          Updated layerIII routines. Added code *
 *                               for subblock_gain, switched block     *
 *                               modes, stereo pre-processing.         *
 *                               Corrected sign bits for huffman       *
 *                               decoding of quadruples region and     *
 *                               adjusted gain factor in III_dequant.  *
 * 11/21/92 Amit Gulati          Several layerIII bugs fixed.          *
 * 12/15/92 Amit Gulati          Corrected reordering (indexing)       *
 *          Stan Searing         within IMDCT routine.                 *
 *  8/24/93 Masahiro Iwadare     Included IS modification in Layer III.*
 *                               Changed for 1 pass decoding.          *
 *  9/07/93 Toshiyuki Ishino     Integrated Layer III with Ver 3.9.    *
 *--------------------------------------------------------------------*
 * 11/20/93 Masahiro Iwadare     Integrated Layer III with Ver 4.0.    *
 *--------------------------------------------------------------------*
 *  7/14/94 Juergen Koller       Bug fixes in Layer III code           *
 *--------------------------------------------------------------------*
 * 9/20/94  Davis Pan            Modification to avoid premature       *
 *                               synchword detection                   *
 *--------------------------------------------------------------------*
 * 11/09/94 Jon Rowlands         Merged premature synchword detection  *
 *                               fix into layer III code version       *
 **********************************************************************/

#include        "common.h"
#include        "decoder.h"
#include        "huffman.h"

/****************************************************************
/*
/* This module contains the core of the decoder ie all the
/* computational routines. (Layer I and II only)
/* Functions are common to both layer unless
/* otherwise specified.
/*
/****************************************************************/

/*****************************************************************
/*
/* The following routines decode the system information
/*
/****************************************************************/

/************ Layer I, Layer II & Layer III ******************/

void decode_info(bs, fr_ps)
Bit_stream_struc *bs;
frame_params *fr_ps;
{
    unsigned int bits;

    layer *hdr = fr_ps->header;
    while( (bits=getbits(bs,8)) == 255); /*discard leading 0xFF's of syncword*/
    hdr->bitrate_index = bits & 0xF;
    bits = bits >> 4;
    switch(bits) {
      case 0:
         hdr->version = 0;
         hdr->lay = 4;
         hdr->error_protection = 1;
         break;
      case 1:
         hdr->version = 0;
         hdr->lay = 4;
         hdr->error_protection = 0;
         break;
      case 2:
         hdr->version = 0;
         hdr->lay = 3;
         hdr->error_protection = 1;
         break;
      case 3:
         hdr->version = 0;
         hdr->lay = 3;
```

```
            hdr->error_protection = 0;
            break;
         case 4:
            hdr->version = 0;
            hdr->lay = 2;
            hdr->error_protection = 1;
            break;
         case 5:
            hdr->version = 0;
            hdr->lay = 2;
            hdr->error_protection = 0;
            break;
         case 6:
            hdr->version = 0;
            hdr->lay = 1;
            hdr->error_protection = 1;
            break;
         case 7:
            hdr->version = 0;
            hdr->lay = 1;
            hdr->error_protection = 0;
            break;
         case 8:
            hdr->version = 1;
            hdr->lay = 4;
            hdr->error_protection = 1;
            break;
         case 9:
            hdr->version = 1;
            hdr->lay = 4;
            hdr->error_protection = 0;
            break;
         case 10:
            hdr->version = 1;
            hdr->lay = 3;
            hdr->error_protection = 1;
            break;
         case 11:
            hdr->version = 1;
            hdr->lay = 3;
            hdr->error_protection = 0;
            break;
         case 12:
            hdr->version = 1;
            hdr->lay = 2;
            hdr->error_protection = 1;
            break;
         case 13:
            hdr->version = 1;
            hdr->lay = 2;
            hdr->error_protection = 0;
            break;
         case 14:
            hdr->version = 1;
            hdr->lay = 1;
            hdr->error_protection = 1;
            break;
         default:
            hdr->version = 1;
            hdr->lay = 1;
            hdr->error_protection = 0;
     }
     hdr->sampling_frequency = getbits(bs,2);
     hdr->padding = get1bit(bs);
     hdr->extension = get1bit(bs);
     hdr->mode = getbits(bs,2);
     hdr->mode_ext = getbits(bs,2);
     hdr->copyright = get1bit(bs);
     hdr->original = get1bit(bs);
     hdr->emphasis = getbits(bs,2);
}

/*******************************************************************
/*
/* The bit allocation information is decoded. Layer I
/* has 4 bit per subband whereas Layer II is Ws and bit rate
```

```
/* dependent.
/*
/*********************************************************************/

/**************************** Layer II *************/

void II_decode_bitalloc(bs, bit_alloc, fr_ps)
Bit_stream_struc *bs;
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
{
    int i,j;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    al_table *alloc = fr_ps->alloc;

    for (i=0;i<jsbound;i++) for (j=0;j<stereo;j++)
        bit_alloc[j][i] = (char) getbits(bs,(*alloc)[i][0].bits);

    for (i=jsbound;i<sblimit;i++) /* expand to 2 channels */
        bit_alloc[0][i] = bit_alloc[1][i] =
            (char) getbits(bs,(*alloc)[i][0].bits);

    for (i=sblimit;i<SBLIMIT;i++) for (j=0;j<stereo;j++)
        bit_alloc[j][i] = 0;
}

/**************************** Layer I *************/

void I_decode_bitalloc(bs, bit_alloc, fr_ps)
Bit_stream_struc *bs;
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
{
    int i,j;
    int stereo  = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    int b;

    for (i=0;i<jsbound;i++) for (j=0;j<stereo;j++)
        bit_alloc[j][i] = getbits(bs,4);
    for (i=jsbound;i<SBLIMIT;i++) {
        b = getbits(bs,4);
        for (j=0;j<stereo;j++)
            bit_alloc[j][i] = b;
    }
}

/*****************************************************************
/*
/* The following two functions implement the layer I and II
/* format of scale factor extraction. Layer I involves reading
/* 6 bit per subband as scale factor. Layer II requires reading
/* first the scfsi which in turn indicate the number of scale factors
/* transmitted.
/*    Layer I : I_decode_scale
/*   Layer II : II_decode_scale
/*
/****************************************************************/

/************************** Layer I stuff ************************/

void I_decode_scale(bs, bit_alloc, scale_index, fr_ps)
Bit_stream_struc *bs;
unsigned int bit_alloc[2][SBLIMIT], scale_index[2][3][SBLIMIT];
frame_params *fr_ps;
{
    int i,j;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;

    for (i=0;i<SBLIMIT;i++) for (j=0;j<stereo;j++)
        if (!bit_alloc[j][i])
            scale_index[j][0][i] = SCALE_RANGE-1;
```

```
        else                    /* 6 bit per scale factor */
            scale_index[j][0][i] =  getbits(bs,6);

}

/************************** Layer II stuff **************************/

void II_decode_scale(bs,scfsi, bit_alloc,scale_index, fr_ps)
Bit_stream_struc *bs;
unsigned int scfsi[2][SBLIMIT], bit_alloc[2][SBLIMIT],
             scale_index[2][3][SBLIMIT];
frame_params *fr_ps;
{
    int i,j;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;

    for (i=0;i<sblimit;i++) for (j=0;j<stereo;j++) /* 2 bit scfsi */
        if (bit_alloc[j][i]) scfsi[j][i] = (char) getbits(bs,2);
    for (i=sblimit;i<SBLIMIT;i++) for (j=0;j<stereo;j++)
        scfsi[j][i] = 0;

    for (i=0;i<sblimit;i++) for (j=0;j<stereo;j++) {
        if (bit_alloc[j][i])
            switch (scfsi[j][i]) {
                /* all three scale factors transmitted */
             case 0 : scale_index[j][0][i] = getbits(bs,6);
                scale_index[j][1][i] = getbits(bs,6);
                scale_index[j][2][i] = getbits(bs,6);
                break;
                /* scale factor 1 & 3 transmitted */
             case 1 : scale_index[j][0][i] =
                 scale_index[j][1][i] = getbits(bs,6);
                scale_index[j][2][i] = getbits(bs,6);
                break;
                /* scale factor 1 & 2 transmitted */
             case 3 : scale_index[j][0][i] = getbits(bs,6);
                scale_index[j][1][i] =
                    scale_index[j][2][i] =  getbits(bs,6);
                break;
                /* only one scale factor transmitted */
             case 2 : scale_index[j][0][i] =
                 scale_index[j][1][i] =
                     scale_index[j][2][i] = getbits(bs,6);
                break;
                default : break;
            }
        else {
            scale_index[j][0][i] = scale_index[j][1][i] =
                scale_index[j][2][i] = SCALE_RANGE-1;
        }
    }
    for (i=sblimit;i<SBLIMIT;i++) for (j=0;j<stereo;j++) {
        scale_index[j][0][i] = scale_index[j][1][i] =
            scale_index[j][2][i] = SCALE_RANGE-1;
    }
}

/***************************************************************
/*
/*   The following two routines take care of reading the
/* compressed sample from the bit stream for both layer 1 and
/* layer 2. For layer 1, read the number of bits as indicated
/* by the bit_alloc information. For layer 2, if grouping is
/* indicated for a particular subband, then the sample size has
/* to be read from the bits_group and the merged samples has
/* to be decompose into the three distinct samples. Otherwise,
/* it is the same for as layer one.
/*
/***************************************************************/

/****************************** Layer I stuff ******************/

void I_buffer_sample(bs, sample, bit_alloc, fr_ps)
unsigned int FAR sample[2][3][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
```

```
Bit_stream_struc *bs;
frame_params *fr_ps;
{
    int i,j,k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    unsigned int s;

    for (i=0;i<jsbound;i++) for (j=0;j<stereo;j++)
        if ( (k = bit_alloc[j][i]) == 0)
            sample[j][0][i] = 0;
        else
            sample[j][0][i] = (unsigned int) getbits(bs,k+1);
    for (i=jsbound;i<SBLIMIT;i++) {
        if ( (k = bit_alloc[0][i]) == 0)
            s = 0;
        else
            s = (unsigned int)getbits(bs,k+1);
        for (j=0;j<stereo;j++)
            sample[j][0][i]    = s;
    }
}

/*************************** Layer II stuff ************************/

void II_buffer_sample(bs,sample,bit_alloc,fr_ps)
unsigned int FAR sample[2][3][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
Bit_stream_struc *bs;
frame_params *fr_ps;
{
    int i,j,k,m;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    al_table *alloc = fr_ps->alloc;

    for (i=0;i<sblimit;i++) for (j=0;j<((i<jsbound)?stereo:1);j++) {
        if (bit_alloc[j][i]) {
            /* check for grouping in subband */
            if ((*alloc)[i][bit_alloc[j][i]].group==3)
                for (m=0;m<3;m++) {
                    k = (*alloc)[i][bit_alloc[j][i]].bits;
                    sample[j][m][i] = (unsigned int) getbits(bs,k);
                }
            else {              /* bit_alloc = 3, 5, 9 */
                unsigned int nlevels, c=0;

                nlevels = (*alloc)[i][bit_alloc[j][i]].steps;
                k=(*alloc)[i][bit_alloc[j][i]].bits;
                c = (unsigned int) getbits(bs, k);
                for (k=0;k<3;k++) {
                    sample[j][k][i] = c % nlevels;
                    c /= nlevels;
                }
            }
        }
        else {                  /* for no sample transmitted */
            for (k=0;k<3;k++) sample[j][k][i] = 0;
        }
        if(stereo == 2 && i>= jsbound) /* joint stereo : copy L to R */
            for (k=0;k<3;k++) sample[1][k][i] = sample[0][k][i];
    }
    for (i=sblimit;i<SBLIMIT;i++) for (j=0;j<stereo;j++) for (k=0;k<3;k++)
        sample[j][k][i] = 0;
}

/**************************************************************
/*
/*   Restore the compressed sample to a factional number.
/*   first complement the MSB of the sample
/*    for layer I :
/*    Use s = (s' + 2^(-nb+1) ) * 2^nb / (2^nb-1)
/*   for Layer II :
/*   Use the formula s = s' * c + d
```

```
/*
/*****************************************************************/

static double c[17] = { 1.33333333333, 1.60000000000, 1.14285714286,
                        1.77777777777, 1.06666666666, 1.03225806452,
                        1.01587301587, 1.00787401575, 1.00392156863,
                        1.00195694716, 1.00097751711, 1.00048851979,
                        1.00024420024, 1.00012208522, 1.00006103888,
                        1.00003051851, 1.00001525902 };

static double d[17] = { 0.500000000, 0.500000000, 0.250000000, 0.500000000,
                        0.125000000, 0.062500000, 0.031250000, 0.015625000,
                        0.007812500, 0.003906250, 0.001953125, 0.0009765625,
                        0.00048828125, 0.00024414063, 0.00012207031,
                        0.00006103516, 0.00003051758 };

/************************** Layer II stuff ************************/

void II_dequantize_sample(sample, bit_alloc, fraction, fr_ps)
unsigned int FAR sample[2][3][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
double FAR fraction[2][3][SBLIMIT];
frame_params *fr_ps;
{
    int i, j, k, x;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    al_table *alloc = fr_ps->alloc;

    for (i=0;i<sblimit;i++)  for (j=0;j<3;j++) for (k=0;k<stereo;k++)
        if (bit_alloc[k][i]) {

            /* locate MSB in the sample */
            x = 0;
#ifndef MS_DOS
            while ((1L<<x) < (*alloc)[i][bit_alloc[k][i]].steps) x++;
#else
            /* microsoft C thinks an int is a short */
            while (( (unsigned long) (1L<<(long)x) <
                    (unsigned long)( (*alloc)[i][bit_alloc[k][i]].steps)
                    ) && ( x < 16) ) x++;
#endif

            /* MSB inversion */
            if (((sample[k][j][i] >> x-1) & 1) == 1)
                fraction[k][j][i] = 0.0;
            else  fraction[k][j][i] = -1.0;

            /* Form a 2's complement sample */
            fraction[k][j][i] += (double) (sample[k][j][i] & ((1<<x-1)-1)) /
                (double) (1L<<x-1);

            /* Dequantize the sample */
            fraction[k][j][i] += d[(*alloc)[i][bit_alloc[k][i]].quant];
            fraction[k][j][i] *= c[(*alloc)[i][bit_alloc[k][i]].quant];
        }
        else fraction[k][j][i] = 0.0;

    for (i=sblimit;i<SBLIMIT;i++) for (j=0;j<3;j++) for(k=0;k<stereo;k++)
        fraction[k][j][i] = 0.0;
}

/***************************** Layer I stuff **********************/

void I_dequantize_sample(sample, fraction, bit_alloc, fr_ps)
unsigned int FAR sample[2][3][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
double FAR fraction[2][3][SBLIMIT];
frame_params *fr_ps;
{
    int i, nb, k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;

    for (i=0;i<SBLIMIT;i++)
        for (k=0;k<stereo;k++)
```

```
            if (bit_alloc[k][i]) {
                nb = bit_alloc[k][i] + 1;
                if (((sample[k][0][i] >> nb-1) & 1) == 1) fraction[k][0][i] = 0.0;
                else fraction[k][0][i] = -1.0;
                fraction[k][0][i] += (double) (sample[k][0][i] & ((1<<nb-1)-1)) /
                    (double) (1L<<nb-1);

                fraction[k][0][i] =
                    (double) (fraction[k][0][i]+1.0/(double)(1L<<nb-1)) *
                        (double) (1L<<nb) / (double) ((1L<<nb)-1);
            }
            else fraction[k][0][i] = 0.0;
}

/************************************************************
/*
/*   Restore the original value of the sample ie multiply
/*    the fraction value by its scalefactor.
/*
/************************************************************/

/************************* Layer II Stuff **********************/

void II_denormalize_sample(fraction, scale_index,fr_ps,x)
double FAR fraction[2][3][SBLIMIT];
unsigned int scale_index[2][3][SBLIMIT];
frame_params *fr_ps;
int x;
{
    int i,j,k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;

    for (i=0;i<sblimit;i++) for (j=0;j<stereo;j++) {
        fraction[j][0][i] *= multiple[scale_index[j][x][i]];
        fraction[j][1][i] *= multiple[scale_index[j][x][i]];
        fraction[j][2][i] *= multiple[scale_index[j][x][i]];
    }
}

/**************************** Layer I stuff *****************************/

void I_denormalize_sample(fraction,scale_index,fr_ps)
double FAR fraction[2][3][SBLIMIT];
unsigned int scale_index[2][3][SBLIMIT];
frame_params *fr_ps;
{
    int i,j,k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;

    for (i=0;i<SBLIMIT;i++) for (j=0;j<stereo;j++)
        fraction[j][0][i] *= multiple[scale_index[j][0][i]];
}

/*****************************************************************
/*
/* The following are the subband synthesis routines. They apply
/* to both layer I and layer II stereo or mono. The user has to
/* decide what parameters are to be passed to the routines.
/*
/***************************************************************/

/*************************************************************
/*
/*   Pass the subband sample through the synthesis window
/*
/**************************************************************/

/* create in synthesis filter */

void create_syn_filter(filter)
double FAR filter[64][SBLIMIT];
{
    register int i,k;
```

```
    for (i=0; i<64; i++)
        for (k=0; k<32; k++) {
            if ((filter[i][k] = 1e9*cos((double)((PI64*i+PI4)*(2*k+1)))) >= 0)
                modf(filter[i][k]+0.5, &filter[i][k]);
            else
                modf(filter[i][k]-0.5, &filter[i][k]);
            filter[i][k] *= 1e-9;
        }
}

/****************************************************************
/*
/*   Window the restored sample
/*
/****************************************************************/

/* read in synthesis window */

void read_syn_window(window)
double FAR window[HAN_SIZE];
{
    int i,j[4];
    FILE *fp;
    double f[4];
    char t[150];

    if (!(fp = OpenTableFile("dewindow") )) {
        printf("Please check synthesis window table 'dewindow'\n");
        exit(1);
    }
    for (i=0;i<512;i+=4) {
        fgets(t, 150, fp);
        sscanf(t,"D[%d] = %lf D[%d] = %lf D[%d] = %lf D[%d] = %lf\n",
               j, f,j+1,f+1,j+2,f+2,j+3,f+3);
        if (i==j[0]) {
            window[i] = f[0];
            window[i+1] = f[1];
            window[i+2] = f[2];
            window[i+3] = f[3];
        }
        else {
            printf("Check index in synthesis window table\n");
            exit(1);
        }
        fgets(t,150,fp);
    }
    fclose(fp);
}

int SubBandSynthesis (bandPtr, channel, samples)
double *bandPtr;
int channel;
short *samples;
{
    register int i,j,k;
    register double *bufOffsetPtr, sum;
    static int init = 1;
    typedef double NN[64][32];
    static NN FAR *filter;
    typedef double BB[2][2*HAN_SIZE];
    static BB FAR *buf;
    static int bufOffset[2] = {64,64};
    static double FAR *window;
    int clip = 0;               /* count & return how many samples clipped */

    if (init) {
        buf = (BB FAR *) mem_alloc(sizeof(BB),"BB");
        filter = (NN FAR *) mem_alloc(sizeof(NN), "NN");
        create_syn_filter(*filter);
        window = (double FAR *) mem_alloc(sizeof(double) * HAN_SIZE, "WIN");
        read_syn_window(window);
        init = 0;
    }
/*    if (channel == 0) */
    bufOffset[channel] = (bufOffset[channel] - 64) & 0x3ff;
    bufOffsetPtr = &((*buf)[channel][bufOffset[channel]]);
```

```
    for (i=0; i<64; i++) {
        sum = 0;
        for (k=0; k<32; k++)
            sum += bandPtr[k] * (*filter)[i][k];
        bufOffsetPtr[i] = sum;
    }
    /*  S(i,j) = D(j+32i) * U(j+32i+((i+1)>>1)*64)  */
    /*  samples(i,j) = MWindow(j+32i) * bufPtr(j+32i+((i+1)>>1)*64)  */
    for (j=0; j<32; j++) {
        sum = 0;
        for (i=0; i<16; i++) {
            k = j + (i<<5);
            sum += window[k] * (*buf) [channel] [( (k + ( ((i+1)>>1) <<6) ) +
                                          bufOffset[channel]) & 0x3ff];
        }

/*   {long foo = (sum > 0) ? sum * SCALE + 0.5 : sum * SCALE - 0.5; */
     {long foo = sum * SCALE;
     if (foo >= (long) SCALE)      {samples[j] = SCALE-1; ++clip;}
     else if (foo < (long) -SCALE) {samples[j] = -SCALE;  ++clip;}
     else                           samples[j] = foo;
 }
    }
    return(clip);
}

void out_fifo(pcm_sample, num, fr_ps, done, outFile, psampFrames)
short FAR pcm_sample[2][SSLIMIT][SBLIMIT];
int num;
frame_params *fr_ps;
int done;
FILE *outFile;
unsigned long *psampFrames;
{
    int i,j,l;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    static short int outsamp[1600];
    static long k = 0;

    if (!done)
        for (i=0;i<num;i++) for (j=0;j<SBLIMIT;j++) {
            (*psampFrames)++;
            for (l=0;l<stereo;l++) {
                if (!(k%1600) && k) {
                    fwrite(outsamp,2,1600,outFile);
                    k = 0;
                }
                outsamp[k++] = pcm_sample[l][i][j];
            }
        }
    else {
        fwrite(outsamp,2,(int)k,outFile);
        k = 0;
    }
}

void  buffer_CRC(bs, old_crc)
Bit_stream_struc  *bs;
unsigned int  *old_crc;
{
    *old_crc = getbits(bs, 16);
}

void  recover_CRC_error(pcm_sample, error_count, fr_ps, outFile, psampFrames)
short FAR pcm_sample[2][SSLIMIT][SBLIMIT];
int error_count;
frame_params *fr_ps;
FILE *outFile;
unsigned long *psampFrames;
{
    int  stereo = fr_ps->stereo;
    int  num, done, i;
    int  samplesPerFrame, samplesPerSlot;
    layer  *hdr = fr_ps->header;
```

```
    long  offset;
    short  *temp;

    num = 3;
    if (hdr->lay == 1) num = 1;

    samplesPerSlot = SBLIMIT * num * stereo;
    samplesPerFrame = samplesPerSlot * 32;

    if (error_count == 1) {     /* replicate previous error_free frame */
        done = 1;
        /* flush out fifo */
        out_fifo(pcm_sample, num, fr_ps, done, outFile, psampFrames);
        /* go back to the beginning of the previous frame */
        offset = sizeof(short int) * samplesPerFrame;
        fseek(outFile, -offset, SEEK_CUR);
        done = 0;
        for (i = 0; i < SCALE_BLOCK; i++) {
            fread(pcm_sample, 2, samplesPerSlot, outFile);
            out_fifo(pcm_sample, num, fr_ps, done, outFile, psampFrames);
        }
    }
    else{                       /* mute the frame */
        temp = (short*) pcm_sample;
        done = 0;
        for (i = 0; i < 2*3*SBLIMIT; i++)
            *temp++ = MUTE;     /* MUTE value is in decoder.h */
        for (i = 0; i < SCALE_BLOCK; i++)
            out_fifo(pcm_sample, num, fr_ps, done, outFile, psampFrames);
    }
}

/************************* Layer III routines **********************/

void III_get_side_info(bs, si, fr_ps)
Bit_stream_struc *bs;
III_side_info_t *si;
frame_params *fr_ps;
{
   int ch, gr, i;
   int stereo = fr_ps->stereo;

   si->main_data_begin = getbits(bs, 9);
   if (stereo == 1)
      si->private_bits = getbits(bs,5);
      else si->private_bits = getbits(bs,3);

   for (ch=0; ch<stereo; ch++)
      for (i=0; i<4; i++)
      si->ch[ch].scfsi[i] = get1bit(bs);

   for (gr=0; gr<2; gr++) {
      for (ch=0; ch<stereo; ch++) {
         si->ch[ch].gr[gr].part2_3_length = getbits(bs, 12);
         si->ch[ch].gr[gr].big_values = getbits(bs, 9);
         si->ch[ch].gr[gr].global_gain = getbits(bs, 8);
         si->ch[ch].gr[gr].scalefac_compress = getbits(bs, 4);
         si->ch[ch].gr[gr].window_switching_flag = get1bit(bs);
         if (si->ch[ch].gr[gr].window_switching_flag) {
            si->ch[ch].gr[gr].block_type = getbits(bs, 2);
            si->ch[ch].gr[gr].mixed_block_flag = get1bit(bs);
            for (i=0; i<2; i++)
               si->ch[ch].gr[gr].table_select[i] = getbits(bs, 5);
            for (i=0; i<3; i++)
               si->ch[ch].gr[gr].subblock_gain[i] = getbits(bs, 3);

            /* Set region_count parameters since they are implicit in this case. */

            if (si->ch[ch].gr[gr].block_type == 0) {
               printf("Side info bad: block_type == 0 in split block.\n");
               exit(0);
               }
            else if (si->ch[ch].gr[gr].block_type == 2
                     && si->ch[ch].gr[gr].mixed_block_flag == 0)
               si->ch[ch].gr[gr].region0_count = 8; /* MI 9; */
            else si->ch[ch].gr[gr].region0_count = 7; /* MI 8; */
```

```
            si->ch[ch].gr[gr].region1_count = 20 -
                       si->ch[ch].gr[gr].region0_count;
            }
         else {
            for (i=0; i<3; i++)
               si->ch[ch].gr[gr].table_select[i] = getbits(bs, 5);
            si->ch[ch].gr[gr].region0_count = getbits(bs, 4);
            si->ch[ch].gr[gr].region1_count = getbits(bs, 3);
            si->ch[ch].gr[gr].block_type = 0;
            }
         si->ch[ch].gr[gr].preflag = get1bit(bs);
         si->ch[ch].gr[gr].scalefac_scale = get1bit(bs);
         si->ch[ch].gr[gr].count1table_select = get1bit(bs);
         }
      }
}

void III_put_side_info(bs, si, fr_ps)
frame_params *fr_ps;
Bit_stream_struc *bs;
III_side_info_t *si;
{
   int ch, gr, i;
   int stereo = fr_ps->stereo;

   putbits(bs, si->main_data_begin,9);
   if (stereo == 1)
      putbits(bs, si->private_bits, 5);
      else putbits(bs, si->private_bits, 3);

   for (ch=0; ch<stereo; ch++)
      for (i=0; i<4; i++)
         put1bit(bs, si->ch[ch].scfsi[i]);

   for (gr=0; gr<2; gr++) {
      for (ch=0; ch<stereo; ch++) {
         putbits(bs, si->ch[ch].gr[gr].part2_3_length, 12);
         putbits(bs, si->ch[ch].gr[gr].big_values, 9);
         putbits(bs, si->ch[ch].gr[gr].global_gain, 8);
         putbits(bs, si->ch[ch].gr[gr].scalefac_compress, 4);
         put1bit(bs, si->ch[ch].gr[gr].window_switching_flag);
         if (si->ch[ch].gr[gr].window_switching_flag) {
            putbits(bs, si->ch[ch].gr[gr].block_type, 2);
            put1bit(bs, si->ch[ch].gr[gr].mixed_block_flag);
            for (i=0; i<2; i++)
               putbits(bs, si->ch[ch].gr[gr].table_select[i], 5);
            for (i=0; i<3; i++)
               putbits(bs, si->ch[ch].gr[gr].subblock_gain[i], 3);
            }
         else {
            for (i=0; i<3; i++)
            putbits(bs, si->ch[ch].gr[gr].table_select[i], 5);
            putbits(bs, si->ch[ch].gr[gr].region0_count, 4);
            putbits(bs, si->ch[ch].gr[gr].region1_count, 3);
            }

         put1bit(bs, si->ch[ch].gr[gr].preflag);
         put1bit(bs, si->ch[ch].gr[gr].scalefac_scale);
         put1bit(bs, si->ch[ch].gr[gr].count1table_select);
         }
      }
}

struct {
   int l[5];
   int s[3];} sfbtable = {{0, 6, 11, 16, 21},
                          {0, 6, 12}};

int slen[2][16] = {{0, 0, 0, 0, 3, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4},
                   {0, 1, 2, 3, 0, 1, 2, 3, 1, 2, 3, 1, 2, 3, 2, 3}};
struct  {
   int l[23];
   int s[14];} sfBandIndex[3] =
   {{{0,4,8,12,16,20,24,30,36,44,52,62,74,90,110,134,162,196,238,288,342,418,576},
     {0,4,8,12,16,22,30,40,52,66,84,106,136,192}},
    {{0,4,8,12,16,20,24,30,36,42,50,60,72,88,106,128,156,190,230,276,330,384,576},
```

```
     {0,4,8,12,16,22,28,38,50,64,80,100,126,192}},
    {{0,4,8,12,16,20,24,30,36,44,54,66,82,102,126,156,194,240,296,364,448,550,576},
     {0,4,8,12,16,22,30,42,58,78,104,138,180,192}}};


void III_get_scale_factors(scalefac, si, gr, ch, fr_ps)
III_scalefac_t *scalefac;
III_side_info_t *si;
int gr, ch;
frame_params *fr_ps;
{
int sfb, i, window;
struct gr_info_s *gr_info = &(si->ch[ch].gr[gr]);

    if (gr_info->window_switching_flag && (gr_info->block_type == 2)) {
      if (gr_info->mixed_block_flag) { /* MIXED */ /* NEW - ag 11/25 */
         for (sfb = 0; sfb < 8; sfb++)
            (*scalefac)[ch].l[sfb] = hgetbits(
                 slen[0][gr_info->scalefac_compress]);
         for (sfb = 3; sfb < 6; sfb++)
            for (window=0; window<3; window++)
               (*scalefac)[ch].s[window][sfb] = hgetbits(
                 slen[0][gr_info->scalefac_compress]);
         for (sfb = 6; sfb < 12; sfb++)
            for (window=0; window<3; window++)
               (*scalefac)[ch].s[window][sfb] = hgetbits(
                 slen[1][gr_info->scalefac_compress]);
         for (sfb=12,window=0; window<3; window++)
            (*scalefac)[ch].s[window][sfb] = 0;
      }
      else {  /* SHORT*/
         for (i=0; i<2; i++)
            for (sfb = sfbtable.s[i]; sfb < sfbtable.s[i+1]; sfb++)
               for (window=0; window<3; window++)
                  (*scalefac)[ch].s[window][sfb] = hgetbits(
                    slen[i][gr_info->scalefac_compress]);
         for (sfb=12,window=0; window<3; window++)
            (*scalefac)[ch].s[window][sfb] = 0;
      }
    }
    else {   /* LONG types 0,1,3 */
        for (i=0; i<4; i++) {
           if ((si->ch[ch].scfsi[i] == 0) || (gr == 0))
              for (sfb = sfbtable.l[i]; sfb < sfbtable.l[i+1]; sfb++)
                  (*scalefac)[ch].l[sfb] = hgetbits(
                 slen[(i<2)?0:1][gr_info->scalefac_compress]);
        }
        (*scalefac)[ch].l[22] = 0;
    }
}


/* Already declared in huffman.c
struct huffcodetab ht[HTN];
*/
int huffman_initialized = FALSE;

void initialize_huffman() {
   FILE *fi;

   if (huffman_initialized) return;
   if (!(fi = OpenTableFile("huffdec") )) {
      printf("Please check huffman table 'huffdec'\n");
      exit(1);
   }

   if (fi==NULL) {

      fprintf(stderr,"decoder table open error\n");

      exit(3);

      }

   if (read_decoder_table(fi) != HTN) {
      fprintf(stderr,"decoder table read error\n");
```

```
      exit(4);
      }
huffman_initialized = TRUE;
}

III_hufman_decode(is, si, ch, gr, part2_start, fr_ps)
long int is[SBLIMIT][SSLIMIT];
III_side_info_t *si;
int gr, ch, part2_start;
frame_params *fr_ps;
{
   int i, x, y;
   int v, w;
   struct huffcodetab *h;
   int region1Start;
   int region2Start;
   int bt = (*si).ch[ch].gr[gr].window_switching_flag && ((*si).ch[ch].gr[gr].block_type == 2);

   initialize_huffman();

   /* Find region boundary for short block case. */

   if ( ((*si).ch[ch].gr[gr].window_switching_flag) &&
        ((*si).ch[ch].gr[gr].block_type == 2) ) {

      /* Region2. */

      region1Start = 36;  /* sfb[9/3]*3=36 */
      region2Start = 576; /* No Region2 for short block case. */
   }


   else {          /* Find region boundary for long block case. */

      region1Start = sfBandIndex[fr_ps->header->sampling_frequency]
                           .l[(*si).ch[ch].gr[gr].region0_count + 1]; /* MI */
      region2Start = sfBandIndex[fr_ps->header->sampling_frequency]
                              .l[(*si).ch[ch].gr[gr].region0_count +
                              (*si).ch[ch].gr[gr].region1_count + 2]; /* MI */
      }


   /* Read bigvalues area. */
   for (i=0; i<(*si).ch[ch].gr[gr].big_values*2; i+=2) {
      if      (i<region1Start) h = &ht[(*si).ch[ch].gr[gr].table_select[0]];
      else if (i<region2Start) h = &ht[(*si).ch[ch].gr[gr].table_select[1]];
           else                h = &ht[(*si).ch[ch].gr[gr].table_select[2]];
      huffman_decoder(h, &x, &y, &v, &w);
      is[i/SSLIMIT][i%SSLIMIT] = x;
      is[(i+1)/SSLIMIT][(i+1)%SSLIMIT] = y;
      }

   /* Read count1 area. */
   h = &ht[(*si).ch[ch].gr[gr].count1table_select+32];
   while ((hsstell() < part2_start + (*si).ch[ch].gr[gr].part2_3_length ) &&
     ( i < SSLIMIT*SBLIMIT )) {
      huffman_decoder(h, &x, &y, &v, &w);
      is[i/SSLIMIT][i%SSLIMIT] = v;
      is[(i+1)/SSLIMIT][(i+1)%SSLIMIT] = w;
      is[(i+2)/SSLIMIT][(i+2)%SSLIMIT] = x;
      is[(i+3)/SSLIMIT][(i+3)%SSLIMIT] = y;
      i += 4;
      }

   if (hsstell() > part2_start + (*si).ch[ch].gr[gr].part2_3_length)
   {  i -=4;
      rewindNbits(hsstell()-part2_start - (*si).ch[ch].gr[gr].part2_3_length);
   }

   /* Dismiss stuffing Bits */
   if ( hsstell() < part2_start + (*si).ch[ch].gr[gr].part2_3_length )
      hgetbits( part2_start + (*si).ch[ch].gr[gr].part2_3_length - hsstell());

   /* Zero out rest. */
   for (; i<SSLIMIT*SBLIMIT; i++)
      is[i/SSLIMIT][i%SSLIMIT] = 0;
```

```
}

int pretab[21] = {0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,2};

void III_dequantize_sample(is,xr,scalefac,gr_info, ch,fr_ps)
long int is[SBLIMIT][SSLIMIT];
double xr[SBLIMIT][SSLIMIT];
struct gr_info_s *gr_info;
III_scalefac_t *scalefac;
frame_params *fr_ps;
int ch;
{
   int ss,sb,cb=0,sfreq=fr_ps->header->sampling_frequency;
   int stereo = fr_ps->stereo;
   int next_cb_boundary, cb_begin, cb_width, sign;

   /* choose correct scalefactor band per block type, initalize boundary */

   if (gr_info->window_switching_flag && (gr_info->block_type == 2) )
      if (gr_info->mixed_block_flag)
         next_cb_boundary=sfBandIndex[sfreq].l[1];  /* LONG blocks: 0,1,3 */
      else {
         next_cb_boundary=sfBandIndex[sfreq].s[1]*3; /* pure SHORT block */
    cb_width = sfBandIndex[sfreq].s[1];
    cb_begin = 0;
      }
   else
      next_cb_boundary=sfBandIndex[sfreq].l[1];  /* LONG blocks: 0,1,3 */

   /* apply formula per block type */

   for (sb=0 ; sb < SBLIMIT ; sb++)
      for (ss=0 ; ss < SSLIMIT ; ss++) {

         if ( (sb*18)+ss == next_cb_boundary)  { /* Adjust critical band boundary */
            if (gr_info->window_switching_flag && (gr_info->block_type == 2)) {
               if (gr_info->mixed_block_flag)  {
                  if (((sb*18)+ss) == sfBandIndex[sfreq].l[8])  {
                     next_cb_boundary=sfBandIndex[sfreq].s[4]*3;
                     cb = 3;
                     cb_width = sfBandIndex[sfreq].s[cb+1] -
                                sfBandIndex[sfreq].s[cb];
                     cb_begin = sfBandIndex[sfreq].s[cb]*3;
                  }
                  else if (((sb*18)+ss) < sfBandIndex[sfreq].l[8])
                     next_cb_boundary = sfBandIndex[sfreq].l[(++cb)+1];
                  else {
                     next_cb_boundary = sfBandIndex[sfreq].s[(++cb)+1]*3;
                     cb_width = sfBandIndex[sfreq].s[cb+1] -
                                    sfBandIndex[sfreq].s[cb];
                     cb_begin = sfBandIndex[sfreq].s[cb]*3;
                  }
               }
               else  {
                  next_cb_boundary = sfBandIndex[sfreq].s[(++cb)+1]*3;
                  cb_width = sfBandIndex[sfreq].s[cb+1] -
                               sfBandIndex[sfreq].s[cb];
                cb_begin = sfBandIndex[sfreq].s[cb]*3;
               }
            }
            else /* long blocks */
               next_cb_boundary = sfBandIndex[sfreq].l[(++cb)+1];
         }

         /* Compute overall (global) scaling. */

         xr[sb][ss] = pow( 2.0 , (0.25 * (gr_info->global_gain - 210.0)));

         /* Do long/short dependent scaling operations. */

         if (gr_info->window_switching_flag && (
            ((gr_info->block_type == 2) && (gr_info->mixed_block_flag == 0)) ||
            ((gr_info->block_type == 2) && gr_info->mixed_block_flag && (sb >= 2)) )) {

            xr[sb][ss] *= pow(2.0, 0.25 * -8.0 *
```

```
                    gr_info->subblock_gain[(((sb*18)+ss) - cb_begin)/cb_width]);
            xr[sb][ss] *= pow(2.0, 0.25 * -2.0 * (1.0+gr_info->scalefac_scale)
              * (*scalefac)[ch].s[(((sb*18)+ss) - cb_begin)/cb_width][cb]);
         }
         else {   /* LONG block types 0,1,3 & 1st 2 subbands of switched blocks */
            xr[sb][ss] *= pow(2.0, 0.25 * -2.0 * (1.0+gr_info->scalefac_scale)
                                        * gr_info->preflag * pretab[cb]);
            xr[sb][ss] *= pow(2.0, 0.25 * -2.0 * (1.0+gr_info->scalefac_scale)
                                        * (*scalefac)[ch].l[cb]);
         }

         /* Scale quantized value. */

         sign = (is[sb][ss]<0) ? 1 : 0;
         xr[sb][ss] *= pow( (double) abs(is[sb][ss]), ((double)4.0/3.0) );
         if (sign) xr[sb][ss] = -xr[sb][ss];
      }
}

III_reorder (xr, ro, gr_info, fr_ps)
double xr[SBLIMIT][SSLIMIT];
double ro[SBLIMIT][SSLIMIT];
struct gr_info_s *gr_info;
frame_params *fr_ps;
{
   int sfreq=fr_ps->header->sampling_frequency;
   int sfb, sfb_start, sfb_lines;
   int sb, ss, window, freq, src_line, des_line;

   for(sb=0;sb<SBLIMIT;sb++)
      for(ss=0;ss<SSLIMIT;ss++)
         ro[sb][ss] = 0;

   if (gr_info->window_switching_flag && (gr_info->block_type == 2)) {
      if (gr_info->mixed_block_flag) {
         /* NO REORDER FOR LOW 2 SUBBANDS */
         for (sb=0 ; sb < 2 ; sb++)
            for (ss=0 ; ss < SSLIMIT ; ss++) {
               ro[sb][ss] = xr[sb][ss];
            }
         /* REORDERING FOR REST SWITCHED SHORT */
         for(sfb=3,sfb_start=sfBandIndex[sfreq].s[3],
            sfb_lines=sfBandIndex[sfreq].s[4] - sfb_start;
            sfb < 13; sfb++,sfb_start=sfBandIndex[sfreq].s[sfb],
            (sfb_lines=sfBandIndex[sfreq].s[sfb+1] - sfb_start))
               for(window=0; window<3; window++)
                  for(freq=0;freq<sfb_lines;freq++) {
                     src_line = sfb_start*3 + window*sfb_lines + freq;
                     des_line = (sfb_start*3) + window + (freq*3);
                     ro[des_line/SSLIMIT][des_line%SSLIMIT] =
                                    xr[src_line/SSLIMIT][src_line%SSLIMIT];
               }
      }
      else {  /* pure short */
         for(sfb=0,sfb_start=0,sfb_lines=sfBandIndex[sfreq].s[1];
            sfb < 13; sfb++,sfb_start=sfBandIndex[sfreq].s[sfb],
            (sfb_lines=sfBandIndex[sfreq].s[sfb+1] - sfb_start))
               for(window=0; window<3; window++)
                  for(freq=0;freq<sfb_lines;freq++) {
                     src_line = sfb_start*3 + window*sfb_lines + freq;
                     des_line = (sfb_start*3) + window + (freq*3);
                     ro[des_line/SSLIMIT][des_line%SSLIMIT] =
                                    xr[src_line/SSLIMIT][src_line%SSLIMIT];
               }
      }
   }
   else {   /*long blocks */
      for (sb=0 ; sb < SBLIMIT ; sb++)
         for (ss=0 ; ss < SSLIMIT ; ss++)
            ro[sb][ss] = xr[sb][ss];
   }
}


void III_stereo(xr, lr, scalefac, gr_info, fr_ps)
```

```
double xr[2][SBLIMIT][SSLIMIT];
double lr[2][SBLIMIT][SSLIMIT];
III_scalefac_t *scalefac;
struct gr_info_s *gr_info;
frame_params *fr_ps;
{
   int sfreq = fr_ps->header->sampling_frequency;
   int stereo = fr_ps->stereo;
   int ms_stereo = (fr_ps->header->mode = MPG_MD_JOINT_STEREO) &&
                   (fr_ps->header->mode_ext & 0x2);
   int i_stereo = (fr_ps->header->mode = MPG_MD_JOINT_STEREO) &&
                  (fr_ps->header->mode_ext & 0x1);
   int js_bound;  /* frequency line that marks the beggining of the zero part */
   int sfb,next_sfb_boundary;
   int i,j,sb,ss,ch,is_pos[576];
   double is_ratio[576];

   /* intialization */
   for ( i=0; i<576; i++ )
      is_pos[i] = 7;

   if ((stereo == 2) && i_stereo && !ms_stereo )
   {  if (gr_info->window_switching_flag && (gr_info->block_type == 2))
      {  if( gr_info->mixed_block_flag )
         {  int max_sfb = 0;

            for ( j=0; j<3; j++ )
            {  i = 2 + (180*(j+1)) / 18;
               ss = 180*(j+1) - 18 * i;
               sb = 0;
               while ( i >= (2 + (180*j)))
               {  if ( xr[1][i][ss] != 0.0 )
                  {  sb = i*18+ss;
                     i = -1;
                  } else
                  {  ss--;
                     if ( ss < 0 )
                     {  i--;
                        ss = 17;
                     }
                  }
               }
               if ( i > 0  )
                  i = 3;
               else
               {  sb = sb - 180*j - 36 + 12;
                  i = 0;
                  while ( sfBandIndex[sfreq].s[i] <= sb )
                     i++;
               }
               sfb = i;
               if ( sfb > max_sfb )
                  max_sfb = sfb;
               i = 36 + 180*j + sfBandIndex[sfreq].s[i] - 12;
               while( sfb<12 )
               {  sb = sfBandIndex[sfreq].s[sfb+1] - sfBandIndex[sfreq].s[sfb];
                  for ( ; sb > 0; sb--)
                  {  is_pos[i] = (*scalefac)[1].s[j][sfb];
                     if ( is_pos[i] != 7 )
                        is_ratio[i] = tan( is_pos[i] * (PI / 12));
                     i++;
                  }
                  sfb++;
               }
               sfb = sfBandIndex[sfreq].s[11];
               for ( sb = 192 - sfBandIndex[sfreq].s[12]; sb > 0; sb-- )
               {  is_pos[i] = is_pos[sfb];
                  is_ratio[i] = is_ratio[sfb];
                  i++;
               }
               if ( max_sfb <= 3 )
               {  i = 2;
                  ss = 17;
                  while ( i >= 0 )
                  {  if ( xr[1][i][ss] != 0.0 )
                     {  sb = i*18+ss;
```

```
                  i = -1;
               } else
               {  ss--;
                  if ( ss < 0 )
                  {  i--;
                     ss = 17;
                  }
               }
            }
            i = 0;
            while ( sfBandIndex[sfreq].l[i] <= sb )
               i++;
            sfb = i;
            i = sfBandIndex[sfreq].l[i];
            for ( ; sfb<8; sfb++ )
            {  sb = sfBandIndex[sfreq].l[sfb+1]-sfBandIndex[sfreq].l[sfb];
               for ( ; sb > 0; sb--)
               {  is_pos[i] = (*scalefac)[1].l[sfb];
                  if ( is_pos[i] != 7 )
                     is_ratio[i] = tan( is_pos[i] * (PI / 12));
                  i++;
               }
            }
         }
      }
   } else
   {  for ( j=0; j<3; j++ )
      {  i = 192*(j+1) / 18;
         ss = 192*(j+1) - 18 * i;
         sb = -1;
         while ( i >= (192 * j))
         {  if ( xr[1][i][ss] != 0.0 )
            {  sb = i*18+ss;
               i = -1;
            } else
            {  ss--;
               if ( ss < 0 )
               {  i--;
                  ss = 17;
               }
            }
         }
         if ( i>0 )
            i = 0;
         else
         {  sb = sb - 192*j;
            i = 0;
            while ( sfBandIndex[sfreq].s[i] <= sb )
               i++;
         }
         sfb = i;
         i = 192*j + sfBandIndex[sfreq].s[i];
         while( sfb<12 )
         {  sb = sfBandIndex[sfreq].s[sfb+1] - sfBandIndex[sfreq].s[sfb];
            for ( ; sb > 0; sb--)
            {  is_pos[i] = (*scalefac)[1].s[j][sfb];
               if ( is_pos[i] != 7 )
                  is_ratio[i] = tan( is_pos[i] * (PI / 12));
               i++;
            }
            sfb++;
         }
         sfb = sfBandIndex[sfreq].s[11];
         for ( sb = 192 - sfBandIndex[sfreq].s[12]; sb > 0; sb-- )
         {  is_pos[i] = is_pos[sfb];
            is_ratio[i] = is_ratio[sfb];
            i++;
         }
      }
   }
} else
{  i = 31;
   ss = 17;
   sb = 0;
   while ( i >= 0 )
   {  if ( xr[1][i][ss] != 0.0 )
```

```
            {  sb = i*18+ss;
               i = -1;
            } else
            {  ss--;
               if ( ss < 0 )
               {  i--;
                  ss = 17;
               }
            }
         }
         i = 0;
         while ( sfBandIndex[sfreq].l[i] <= sb )
            i++;
         sfb = i;
         i = sfBandIndex[sfreq].l[i];
         for ( ; sfb<21; sfb++ )
         {  sb = sfBandIndex[sfreq].l[sfb+1] - sfBandIndex[sfreq].l[sfb];
            for ( ; sb > 0; sb--)
            {  is_pos[i] = (*scalefac)[1].l[sfb];
               if ( is_pos[i] != 7 )
                  is_ratio[i] = tan( is_pos[i] * (PI / 12));
               i++;
            }
         }
         sfb = sfBandIndex[sfreq].l[20];
         for ( sb = 576 - sfBandIndex[sfreq].l[21]; sb > 0; sb-- )
         {  is_pos[i] = is_pos[sfb];
            is_ratio[i] = is_ratio[sfb];
            i++;
         }
      }
   }

   for(ch=0;ch<2;ch++)
      for(sb=0;sb<SBLIMIT;sb++)
         for(ss=0;ss<SSLIMIT;ss++)
            lr[ch][sb][ss] = 0;

   if (stereo==2)
      for(sb=0;sb<SBLIMIT;sb++)
         for(ss=0;ss<SSLIMIT;ss++) {
            i = (sb*18)+ss;
            if ( is_pos[i] == 7 ) {
               if ( ms_stereo ) {
                  lr[0][sb][ss] = (xr[0][sb][ss]+xr[1][sb][ss])/1.41421356;
                  lr[1][sb][ss] = (xr[0][sb][ss]-xr[1][sb][ss])/1.41421356;
               }
               else {
                  lr[0][sb][ss] = xr[0][sb][ss];
                  lr[1][sb][ss] = xr[1][sb][ss];
               }
            }
            else if (i_stereo ) {
               lr[0][sb][ss] = xr[0][sb][ss] * (is_ratio[i]/(1+is_ratio[i]));
               lr[1][sb][ss] = xr[0][sb][ss] * (1/(1+is_ratio[i]));
            }
            else {
               printf("Error in streo processing\n");
            }
         }
   else  /* mono , bypass xr[0][][] to lr[0][][]*/
      for(sb=0;sb<SBLIMIT;sb++)
         for(ss=0;ss<SSLIMIT;ss++)
            lr[0][sb][ss] = xr[0][sb][ss];

}

double Ci[8]={-0.6,-0.535,-0.33,-0.185,-0.095,-0.041,-0.0142,-0.0037};

void III_antialias(xr, hybridIn, gr_info, fr_ps)
double xr[SBLIMIT][SSLIMIT];
double hybridIn[SBLIMIT][SSLIMIT];
struct gr_info_s *gr_info;
frame_params *fr_ps;
{
   static int    init = 1;
```

```
   static double ca[8],cs[8];
   double        bu,bd;  /* upper and lower butterfly inputs */
   int           ss,sb,sblim;

   if (init) {
      int i;
      double    sq;
      for (i=0;i<8;i++) {
         sq=sqrt(1.0+Ci[i]*Ci[i]);
         cs[i] = 1.0/sq;
         ca[i] = Ci[i]/sq;
      }
      init = 0;
   }

   /* clear all inputs */

    for(sb=0;sb<SBLIMIT;sb++)
       for(ss=0;ss<SSLIMIT;ss++)
          hybridIn[sb][ss] = xr[sb][ss];

   if  (gr_info->window_switching_flag && (gr_info->block_type == 2) &&
       !gr_info->mixed_block_flag ) return;

   if ( gr_info->mixed_block_flag )
      sblim = 1;
   else
      sblim = SBLIMIT-1;

   /* 31 alias-reduction operations between each pair of sub-bands */
   /* with 8 butterflies between each pair                         */

   for(sb=0;sb<sblim;sb++)
      for(ss=0;ss<8;ss++) {
         bu = xr[sb][17-ss];
         bd = xr[sb+1][ss];
         hybridIn[sb][17-ss] = (bu * cs[ss]) - (bd * ca[ss]);
         hybridIn[sb+1][ss] = (bd * cs[ss]) + (bu * ca[ss]);
         }
}

void inv_mdct(in, out, block_type)
double in[18];
double out[36];
int block_type;
{
/*------------------------------------------------------------------*/
/*                                                                  */
/*    Function: Calculation of the inverse MDCT                     */
/*    In the case of short blocks the 3 output vectors are already  */
/*    overlapped and added in this modul.                           */
/*                                                                  */
/*    New layer3                                                    */
/*                                                                  */
/*------------------------------------------------------------------*/

int     k,i,m,N,p;
double  tmp[12],sum;
static  double  win[4][36];
static  int init=0;
static  double COS[4*36];

    if(init==0){

    /* type 0 */
      for(i=0;i<36;i++)
         win[0][i] = sin( PI/36 *(i+0.5) );

    /* type 1*/
      for(i=0;i<18;i++)
         win[1][i] = sin( PI/36 *(i+0.5) );
      for(i=18;i<24;i++)
         win[1][i] = 1.0;
      for(i=24;i<30;i++)
         win[1][i] = sin( PI/12 *(i+0.5-18) );
      for(i=30;i<36;i++)
```

```
         win[1][i] = 0.0;

    /* type 3*/
      for(i=0;i<6;i++)
         win[3][i] = 0.0;
      for(i=6;i<12;i++)
         win[3][i] = sin( PI/12 *(i+0.5-6) );
      for(i=12;i<18;i++)
         win[3][i] =1.0;
      for(i=18;i<36;i++)
         win[3][i] = sin( PI/36*(i+0.5) );

    /* type 2*/
      for(i=0;i<12;i++)
         win[2][i] = sin( PI/12*(i+0.5) ) ;
      for(i=12;i<36;i++)
         win[2][i] = 0.0 ;

      for (i=0; i<4*36; i++)
         COS[i] = cos(PI/(2*36) * i);

      init++;
    }

    for(i=0;i<36;i++)
       out[i]=0;

    if(block_type == 2){
       N=12;
       for(i=0;i<3;i++){
          for(p= 0;p<N;p++){
             sum = 0.0;
             for(m=0;m<N/2;m++)
                sum += in[i+3*m] * cos( PI/(2*N)*(2*p+1+N/2)*(2*m+1) );
             tmp[p] = sum * win[block_type][p] ;
          }
          for(p=0;p<N;p++)
             out[6*i+p+6] += tmp[p];
       }
    }
    else{
      N=36;
      for(p= 0;p<N;p++){
         sum = 0.0;
         for(m=0;m<N/2;m++)
           sum += in[m] * COS[((2*p+1+N/2)*(2*m+1))%(4*36)];
         out[p] = sum * win[block_type][p];
      }
    }
}

void III_hybrid(fsIn, tsOut ,sb, ch, gr_info, fr_ps)
double fsIn[SSLIMIT];   /* freq samples per subband in */
double tsOut[SSLIMIT];  /* time samples per subband out */
int sb, ch;
struct gr_info_s *gr_info;
frame_params *fr_ps;
{
   int ss;
   double rawout[36];
   static double prevblck[2][SBLIMIT][SSLIMIT];
   static int init = 1;
   int bt;

   if (init) {
      int i,j,k;

      for(i=0;i<2;i++)
         for(j=0;j<SBLIMIT;j++)
            for(k=0;k<SSLIMIT;k++)
               prevblck[i][j][k]=0.0;
      init = 0;
   }

   bt = (gr_info->window_switching_flag && (gr_info->block_type == 2) &&
         gr_info->mixed_block_flag && (sb < 2)) ? 0 : gr_info->block_type;
```

```
   inv_mdct( fsIn, rawout, bt);

   /* overlap addition */
   for(ss=0; ss<SSLIMIT; ss++) {
      tsOut[ss] = rawout[ss] + prevblck[ch][sb][ss];
      prevblck[ch][sb][ss] = rawout[ss+18];
   }
}

/* Return the number of slots for main data of current frame, */

int main_data_slots(fr_ps)
frame_params fr_ps;
{int nSlots;

   nSlots = (144 * bitrate[2][fr_ps.header->bitrate_index])
        / s_freq[fr_ps.header->sampling_frequency];
  if (fr_ps.header->padding) nSlots++;
  nSlots -= 4;
  if (fr_ps.header->error_protection) nSlots -= 2;
  if (fr_ps.stereo == 1) nSlots -= 17; else nSlots -=32;
  return(nSlots);
}
```

C.3.4 decoder.h

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
decoder.h
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Doulas Wong,        start of version 1.0 records          *
 *          Davis Pan                                                 *
 * 5/10/91  Vish (PRISM)        Renamed and regrouped all ".h" files  *
 *                              into "common.h" and "decoder.h".      *
 *                              Ported to Macintosh and Unix.         *
 * 27jun91  dpwe (Aware)        New prototype for out_fifo()          *
 *                              Moved "alloc_*" stuff to common.h     *
 *                              Use ifdef PROTO_ARGS for prototypes   *
 *                              prototypes reflect frame_params struct*
 * 10/3/91  Don H. Lee          implemented CRC-16 error protection   *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most   *
 *                              important fixes involved changing     *
 *                              16-bit ints to long or unsigned in    *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args.     *
 *                              Removed "Other Joint Stereo" option   *
 *                              and made bitrate be total channel     *
 *                              bitrate, irrespective of the mode.    *
 *                              Fixed many small bugs & reorganized.  *
 *                              Modified some function prototypes.    *
 * 08/07/92 Mike Coleman        Made small changes for portability    *
 * 9/07/93  Toshiyuki Ishino    Integrated with Layer III.            *
 * 11/04/94 Jon Rowlands        fix protos for usage() and            *
 *                              recover_CRC_error()                   *
 **********************************************************************/

/***********************************************************************
*
*  Decoder Include Files
*
***********************************************************************/

/***********************************************************************
*
*  Decoder Definitions
```

```
*
**********************************************************************/

#define   DFLT_OPEXT        ".dec"  /* default output file name extension */
/*
 NOTE: The value of a multiple-character constant is
 implementation-defined.
*/
#if !defined(MS_DOS) && !defined(AIX)
#define   FILTYP_DEC_AIFF   'AIFF'
#define   FILTYP_DEC_BNRY   'TEXT'
#define   CREATR_DEC_AIFF   'Sd2a'
/*
  The following character constant is ASCII '????'
  It is declared in hex because the character
  constant contains a trigraph, causing an error in
  parsing with ANSI preprocessors.
*/
#define   CREATR_DEC_BNRY   0x3f3f3f3f
#else
#define   FILTYP_DEC_AIFF   "AIFF"
#define   FILTYP_DEC_BNRY   "TEXT"
#define   CREATR_DEC_AIFF   "Sd2a"
#define   CREATR_DEC_BNRY   "????"
#endif

#define   SYNC_WORD         (long) 0xfff
#define   SYNC_WORD_LNGTH   12

#define   MUTE              0

/**********************************************************************
*
*  Decoder Type Definitions
*
**********************************************************************/

/**********************************************************************
*
*  Decoder Variable External Declarations
*
**********************************************************************/

/**********************************************************************
*
*  Decoder Function Prototype Declarations
*
**********************************************************************/

/* The following functions are in the file "musicout.c" */

#ifdef   PROTO_ARGS
static void   usage(void);
#else
static void   usage();
#endif

/* The following functions are in the file "decode.c" */

#ifdef   PROTO_ARGS
extern void   decode_info(Bit_stream_struc*, frame_params*);
extern void   II_decode_bitalloc(Bit_stream_struc*, unsigned int[2][SBLIMIT],
                       frame_params*);
extern void   I_decode_bitalloc(Bit_stream_struc*, unsigned int[2][SBLIMIT],
                       frame_params*);
extern void   I_decode_scale(Bit_stream_struc*, unsigned int[2][SBLIMIT],
                       unsigned int[2][3][SBLIMIT], frame_params*);
extern void   II_decode_scale(Bit_stream_struc*, unsigned int[2][SBLIMIT],
                       unsigned int[2][SBLIMIT], unsigned int[2][3][SBLIMIT],
                       frame_params*);
extern void   I_buffer_sample(Bit_stream_struc*, unsigned int[2][3][SBLIMIT],
                       unsigned int[2][SBLIMIT], frame_params*);
extern void   II_buffer_sample(Bit_stream_struc*, unsigned int[2][3][SBLIMIT],
                       unsigned int[2][SBLIMIT], frame_params*);
extern void   read_quantizer_table(double[17], double[17]);
extern void   II_dequantize_sample(unsigned int[2][3][SBLIMIT],
```

```
                       unsigned int[2][SBLIMIT], double[2][3][SBLIMIT],
                       frame_params*);
extern void   I_dequantize_sample(unsigned int[2][3][SBLIMIT],
                       double[2][3][SBLIMIT], unsigned int[2][SBLIMIT],
                       frame_params*);
extern void   read_scale_factor(double[SCALE_RANGE]);
extern void   II_denormalize_sample(double[2][3][SBLIMIT],
                       unsigned int[2][3][SBLIMIT], frame_params*, int);
extern void   I_denormalize_sample(double[2][3][SBLIMIT],
                       unsigned int[2][3][SBLIMIT], frame_params*);
extern void   create_syn_filter(double[64][SBLIMIT]);
extern int    SubBandSynthesis (double*, int, short*);
extern void   read_syn_window(double[HAN_SIZE]);
extern void   window_sample(double*, double*);
extern void   out_fifo(short[2][SSLIMIT][SBLIMIT], int, frame_params*, int,
                       FILE*, unsigned long*);
extern void   buffer_CRC(Bit_stream_struc*, unsigned int*);
extern void   recover_CRC_error(short[2][3][SBLIMIT], int, frame_params*,
                       FILE*, unsigned long*);
extern void   III_dequantize_sample(long int[SBLIMIT][SSLIMIT],
             double [SBLIMIT][SSLIMIT], III_scalefac_t *,
                        struct gr_info_s *, int, frame_params *);
extern void   III_antialias(double[SBLIMIT][SSLIMIT], double[SBLIMIT][SSLIMIT],
                          struct gr_info_s *, frame_params *);
extern void   inv_mdct(double[18], double[36], int);
extern void   III_hybrid(double[SSLIMIT], double[SSLIMIT] , int, int,
                       struct gr_info_s *, frame_params *);

#else
extern void   decode_info();
extern void   II_decode_bitalloc();
extern void   I_decode_bitalloc();
extern void   I_decode_scale();
extern void   II_decode_scale();
extern void   I_buffer_sample();
extern void   II_buffer_sample();
extern void   read_quantizer_table();
extern void   II_dequantize_sample();
extern void   I_dequantize_sample();
extern void   read_scale_factor();
extern void   II_denormalize_sample();
extern void   I_denormalize_sample();
extern void   create_syn_filter();
extern int    SubBandSynthesis ();
extern void   read_syn_window();
extern void   window_sample();
extern void   out_fifo();
extern void   buffer_CRC();
extern void   recover_CRC_error();
extern void   III_dequantize_sample();
extern void   III_antialias();
extern void   inv_mdct();
extern void   III_hybrid();
#endif
```

C.3.5 encode.c

```
/**********************************************************************
Copyright (c) 1991 ISO/IEC JTC1 SC29 WG1, All Rights Reserved
encode.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress             *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 3/01/91  Douglas Wong,       start of version 1.1 records          *
 *          Davis Pan                                                 *
 * 3/06/91  Douglas Wong        rename: setup.h to endef.h            *
 *                                      efilter to enfilter           *
 *                                      ewindow to enwindow           *
```

```
 *                              integrated "quantizer", "scalefactor",*
 *                              and "transmission" files             *
 *                              update routine "window_subband"      *
 * 3/31/91  Bill Aspromonte     replaced read_filter by              *
 *                              create_an_filter                     *
 * 5/10/91  W. Joseph Carter    Ported to Macintosh and Unix.        *
 *                              Incorporated Jean-Georges Fritsch's  *
 *                              "bitstream.c" package.               *
 *                              Incorporated Bill Aspromonte's       *
 *                              filterbank coefficient matrix        *
 *                              calculation routines and added       *
 *                              roundoff to coincide with specs.     *
 *                              Modified to strictly adhere to       *
 *                              encoded bitstream specs, including   *
 *                              "Berlin changes".                    *
 *                              Modified PCM sound file handling to  *
 *                              process all incoming samples and fill *
 *                              out last encoded frame with zeros    *
 *                              (silence) if needed.                 *
 *                              Located and fixed numerous software  *
 *                              bugs and table data errors.          *
 * 19jun91  dpwe (Aware)        moved "alloc_*" reader to common.c   *
 *                              Globals sblimit, alloc replaced by new*
 *                              struct 'frame_params' passed as arg. *
 *                              Added JOINT STEREO coding, layers I,II*
 *                              Affects: *_bit_allocation,           *
 *                              subband_quantization, encode_bit_alloc*
 *                              sample_encoding                      *
 * 6/10/91  Earle Jennings      modified II_subband_quantization to  *
 *                              resolve type cast problem for MS_DOS *
 * 6/11/91  Earle Jennings      modified to avoid overflow on MS_DOS *
 *                              in routine filter_subband            *
 * 7/10/91  Earle Jennings      port to MsDos from MacIntosh version *
 * 8/ 8/91  Jens Spille         Change for MS-C6.00                  *
 *10/ 1/91  S.I. Sudharsanan,   Ported to IBM AIX platform.          *
 *          Don H. Lee,                                              *
 *          Peter W. Farrett                                         *
 *10/ 3/91  Don H. Lee          implemented CRC-16 error protection  *
 *                              newly introduced function encode_CRC *
 *11/ 8/91  Kathy Wang          Documentation of code                *
 *                              All variablenames are referred to    *
 *                              with surrounding pound (#) signs     *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most  *
 *                              important fixes involved changing    *
 *                              16-bit ints to long or unsigned in   *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args.    *
 *                              Removed "Other Joint Stereo" option  *
 *                              and made bitrate be total channel    *
 *                              bitrate, irrespective of the mode.   *
 *                              Fixed many small bugs & reorganized. *
 * 6/16/92  Shaun Astarabadi    Changed I_scale_factor_calc() and    *
 *                              II_scale_factor_calc() to use scale  *
 *                              factor 0 thru 62 only and not to     *
 *                              encode index 63 into the bit stream. *
 * 7/27/92  Mike Li             (re-)Port to MS-DOS                  *
 * 9/22/92  jddevine@aware.com  Fixed _scale_factor_calc() defs      *
 * 3/31/93  Giogio Dimino       changed II_a_bit_allocation() from:  *
 *                              if( ad > ...) to if(ad >= ...)       *
 * 8/05/93  TEST                changed I_a_bit_allocation() from:   *
 *                              if( ad > ...) to if(ad >= ...)       *
 **********************************************************************/

#include "common.h"
#include "encoder.h"

#ifdef MS_DOS
extern unsigned _stklen = 16384;
#endif


/*=======================================================================\
|                                                                       |
| This segment contains all the core routines of the encoder,          |
| except for the psychoacoustic models.                                 |
|                                                                       |
```

```
| The user can select either one of the two psychoacoustic             |
| models. Model I is a simple tonal and noise masking threshold        |
| generator, and Model II is a more sophisticated cochlear masking     |
| threshold generator. Model I is recommended for lower complexity     |
| applications whereas Model II gives better subjective quality at low |
| bit rates.                                                           |
|                                                                      |
| Layers I and II of mono, stereo, and joint stereo modes are supported.|
| Routines associated with a given layer are prefixed by "I_" for layer |
| 1 and "II_" for layer 2.                                             |
\======================================================================*/

/************************************************************************/
/*
/* read_samples()
/*
/* PURPOSE:  reads the PCM samples from a file to the buffer
/*
/*  SEMANTICS:
/* Reads #samples_read# number of shorts from #musicin# filepointer
/* into #sample_buffer[]#.  Returns the number of samples read.
/*
/************************************************************************/

unsigned long read_samples(musicin, sample_buffer, num_samples, frame_size)
FILE *musicin;
short sample_buffer[2304];
unsigned long num_samples, frame_size;
{
    unsigned long samples_read;
    static unsigned long samples_to_read;
    static char init = TRUE;

    if (init) {
        samples_to_read = num_samples;
        init = FALSE;
    }
    if (samples_to_read >= frame_size)
        samples_read = frame_size;
    else
        samples_read = samples_to_read;
    if ((samples_read =
         fread(sample_buffer, sizeof(short), (int)samples_read, musicin)) == 0)
        printf("Hit end of audio data\n");
    samples_to_read -= samples_read;
    if (samples_read < frame_size && samples_read > 0) {
        printf("Insufficient PCM input for one frame - fillout with zeros\n");
        for (; samples_read < frame_size; sample_buffer[samples_read++] = 0);
        samples_to_read = 0;
    }
    return(samples_read);
}

/************************************************************************/
/*
/* get_audio()
/*
/* PURPOSE:  reads a frame of audio data from a file to the buffer,
/*   aligns the data for future processing, and separates the
/*   left and right channels
/*
/*  SEMANTICS:
/* Calls read_samples() to read a frame of audio data from filepointer
/* #musicin# to #insampl[]#.  The data is shifted to make sure the data
/* is centered for the 1024pt window to be used by the psychoacoustic model,
/* and to compensate for the 256 sample delay from the filter bank. For
/* stereo, the channels are also demultiplexed into #buffer[0][]# and
/* #buffer[1][]#
/*
/************************************************************************/

unsigned long get_audio(musicin, buffer, num_samples, stereo, lay)
FILE *musicin;
short FAR buffer[2][1152];
unsigned long num_samples;
int stereo, lay;
```

```
{
   int j;
   short insamp[2304];
   unsigned long samples_read;

   if (lay == 1){
      if(stereo == 2){ /* layer 1, stereo */
         samples_read = read_samples(musicin, insamp, num_samples,
                                     (unsigned long) 768);
         for(j=0;j<448;j++) {
            if(j<64) {
               buffer[0][j] = buffer[0][j+384];
               buffer[1][j] = buffer[1][j+384];
            }
            else {
               buffer[0][j] = insamp[2*j-128];
               buffer[1][j] = insamp[2*j-127];
            }
         }
      }
      else { /* layer 1, mono */
         samples_read = read_samples(musicin, insamp, num_samples,
                                     (unsigned long) 384);
         for(j=0;j<448;j++){
            if(j<64) {
               buffer[0][j] = buffer[0][j+384];
               buffer[1][j] = 0;
            }
            else {
               buffer[0][j] = insamp[j-64];
               buffer[1][j] = 0;
            }
         }
      }
   }
   else {
      if(stereo == 2){ /* layer 2 (or 3), stereo */
         samples_read = read_samples(musicin, insamp, num_samples,
                                     (unsigned long) 2304);
         for(j=0;j<1152;j++) {
            buffer[0][j] = insamp[2*j];
            buffer[1][j] = insamp[2*j+1];
         }
      }
      else { /* layer 2 (or 3), mono */
         samples_read = read_samples(musicin, insamp, num_samples,
                                     (unsigned long) 1152);
         for(j=0;j<1152;j++){
            buffer[0][j] = insamp[j];
            buffer[1][j] = 0;
         }
      }
   }
   return(samples_read);
}

/************************************************************************/
/*
/* read_ana_window()
/*
/* PURPOSE:  Reads encoder window file "enwindow" into array #ana_win#
/*
/************************************************************************/

void read_ana_window(ana_win)
double FAR ana_win[HAN_SIZE];
{
    int i,j[4];
    FILE *fp;
    double f[4];
    char t[150];

    if (!(fp = OpenTableFile("enwindow") ) ) {
       printf("Please check analysis window table 'enwindow'\n");
       exit(1);
    }
```

```
    for (i=0;i<512;i+=4) {
       fgets(t, 150, fp);
       sscanf(t,"C[%d] = %lf C[%d] = %lf C[%d] = %lf C[%d] = %lf\n",
              j, f,j+1,f+1,j+2,f+2,j+3,f+3);
       if (i==j[0]) {
          ana_win[i] = f[0];
          ana_win[i+1] = f[1];
          ana_win[i+2] = f[2];
          ana_win[i+3] = f[3];
       }
       else {
          printf("Check index in analysis window table\n");
          exit(1);
       }
       fgets(t,150,fp);
    }
    fclose(fp);
}

/*************************************************************************/
/*
/* window_subband()
/*
/* PURPOSE:  Overlapping window on PCM samples
/*
/* SEMANTICS:
/* 32 16-bit pcm samples are scaled to fractional 2's complement and
/* concatenated to the end of the window buffer #x#. The updated window
/* buffer #x# is then windowed by the analysis window #c# to produce the
/* windowed sample #z#
/*
/*************************************************************************/

void window_subband(buffer, z, k)
short FAR **buffer;
double FAR z[HAN_SIZE];
int k;
{
    typedef double FAR XX[2][HAN_SIZE];
    static XX FAR *x;
    int i, j;
    static off[2] = {0,0};
    static char init = 0;
    static double FAR *c;
    if (!init) {
        c = (double FAR *) mem_alloc(sizeof(double) * HAN_SIZE, "window");
        read_ana_window(c);
        x = (XX FAR *) mem_alloc(sizeof(XX),"x");
        for (i=0;i<2;i++)
            for (j=0;j<HAN_SIZE;j++)
                (*x)[i][j] = 0;
        init = 1;
    }

    /* replace 32 oldest samples with 32 new samples */
    for (i=0;i<32;i++) (*x)[k][31-i+off[k]] = (double) *(*buffer)++/SCALE;
    /* shift samples into proper window positions */
    for (i=0;i<HAN_SIZE;i++) z[i] = (*x)[k][(i+off[k])&HAN_SIZE-1] * c[i];
    off[k] += 480;              /*offset is modulo (HAN_SIZE-1)*/
    off[k] &= HAN_SIZE-1;

}

/*************************************************************************/
/*
/* create_ana_filter()
/*
/* PURPOSE:  Calculates the analysis filter bank coefficients
/*
/* SEMANTICS:
/* Calculates the analysis filterbank coefficients and rounds to the
/* 9th decimal place accuracy of the filterbank tables in the ISO
/* document.  The coefficients are stored in #filter#
/*
/*************************************************************************/
```

```
void create_ana_filter(filter)
double FAR filter[SBLIMIT][64];
{
   register int i,k;

   for (i=0; i<32; i++)
      for (k=0; k<64; k++) {
          if ((filter[i][k] = 1e9*cos((double)((2*i+1)*(16-k)*PI64))) >= 0)
             modf(filter[i][k]+0.5, &filter[i][k]);
          else
             modf(filter[i][k]-0.5, &filter[i][k]);
          filter[i][k] *= 1e-9;
   }
}

/************************************************************************/
/*
/* filter_subband()
/*
/* PURPOSE:  Calculates the analysis filter bank coefficients
/*
/* SEMANTICS:
/*      The windowed samples #z# is filtered by the digital filter matrix #m#
/* to produce the subband samples #s#. This done by first selectively
/* picking out values from the windowed samples, and then multiplying
/* them by the filter matrix, producing 32 subband samples.
/*
/************************************************************************/

void filter_subband(z,s)
double FAR z[HAN_SIZE], s[SBLIMIT];
{
   double y[64];
   int i,j;
static char init = 0;
   typedef double MM[SBLIMIT][64];
static MM FAR *m;
#ifdef MS_DOS
   long    SIZE_OF_MM;
   SIZE_OF_MM      = SBLIMIT*64;
   SIZE_OF_MM      *= 8;
   if (!init) {
       m = (MM FAR *) mem_alloc(SIZE_OF_MM, "filter");
       create_ana_filter(*m);
       init = 1;
   }
#else
   if (!init) {
       m = (MM FAR *) mem_alloc(sizeof(MM), "filter");
       create_ana_filter(*m);
       init = 1;
   }
#endif
   for (i=0;i<64;i++) for (j=0, y[i] = 0;j<8;j++) y[i] += z[i+64*j];
   for (i=0;i<SBLIMIT;i++)
       for (j=0, s[i]= 0;j<64;j++) s[i] += (*m)[i][j] * y[j];
}

/************************************************************************/
/*
/* encode_info()
/*
/* PURPOSE:  Puts the syncword and header information on the output
/* bitstream.
/*
/************************************************************************/

void encode_info(fr_ps,bs)
frame_params *fr_ps;
Bit_stream_struc *bs;
{
        layer *info = fr_ps->header;

        putbits(bs,0xfff,12);                    /* syncword 12 bits */
        put1bit(bs,info->version);               /* ID        1 bit  */
        putbits(bs,4-info->lay,2);               /* layer     2 bits */
```

```
        put1bit(bs,!info->error_protection);     /* bit set => no err prot */
        putbits(bs,info->bitrate_index,4);
        putbits(bs,info->sampling_frequency,2);
        put1bit(bs,info->padding);
        put1bit(bs,info->extension);             /* private_bit */
        putbits(bs,info->mode,2);
        putbits(bs,info->mode_ext,2);
        put1bit(bs,info->copyright);
        put1bit(bs,info->original);
        putbits(bs,info->emphasis,2);
}

/************************************************************************/
/*
/* mod()
/*
/* PURPOSE:  Returns the absolute value of its argument
/*
/************************************************************************/

double mod(a)
double a;
{
    return (a > 0) ? a : -a;
}

/************************************************************************/
/*
/* I_combine_LR    (Layer I)
/* II_combine_LR   (Layer II)
/*
/* PURPOSE:Combines left and right channels into a mono channel
/*
/* SEMANTICS:  The average of left and right subband samples is put into
/* #joint_sample#
/*
/* Layer I and II differ in frame length and # subbands used
/*
/************************************************************************/

void I_combine_LR(sb_sample, joint_sample)
double FAR sb_sample[2][3][SCALE_BLOCK][SBLIMIT];
double FAR joint_sample[3][SCALE_BLOCK][SBLIMIT];
{   /* make a filtered mono for joint stereo */
    int sb, smp;

   for(sb = 0; sb<SBLIMIT; ++sb)
      for(smp = 0; smp<SCALE_BLOCK; ++smp)
        joint_sample[0][smp][sb] = .5 *
                    (sb_sample[0][0][smp][sb] + sb_sample[1][0][smp][sb]);
}

void II_combine_LR(sb_sample, joint_sample, sblimit)
double FAR sb_sample[2][3][SCALE_BLOCK][SBLIMIT];
double FAR joint_sample[3][SCALE_BLOCK][SBLIMIT];
int sblimit;
{  /* make a filtered mono for joint stereo */
   int sb, smp, sufr;

   for(sb = 0; sb<sblimit; ++sb)
      for(smp = 0; smp<SCALE_BLOCK; ++smp)
         for(sufr = 0; sufr<3; ++sufr)
            joint_sample[sufr][smp][sb] = .5 * (sb_sample[0][sufr][smp][sb]
                                          + sb_sample[1][sufr][smp][sb]);
}

/************************************************************************
/*
/* I_scale_factor_calc     (Layer I)
/* II_scale_factor_calc    (Layer II)
/*
/* PURPOSE:For each subband, calculate the scale factor for each set
/* of the 12 subband samples
/*
/* SEMANTICS:  Pick the scalefactor #multiple[]# just larger than the
/* absolute value of the peak subband sample of 12 samples,
```

```
/* and store the corresponding scalefactor index in #scalar#.
/*
/* Layer II has three sets of 12-subband samples for a given
/* subband.
/*
/************************************************************************/

void I_scale_factor_calc(sb_sample,scalar,stereo)
double FAR sb_sample[][3][SCALE_BLOCK][SBLIMIT];
unsigned int scalar[][3][SBLIMIT];
int stereo;
{
   int i,j, k;
   double s[SBLIMIT];

   for (k=0;k<stereo;k++) {
     for (i=0;i<SBLIMIT;i++)
       for (j=1, s[i] = mod(sb_sample[k][0][0][i]);j<SCALE_BLOCK;j++)
         if (mod(sb_sample[k][0][j][i]) > s[i])
            s[i] = mod(sb_sample[k][0][j][i]);

     for (i=0;i<SBLIMIT;i++)
       for (j=SCALE_RANGE-2,scalar[k][0][i]=0;j>=0;j--) /* $A 6/16/92 */
         if (s[i] <= multiple[j]) {
            scalar[k][0][i] = j;
            break;
         }
   }
}

/******************************** Layer II ******************************/

void II_scale_factor_calc(sb_sample,scalar,stereo,sblimit)
double FAR sb_sample[][3][SCALE_BLOCK][SBLIMIT];
unsigned int scalar[][3][SBLIMIT];
int stereo,sblimit;
{
  int i,j, k,t;
  double s[SBLIMIT];

  for (k=0;k<stereo;k++) for (t=0;t<3;t++) {
    for (i=0;i<sblimit;i++)
      for (j=1, s[i] = mod(sb_sample[k][t][0][i]);j<SCALE_BLOCK;j++)
        if (mod(sb_sample[k][t][j][i]) > s[i])
             s[i] = mod(sb_sample[k][t][j][i]);

  for (i=0;i<sblimit;i++)
    for (j=SCALE_RANGE-2,scalar[k][t][i]=0;j>=0;j--)    /* $A 6/16/92 */
      if (s[i] <= multiple[j]) {
         scalar[k][t][i] = j;
         break;
      }
      for (i=sblimit;i<SBLIMIT;i++) scalar[k][t][i] = SCALE_RANGE-1;
    }
}

/************************************************************************
/*
/* pick_scale  (Layer II)
/*
/* PURPOSE:For each subband, puts the smallest scalefactor of the 3
/* associated with a frame into #max_sc#.  This is used
/* used by Psychoacoustic Model I.
/* (I would recommend changin max_sc to min_sc)
/*
/************************************************************************/

void pick_scale(scalar, fr_ps, max_sc)
unsigned int scalar[2][3][SBLIMIT];
frame_params *fr_ps;
double FAR max_sc[2][SBLIMIT];
{
  int i,j,k,max;
  int stereo  = fr_ps->stereo;
  int sblimit = fr_ps->sblimit;
```

```
  for (k=0;k<stereo;k++)
    for (i=0;i<sblimit;max_sc[k][i] = multiple[max],i++)
      for (j=1, max = scalar[k][0][i];j<3;j++)
         if (max > scalar[k][j][i]) max = scalar[k][j][i];
  for (i=sblimit;i<SBLIMIT;i++) max_sc[0][i] = max_sc[1][i] = 1E-20;
}

/************************************************************************
/*
/* put_scale   (Layer I)
/*
/* PURPOSE:Sets #max_sc# to the scalefactor index in #scalar.
/* This is used by Psychoacoustic Model I
/*
/************************************************************************/

void put_scale(scalar, fr_ps, max_sc)
unsigned int scalar[2][3][SBLIMIT];
frame_params *fr_ps;
double FAR max_sc[2][SBLIMIT];
{
   int i,j,k, max;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;

   for (k=0;k<stereo;k++) for (i=0;i<SBLIMIT;i++)
        max_sc[k][i] = multiple[scalar[k][0][i]];
}

/************************************************************************
/*
/* II_transmission_pattern (Layer II only)
/*
/* PURPOSE:For a given subband, determines whether to send 1, 2, or
/* all 3 of the scalefactors, and fills in the scalefactor
/* select information accordingly
/*
/* SEMANTICS:  The subbands and channels are classified based on how much
/* the scalefactors changes over its three values (corresponding
/* to the 3 sets of 12 samples per subband).  The classification
/* will send 1 or 2 scalefactors instead of three if the scalefactors
/* do not change much.  The scalefactor select information,
/* #scfsi#, is filled in accordingly.
/*
/************************************************************************/

void II_transmission_pattern(scalar, scfsi, fr_ps)
unsigned int scalar[2][3][SBLIMIT];
unsigned int scfsi[2][SBLIMIT];
frame_params *fr_ps;
{
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int dscf[2];
   int class[2],i,j,k;
static int pattern[5][5] = {0x123, 0x122, 0x122, 0x133, 0x123,
                            0x113, 0x111, 0x111, 0x444, 0x113,
                            0x111, 0x111, 0x111, 0x333, 0x113,
                            0x222, 0x222, 0x222, 0x333, 0x123,
                            0x123, 0x122, 0x122, 0x133, 0x123};

   for (k=0;k<stereo;k++)
     for (i=0;i<sblimit;i++) {
       dscf[0] =  (scalar[k][0][i]-scalar[k][1][i]);
       dscf[1] =  (scalar[k][1][i]-scalar[k][2][i]);
       for (j=0;j<2;j++) {
         if (dscf[j]<=-3) class[j] = 0;
         else if (dscf[j] > -3 && dscf[j] <0) class[j] = 1;
              else if (dscf[j] == 0) class[j] = 2;
                   else if (dscf[j] > 0 && dscf[j] < 3) class[j] = 3;
                        else class[j] = 4;
       }
       switch (pattern[class[0]][class[1]]) {
         case 0x123 :    scfsi[k][i] = 0;
                         break;
         case 0x122 :    scfsi[k][i] = 3;
```

```
                         scalar[k][2][i] = scalar[k][1][i];
                         break;
         case 0x133 :    scfsi[k][i] = 3;
                         scalar[k][1][i] = scalar[k][2][i];
                         break;
         case 0x113 :    scfsi[k][i] = 1;
                         scalar[k][1][i] = scalar[k][0][i];
                         break;
         case 0x111 :    scfsi[k][i] = 2;
                         scalar[k][1][i] = scalar[k][2][i] = scalar[k][0][i];
                         break;
         case 0x222 :    scfsi[k][i] = 2;
                         scalar[k][0][i] = scalar[k][2][i] = scalar[k][1][i];
                         break;
         case 0x333 :    scfsi[k][i] = 2;
                         scalar[k][0][i] = scalar[k][1][i] = scalar[k][2][i];
                         break;
         case 0x444 :    scfsi[k][i] = 2;
                         if (scalar[k][0][i] > scalar[k][2][i])
                              scalar[k][0][i] = scalar[k][2][i];
                         scalar[k][1][i] = scalar[k][2][i] = scalar[k][0][i];
      }
   }
}

/************************************************************************
/*
/* I_encode_scale  (Layer I)
/* II_encode_scale (Layer II)
/*
/* PURPOSE:The encoded scalar factor information is arranged and
/* queued into the output fifo to be transmitted.
/*
/* For Layer II, the three scale factors associated with
/* a given subband and channel are transmitted in accordance
/* with the scfsi, which is transmitted first.
/*
/************************************************************************/

void I_encode_scale(scalar, bit_alloc, fr_ps, bs)
unsigned int scalar[2][3][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struc *bs;
{
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int i,j;

   for (i=0;i<SBLIMIT;i++) for (j=0;j<stereo;j++)
      if (bit_alloc[j][i]) putbits(bs,scalar[j][0][i],6);
}

/***************************** Layer II  ********************************/

void II_encode_scale(bit_alloc, scfsi, scalar, fr_ps, bs)
unsigned int bit_alloc[2][SBLIMIT], scfsi[2][SBLIMIT];
unsigned int scalar[2][3][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struc *bs;
{
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   int i,j,k;

   for (i=0;i<sblimit;i++) for (k=0;k<stereo;k++)
     if (bit_alloc[k][i])  putbits(bs,scfsi[k][i],2);

   for (i=0;i<sblimit;i++) for (k=0;k<stereo;k++)
     if (bit_alloc[k][i])  /* above jsbound, bit_alloc[0][i] == ba[1][i] */
        switch (scfsi[k][i]) {
           case 0: for (j=0;j<3;j++)
                     putbits(bs,scalar[k][j][i],6);
                   break;
           case 1:
```

```
          case 3: putbits(bs,scalar[k][0][i],6);
                  putbits(bs,scalar[k][2][i],6);
                  break;
          case 2: putbits(bs,scalar[k][0][i],6);
        }
}

/*======================================================================\
|                                                                       |
|      The following routines are done after the masking threshold      |
| has been calculated by the fft analysis routines in the Psychoacoustic |
| model. Using the MNR calculated, the actual number of bits allocated  |
| to each subband is found iteratively.                                 |
|                                                                       |
\======================================================================*/

/************************************************************************
/*
/* I_bits_for_nonoise  (Layer I)
/* II_bits_for_nonoise (Layer II)
/*
/* PURPOSE:Returns the number of bits required to produce a
/* mask-to-noise ratio better or equal to the noise/no_noise threshold.
/*
/* SEMANTICS:
/* bbal = # bits needed for encoding bit allocation
/* bsel = # bits needed for encoding scalefactor select information
/* banc = # bits needed for ancillary data (header info included)
/*
/* For each subband and channel, will add bits until one of the
/* following occurs:
/* - Hit maximum number of bits we can allocate for that subband
/* - MNR is better than or equal to the minimum masking level
/*   (NOISY_MIN_MNR)
/* Then the bits required for scalefactors, scfsi, bit allocation,
/* and the subband samples are tallied (#req_bits#) and returned.
/*
/* (NOISY_MIN_MNR) is the smallest MNR a subband can have before it is
/* counted as 'noisy' by the logic which chooses the number of JS
/* subbands.
/*
/* Joint stereo is supported.
/*
/************************************************************************/

static double snr[18] = {0.00, 7.00, 11.00, 16.00, 20.84,
                         25.28, 31.59, 37.75, 43.84,
                         49.89, 55.93, 61.96, 67.98, 74.01,
                         80.03, 86.05, 92.01, 98.01};

int I_bits_for_nonoise(perm_smr, fr_ps)
double FAR perm_smr[2][SBLIMIT];
frame_params *fr_ps;
{
   int i,j,k;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   int req_bits = 0;

   /* initial b_anc (header) allocation bits */
   req_bits = 32 + 4 * ( (jsbound * stereo) + (SBLIMIT-jsbound) );

   for(i=0; i<SBLIMIT; ++i)
     for(j=0; j<((i<jsbound)?stereo:1); ++j) {
       for(k=0;k<14; ++k)
         if( (-perm_smr[j][i] + snr[k]) >= NOISY_MIN_MNR)
           break; /* we found enough bits */
         if(stereo == 2 && i >= jsbound)     /* check other JS channel */
           for(;k<14; ++k)
             if( (-perm_smr[1-j][i] + snr[k]) >= NOISY_MIN_MNR) break;
         if(k>0) req_bits += (k+1)*SCALE_BLOCK + 6*((i>=jsbound)?stereo:1);
   }
   return req_bits;
}
```

```
/**************************** Layer II  ******************************/

int II_bits_for_nonoise(perm_smr, scfsi, fr_ps)
double FAR perm_smr[2][SBLIMIT];
unsigned int scfsi[2][SBLIMIT];
frame_params *fr_ps;
{
   int sb,ch,ba;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   al_table *alloc = fr_ps->alloc;
   int req_bits = 0, bbal = 0, berr = 0, banc = 32;
   int maxAlloc, sel_bits, sc_bits, smp_bits;
static int sfsPerScfsi[] = { 3,2,1,2 };    /* lookup # sfs per scfsi */

   /* added 92-08-11 shn */
   if (fr_ps->header->error_protection) berr=16; else berr=0;

   for (sb=0; sb<jsbound; ++sb)
     bbal += stereo * (*alloc)[sb][0].bits;
   for (sb=jsbound; sb<sblimit; ++sb)
     bbal += (*alloc)[sb][0].bits;
   req_bits = banc + bbal + berr;

   for(sb=0; sb<sblimit; ++sb)
     for(ch=0; ch<((sb<jsbound)?stereo:1); ++ch) {
       maxAlloc = (1<<(*alloc)[sb][0].bits)-1;
       sel_bits = sc_bits = smp_bits = 0;
       for(ba=0;ba<maxAlloc-1; ++ba)
         if( (-perm_smr[ch][sb] + snr[(*alloc)[sb][ba].quant+((ba>0)?1:0)])
             >= NOISY_MIN_MNR)
            break;      /* we found enough bits */
       if(stereo == 2 && sb >= jsbound) /* check other JS channel */
         for(;ba<maxAlloc-1; ++ba)
           if( (-perm_smr[1-ch][sb]+ snr[(*alloc)[sb][ba].quant+((ba>0)?1:0)])
               >= NOISY_MIN_MNR)
             break;
       if(ba>0) {
         smp_bits = SCALE_BLOCK * ((*alloc)[sb][ba].group * (*alloc)[sb][ba].bits);
         /* scale factor bits required for subband */
         sel_bits = 2;
         sc_bits  = 6 * sfsPerScfsi[scfsi[ch][sb]];
         if(stereo == 2 && sb >= jsbound) {
           /* each new js sb has L+R scfsis */
           sel_bits += 2;
           sc_bits  += 6 * sfsPerScfsi[scfsi[1-ch][sb]];
         }
         req_bits += smp_bits+sel_bits+sc_bits;
       }
   }
   return req_bits;
}

/**********************************************************************
/*
/* I_main_bit_allocation   (Layer I)
/* II_main_bit_allocation  (Layer II)
/*
/* PURPOSE:For joint stereo mode, determines which of the 4 joint
/* stereo modes is needed.  Then calls *_a_bit_allocation(), which
/* allocates bits for each of the subbands until there are no more bits
/* left, or the MNR is at the noise/no_noise threshold.
/*
/* SEMANTICS:
/*
/* For joint stereo mode, joint stereo is changed to stereo if
/* there are enough bits to encode stereo at or better than the
/* no-noise threshold (NOISY_MIN_MNR).  Otherwise, the system
/* iteratively allocates less bits by using joint stereo until one
/* of the following occurs:
/* - there are no more noisy subbands (MNR >= NOISY_MIN_MNR)
/* - mode_ext has been reduced to 0, which means that all but the
/*   lowest 4 subbands have been converted from stereo to joint
/*   stereo, and no more subbands may be converted
/*
```

```
/*     This function calls *_bits_for_nonoise() and *_a_bit_allocation().
/*
/************************************************************************/

void I_main_bit_allocation(perm_smr, bit_alloc, adb, fr_ps)
double FAR perm_smr[2][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
int *adb;
frame_params *fr_ps;
{
   int  noisy_sbs;
   int  mode, mode_ext, lay, i;
   int  rq_db, av_db = *adb;
static  int init = 0;

   if(init == 0) {
     /* rearrange snr for layer I */
     snr[2] = snr[3];
     for (i=3;i<16;i++) snr[i] = snr[i+2];
     init = 1;
   }

   if((mode = fr_ps->actual_mode) == MPG_MD_JOINT_STEREO) {
     fr_ps->header->mode = MPG_MD_STEREO;
     fr_ps->header->mode_ext = 0;
     fr_ps->jsbound = fr_ps->sblimit;
     if(rq_db = I_bits_for_nonoise(perm_smr, fr_ps) > *adb) {
       fr_ps->header->mode = MPG_MD_JOINT_STEREO;
       mode_ext = 4;           /* 3 is least severe reduction */
       lay = fr_ps->header->lay;
       do {
          --mode_ext;
          fr_ps->jsbound = js_bound(lay, mode_ext);
          rq_db = I_bits_for_nonoise(perm_smr, fr_ps);
       } while( (rq_db > *adb) && (mode_ext > 0));
       fr_ps->header->mode_ext = mode_ext;
     }    /* well we either eliminated noisy sbs or mode_ext == 0 */
   }
   noisy_sbs = I_a_bit_allocation(perm_smr, bit_alloc, adb, fr_ps);
}

/***************************** Layer II  *******************************/

void II_main_bit_allocation(perm_smr, scfsi, bit_alloc, adb, fr_ps)
double FAR perm_smr[2][SBLIMIT];
unsigned int scfsi[2][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
int *adb;
frame_params *fr_ps;
{
   int  noisy_sbs, nn;
   int  mode, mode_ext, lay;
   int  rq_db, av_db = *adb;

   if((mode = fr_ps->actual_mode) == MPG_MD_JOINT_STEREO) {
     fr_ps->header->mode = MPG_MD_STEREO;
     fr_ps->header->mode_ext = 0;
     fr_ps->jsbound = fr_ps->sblimit;
     if((rq_db=II_bits_for_nonoise(perm_smr, scfsi, fr_ps)) > *adb) {
       fr_ps->header->mode = MPG_MD_JOINT_STEREO;
       mode_ext = 4;           /* 3 is least severe reduction */
       lay = fr_ps->header->lay;
       do {
         --mode_ext;
         fr_ps->jsbound = js_bound(lay, mode_ext);
         rq_db = II_bits_for_nonoise(perm_smr, scfsi, fr_ps);
       } while( (rq_db > *adb) && (mode_ext > 0));
       fr_ps->header->mode_ext = mode_ext;
     }    /* well we either eliminated noisy sbs or mode_ext == 0 */
   }
   noisy_sbs = II_a_bit_allocation(perm_smr, scfsi, bit_alloc, adb, fr_ps);
}

/************************************************************************
/*
/* I_a_bit_allocation  (Layer I)
```

```
/* II_a_bit_allocation (Layer II)
/*
/* PURPOSE:Adds bits to the subbands with the lowest mask-to-noise
/* ratios, until the maximum number of bits for the subband has
/* been allocated.
/*
/* SEMANTICS:
/* 1. Find the subband and channel with the smallest MNR (#min_sb#,
/*    and #min_ch#)
/* 2. Calculate the increase in bits needed if we increase the bit
/*    allocation to the next higher level
/* 3. If there are enough bits available for increasing the resolution
/*    in #min_sb#, #min_ch#, and the subband has not yet reached its
/*    maximum allocation, update the bit allocation, MNR, and bits
/*    available accordingly
/* 4. Repeat until there are no more bits left, or no more available
/*    subbands. (A subband is still available until the maximum
/*    number of bits for the subband has been allocated, or there
/*    aren't enough bits to go to the next higher resolution in the
/*    subband.)
/*
/*************************************************************************/

int I_a_bit_allocation(perm_smr, bit_alloc, adb, fr_ps) /* return noisy sbs */
double FAR perm_smr[2][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
int *adb;
frame_params *fr_ps;
{
   int i, k, smpl_bits, scale_bits, min_sb, min_ch, oth_ch;
   int bspl, bscf, ad, noisy_sbs, done = 0, bbal ;
   double mnr[2][SBLIMIT], small;
   char used[2][SBLIMIT];
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   al_table *alloc = fr_ps->alloc;
static char init= 0;
static int banc=32, berr=0;

   if (!init) {
      init = 1;
      if (fr_ps->header->error_protection) berr = 16;  /* added 92-08-11 shn */
   }
   bbal = 4 * ( (jsbound * stereo) + (SBLIMIT-jsbound) );
   *adb -= bbal + berr + banc;
   ad= *adb;

   for (i=0;i<SBLIMIT;i++) for (k=0;k<stereo;k++) {
     mnr[k][i]=snr[0]-perm_smr[k][i];
     bit_alloc[k][i] = 0;
     used[k][i] = 0;
   }
   bspl = bscf = 0;

   do  {
     /* locate the subband with minimum SMR */
     small = mnr[0][0]+1;    min_sb = -1; min_ch = -1;
     for (i=0;i<SBLIMIT;i++) for (k=0;k<stereo;k++)
       /* go on only if there are bits left */
       if (used[k][i] != 2 && small > mnr[k][i]) {
         small = mnr[k][i];
         min_sb = i;  min_ch = k;
       }
     if(min_sb > -1) {   /* there was something to find */
       /* first step of bit allocation is biggest */
       if (used[min_ch][min_sb])  { smpl_bits = SCALE_BLOCK; scale_bits = 0; }
       else                       { smpl_bits = 24; scale_bits = 6; }
       if(min_sb >= jsbound)        scale_bits *= stereo;

       /* check to see enough bits were available for */
       /* increasing resolution in the minimum band */

       if (ad >= bspl + bscf + scale_bits + smpl_bits) {
         bspl += smpl_bits; /* bit for subband sample */
         bscf += scale_bits; /* bit for scale factor */
```

```
          bit_alloc[min_ch][min_sb]++;
          used[min_ch][min_sb] = 1; /* subband has bits */
          mnr[min_ch][min_sb] = -perm_smr[min_ch][min_sb]
                                + snr[bit_alloc[min_ch][min_sb]];
          /* Check if subband has been fully allocated max bits */
          if (bit_alloc[min_ch][min_sb] ==  14 ) used[min_ch][min_sb] = 2;
        }
        else            /* no room to improve this band */
          used[min_ch][min_sb] = 2; /*   for allocation anymore */
        if(stereo == 2 && min_sb >= jsbound) {
          oth_ch = 1-min_ch;  /* joint-st : fix other ch */
          bit_alloc[oth_ch][min_sb] = bit_alloc[min_ch][min_sb];
          used[oth_ch][min_sb] = used[min_ch][min_sb];
          mnr[oth_ch][min_sb] = -perm_smr[oth_ch][min_sb]
                                + snr[bit_alloc[oth_ch][min_sb]];
        }
      }
   } while(min_sb>-1);     /* i.e. still some sub-bands to find */

   /* Calculate the number of bits left, add on to pointed var */
   ad -= bspl+bscf;
   *adb = ad;

   /* see how many channels are noisy */
   noisy_sbs = 0; small = mnr[0][0];
   for(k=0; k<stereo; ++k) {
     for(i = 0; i< SBLIMIT; ++i) {
       if(mnr[k][i] < NOISY_MIN_MNR)   ++noisy_sbs;
       if(small > mnr[k][i])           small = mnr[k][i];
     }
   }
   return noisy_sbs;
}

/****************************** Layer II  ********************************/

int II_a_bit_allocation(perm_smr, scfsi, bit_alloc, adb, fr_ps)
double FAR perm_smr[2][SBLIMIT];
unsigned int scfsi[2][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
int *adb;
frame_params *fr_ps;
{
   int i, min_ch, min_sb, oth_ch, k, increment, scale, seli, ba;
   int bspl, bscf, bsel, ad, noisy_sbs, bbal=0;
   double mnr[2][SBLIMIT], small;
   char used[2][SBLIMIT];
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   al_table *alloc = fr_ps->alloc;
static char init= 0;
static int banc=32, berr=0;
static int sfsPerScfsi[] = { 3,2,1,2 };    /* lookup # sfs per scfsi */

   if (!init) {
       init = 1;
       if (fr_ps->header->error_protection) berr=16; /* added 92-08-11 shn */
   }
   for (i=0; i<jsbound; ++i)
     bbal += stereo * (*alloc)[i][0].bits;
   for (i=jsbound; i<sblimit; ++i)
     bbal += (*alloc)[i][0].bits;
   *adb -= bbal + berr + banc;
   ad = *adb;

   for (i=0;i<sblimit;i++) for (k=0;k<stereo;k++) {
     mnr[k][i]=snr[0]-perm_smr[k][i];
     bit_alloc[k][i] = 0;
     used[k][i] = 0;
   }
   bspl = bscf = bsel = 0;

   do  {
     /* locate the subband with minimum SMR */
     small = 999999.0; min_sb = -1; min_ch = -1;
```

```
    for (i=0;i<sblimit;i++) for(k=0;k<stereo;++k)
      if (used[k][i]  != 2 && small > mnr[k][i]) {
        small = mnr[k][i];
        min_sb = i;  min_ch = k;
    }
    if(min_sb > -1) {   /* there was something to find */
      /* find increase in bit allocation in subband [min] */
      increment = SCALE_BLOCK * ((*alloc)[min_sb][bit_alloc[min_ch][min_sb]+1].group *
                       (*alloc)[min_sb][bit_alloc[min_ch][min_sb]+1].bits);
      if (used[min_ch][min_sb])
        increment -= SCALE_BLOCK * ((*alloc)[min_sb][bit_alloc[min_ch][min_sb]].group*
                          (*alloc)[min_sb][bit_alloc[min_ch][min_sb]].bits);

      /* scale factor bits required for subband [min] */
      oth_ch = 1 - min_ch;    /* above js bound, need both chans */
      if (used[min_ch][min_sb]) scale = seli = 0;
      else {          /* this channel had no bits or scfs before */
        seli = 2;
        scale = 6 * sfsPerScfsi[scfsi[min_ch][min_sb]];
        if(stereo == 2 && min_sb >= jsbound) {
          /* each new js sb has L+R scfsis */
          seli += 2;
          scale += 6 * sfsPerScfsi[scfsi[oth_ch][min_sb]];
        }
      }
      /* check to see enough bits were available for */
      /* increasing resolution in the minimum band */
      if (ad >= bspl + bscf + bsel + seli + scale + increment) {
        ba = ++bit_alloc[min_ch][min_sb]; /* next up alloc */
        bspl += increment;  /* bits for subband sample */
        bscf += scale;      /* bits for scale factor */
        bsel += seli;       /* bits for scfsi code */
        used[min_ch][min_sb] = 1; /* subband has bits */
        mnr[min_ch][min_sb] = -perm_smr[min_ch][min_sb] +
                              snr[(*alloc)[min_sb][ba].quant+1];
        /* Check if subband has been fully allocated max bits */
        if (ba >= (1<<(*alloc)[min_sb][0].bits)-1) used[min_ch][min_sb] = 2;
      }
      else used[min_ch][min_sb] = 2; /* can't increase this alloc */
      if(min_sb >= jsbound && stereo == 2) {
        /* above jsbound, alloc applies L+R */
        ba = bit_alloc[oth_ch][min_sb] = bit_alloc[min_ch][min_sb];
        used[oth_ch][min_sb] = used[min_ch][min_sb];
        mnr[oth_ch][min_sb] = -perm_smr[oth_ch][min_sb] +
                              snr[(*alloc)[min_sb][ba].quant+1];
      }
    }
  } while(min_sb > -1);   /* until could find no channel */
  /* Calculate the number of bits left */
  ad -= bspl+bscf+bsel;   *adb = ad;
  for (i=sblimit;i<SBLIMIT;i++) for (k=0;k<stereo;k++) bit_alloc[k][i]=0;

  noisy_sbs = 0;  small = mnr[0][0];      /* calc worst noise in case */
  for(k=0;k<stereo;++k) {
    for (i=0;i<sblimit;i++) {
      if (small > mnr[k][i]) small = mnr[k][i];
      if(mnr[k][i] < NOISY_MIN_MNR) ++noisy_sbs; /* noise is not masked */

    }
  }
  return noisy_sbs;
}

/************************************************************************
/*
/* I_subband_quantization  (Layer I)
/* II_subband_quantization (Layer II)
/*
/* PURPOSE:Quantizes subband samples to appropriate number of bits
/*
/* SEMANTICS:  Subband samples are divided by their scalefactors, which
/* makes the quantization more efficient. The scaled samples are
/* quantized by the function a*x+b, where a and b are functions of
/* the number of quantization levels. The result is then truncated
/* to the appropriate number of bits and the MSB is inverted.
/*
```

```
/* Note that for fractional 2's complement, inverting the MSB for a
/* negative number x is equivalent to adding 1 to it.
/*
/************************************************************************/

static double a[17] = {
  0.750000000, 0.625000000, 0.875000000, 0.562500000, 0.937500000,
  0.968750000, 0.984375000, 0.992187500, 0.996093750, 0.998046875,
  0.999023438, 0.999511719, 0.999755859, 0.999877930, 0.999938965,
  0.999969482, 0.999984741 };

static double b[17] = {
  -0.250000000, -0.375000000, -0.125000000, -0.437500000, -0.062500000,
  -0.031250000, -0.015625000, -0.007812500, -0.003906250, -0.001953125,
  -0.000976563, -0.000488281, -0.000244141, -0.000122070, -0.000061035,
  -0.000030518, -0.000015259 };

void I_subband_quantization(scalar, sb_samples, j_scale, j_samps,
                            bit_alloc, sbband, fr_ps)
unsigned int scalar[2][3][SBLIMIT];
double FAR sb_samples[2][3][SCALE_BLOCK][SBLIMIT];
unsigned int j_scale[3][SBLIMIT];
double FAR j_samps[3][SCALE_BLOCK][SBLIMIT]; /* L+R for j-stereo if necess */
unsigned int bit_alloc[2][SBLIMIT];
unsigned int FAR sbband[2][3][SCALE_BLOCK][SBLIMIT];
frame_params *fr_ps;
{
   int i, j, k, n, sig;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   double d;
static char init = 0;

   if (!init) {
     init = 1;
     /* rearrange quantization coef to correspond to layer I table */
     a[1] = a[2]; b[1] = b[2];
     for (i=2;i<15;i++) { a[i] = a[i+2]; b[i] = b[i+2]; }
   }
   for (j=0;j<SCALE_BLOCK;j++) for (i=0;i<SBLIMIT;i++)
     for (k=0;k<((i<jsbound)?stereo:1);k++)
       if (bit_alloc[k][i]) {
         /* for joint stereo mode, have to construct a single subband stream
            for the js channels.  At present, we calculate a set of mono
            subband samples and pass them through the scaling system to
            generate an alternate normalised sample stream.

            Could normalise both streams (divide by their scfs), then average
            them.  In bad conditions, this could give rise to spurious
            cancellations.  Instead, we could just select the sb stream from
            the larger channel (higher scf), in which case _that_ channel
            would be 'properly' reconstructed, and the mate would just be a
            scaled version.  Spec recommends averaging the two (unnormalised)
            subband channels, then normalising this new signal without
            actually sending this scale factor... This means looking ahead.
         */
         if(stereo == 2 && i>=jsbound)
           /* use the joint data passed in */
           d = j_samps[0][j][i] / multiple[j_scale[0][i]];
         else
           d = sb_samples[k][0][j][i] / multiple[scalar[k][0][i]];
         /* scale and quantize floating point sample */
         n = bit_alloc[k][i];
         d = d * a[n-1] + b[n-1];
         /* extract MSB N-1 bits from the floating point sample */
         if (d >= 0) sig = 1;
         else { sig = 0; d += 1.0; }
         sbband[k][0][j][i] = (unsigned int) (d * (double) (1L<<n));
         /* tag the inverted sign bit to sbband at position N */
         if (sig) sbband[k][0][j][i] |= 1<<n;
       }
}

/***************************** Layer II  ********************************/
```

```
void II_subband_quantization(scalar, sb_samples, j_scale, j_samps,
                             bit_alloc, sbband, fr_ps)
unsigned int scalar[2][3][SBLIMIT];
double FAR sb_samples[2][3][SCALE_BLOCK][SBLIMIT];
unsigned int j_scale[3][SBLIMIT];
double FAR j_samps[3][SCALE_BLOCK][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
unsigned int FAR sbband[2][3][SCALE_BLOCK][SBLIMIT];
frame_params *fr_ps;
{
   int i, j, k, s, n, qnt, sig;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   unsigned int stps;
   double d;
   al_table *alloc = fr_ps->alloc;

   for (s=0;s<3;s++)
     for (j=0;j<SCALE_BLOCK;j++)
       for (i=0;i<sblimit;i++)
         for (k=0;k<((i<jsbound)?stereo:1);k++)
           if (bit_alloc[k][i]) {
             /* scale and quantize floating point sample */
             if(stereo == 2 && i>=jsbound)       /* use j-stereo samples */
               d = j_samps[s][j][i] / multiple[j_scale[s][i]];
             else
               d = sb_samples[k][s][j][i] / multiple[scalar[k][s][i]];
             if (mod(d) > 1.0)
               printf("Not scaled properly %d %d %d %d\n",k,s,j,i);
             qnt = (*alloc)[i][bit_alloc[k][i]].quant;
             d = d * a[qnt] + b[qnt];
             /* extract MSB N-1 bits from the floating point sample */
             if (d >= 0) sig = 1;
             else { sig = 0; d += 1.0; }
             n = 0;
#ifndef MS_DOS
             stps = (*alloc)[i][bit_alloc[k][i]].steps;
             while ((1L<<n) < stps) n++;
#else
             while  ( ( (unsigned long)(1L<<(long)n) <
                       ((unsigned long) ((*alloc)[i][bit_alloc[k][i]].steps)
                        & 0xffff
                        )
                       ) && ( n <16)
                     ) n++;
#endif
             n--;
             sbband[k][s][j][i] = (unsigned int) (d * (double) (1L<<n));
             /* tag the inverted sign bit to sbband at position N */
             /* The bit inversion is a must for grouping with 3,5,9 steps
                so it is done for all subbands */
             if (sig) sbband[k][s][j][i] |= 1<<n;
           }
           for (s=0;s<3;s++)
             for (j=sblimit;j<SBLIMIT;j++)
               for (i=0;i<SCALE_BLOCK;i++) for (k=0;k<stereo;k++) sbband[k][s][i][j] = 0;
}

/************************************************************************
/*
/* I_encode_bit_alloc  (Layer I)
/* II_encode_bit_alloc (Layer II)
/*
/* PURPOSE:Writes bit allocation information onto bitstream
/*
/* Layer I uses 4 bits/subband for bit allocation information,
/* and Layer II uses 4,3,2, or 0 bits depending on the
/* quantization table used.
/*
/************************************************************************/

void I_encode_bit_alloc(bit_alloc, fr_ps, bs)
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struc *bs;
```

```
{
   int i,k;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;

   for (i=0;i<SBLIMIT;i++)
     for (k=0;k<((i<jsbound)?stereo:1);k++) putbits(bs,bit_alloc[k][i],4);
}

/***************************** Layer II  ********************************/

void II_encode_bit_alloc(bit_alloc, fr_ps, bs)
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struc *bs;
{
   int i,k;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   al_table *alloc = fr_ps->alloc;

   for (i=0;i<sblimit;i++)
     for (k=0;k<((i<jsbound)?stereo:1);k++)
       putbits(bs,bit_alloc[k][i],(*alloc)[i][0].bits);
}

/************************************************************************
/*
/* I_sample_encoding   (Layer I)
/* II_sample_encoding  (Layer II)
/*
/* PURPOSE:Put one frame of subband samples on to the bitstream
/*
/* SEMANTICS:  The number of bits allocated per sample is read from
/* the bit allocation information #bit_alloc#.  Layer 2
/* supports writing grouped samples for quantization steps
/* that are not a power of 2.
/*
/************************************************************************/

void I_sample_encoding(sbband, bit_alloc, fr_ps, bs)
unsigned int FAR sbband[2][3][SCALE_BLOCK][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struc *bs;
{
   int i,j,k;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;

   for(j=0;j<SCALE_BLOCK;j++) {
     for(i=0;i<SBLIMIT;i++)
       for(k=0;k<((i<jsbound)?stereo:1);k++)
         if(bit_alloc[k][i]) putbits(bs,sbband[k][0][j][i],bit_alloc[k][i]+1);
   }
}

/***************************** Layer II  ********************************/

void II_sample_encoding(sbband, bit_alloc, fr_ps, bs)
unsigned int FAR sbband[2][3][SCALE_BLOCK][SBLIMIT];
unsigned int bit_alloc[2][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struc *bs;
{
   unsigned int temp;
   unsigned int i,j,k,s,x,y;
   int stereo  = fr_ps->stereo;
   int sblimit = fr_ps->sblimit;
   int jsbound = fr_ps->jsbound;
   al_table *alloc = fr_ps->alloc;

   for (s=0;s<3;s++)
```

```
     for (j=0;j<SCALE_BLOCK;j+=3)
       for (i=0;i<sblimit;i++)
         for (k=0;k<((i<jsbound)?stereo:1);k++)
           if (bit_alloc[k][i]) {
             if ((*alloc)[i][bit_alloc[k][i]].group == 3) {
               for (x=0;x<3;x++) putbits(bs,sbband[k][s][j+x][i],
                                  (*alloc)[i][bit_alloc[k][i]].bits);
             }
             else {
               y =(*alloc)[i][bit_alloc[k][i]].steps;
               temp = sbband[k][s][j][i] +
                      sbband[k][s][j+1][i] * y +
                      sbband[k][s][j+2][i] * y * y;
               putbits(bs,temp,(*alloc)[i][bit_alloc[k][i]].bits);
             }
           }
}

/************************************************************************
/*
/* encode_CRC
/*
/************************************************************************/

void encode_CRC(crc, bs)
unsigned int crc;
Bit_stream_struc *bs;
{
   putbits(bs, crc, 16);
}
```

C.3.6 encoder.h

```
/**********************************************************************
Copyright (c) 1991 ISO/IEC JTC1 SC29 WG1, All Rights Reserved
encoder.h
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Doulas Wong,        start of version 1.0 records          *
 *          Davis Pan                                                 *
 * 5/10/91  W. Joseph Carter    Reorganized & renamed all ".h" files  *
 *                              into "common.h" and "encoder.h".      *
 *                              Ported to Macintosh and Unix.         *
 *                              Added function prototypes for more    *
 *                              rigorous type checking.               *
 * 27jun91  dpwe (Aware)        moved "alloc_*" types, pros to common *
 *                              Use ifdef PROTO_ARGS for prototypes   *
 *                              prototypes reflect frame_params struct*
 * 7/10/91  Earle Jennings      Conversion of all floats to FLOAT     *
 * 10/3/91  Don H. Lee          implemented CRC-16 error protection   *
 *                              Additions and revisions are marked    *
 *                              with "dhl" for clarity                *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most   *
 *                              important fixes involved changing     *
 *                              16-bit ints to long or unsigned in    *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args.     *
 *                              Removed "Other Joint Stereo" option   *
 *                              and made bitrate be total channel     *
 *                              bitrate, irrespective of the mode.    *
 *                              Fixed many small bugs & reorganized.  *
 *                              Modified some function prototypes.    *
 * 7/27/92  Masahiro Iwadare    FFT modifications for Layer 3         *
 * 8/3/92   Mike Li             removed declaration _stklen for DOS.  *
 * 9/22/92  jddevine@aware.com  Fix protos for _scale_factor_calc()   *
```

```
 * 11/04/94 Jon Rowlands        Fix protos for usage()                *
 **********************************************************************/

/***********************************************************************
*
*  Encoder Include Files
*
***********************************************************************/

/***********************************************************************
*
*  Encoder Definitions
*
***********************************************************************/

/* General Definitions */

/* Default Input Arguments (for command line control) */

#define DFLT_LAY        2      /* default encoding layer is II */
#define DFLT_MOD        's'    /* default mode is stereo */
#define DFLT_PSY        2      /* default psych model is 2 */
#define DFLT_SFQ        44.1   /* default input sampling rate is 44.1 kHz */
#define DFLT_BRT        384    /* default total output bitrate is 384 kbps */
#define DFLT_EMP        'n'    /* default de-emphasis is none */
#define DFLT_EXT        ".mpg" /* default output file extension */

#define FILETYPE_ENCODE 'TEXT'
#define CREATOR_ENCODE  'MpgD'

/* This is the smallest MNR a subband can have before it is counted
   as 'noisy' by the logic which chooses the number of JS subbands */

#define NOISY_MIN_MNR   0.0

/* Psychacoustic Model 1 Definitions */

#define CB_FRACTION     0.33
#define MAX_SNR         1000
#define NOISE           10
#define TONE            20
#define DBMIN           -200.0
#define LAST            -1
#define STOP            -100
#define POWERNORM       90.3090 /* = 20 * log10(32768) to normalize */
                                /* max output power to 96 dB per spec */

/* Psychoacoustic Model 2 Definitions */

#define LOGBLKSIZE      10
#define BLKSIZE         1024
#define HBLKSIZE        513
#define CBANDS          63
#define LXMIN           32.0

/***********************************************************************
*
*  Encoder Type Definitions
*
***********************************************************************/

/* Psychoacoustic Model 1 Type Definitions */

typedef int        IFFT2[FFT_SIZE/2];
typedef int        IFFT[FFT_SIZE];
typedef double     D9[9];
typedef double     D10[10];
typedef double     D640[640];
typedef double     D1408[1408];
typedef double     DFFT2[FFT_SIZE/2];
typedef double     DFFT[FFT_SIZE];
typedef double     DSBL[SBLIMIT];
typedef double     D2SBL[2][SBLIMIT];

typedef struct {
        int        line;
```

```
        double     bark, hear, x;
} g_thres, *g_ptr;

typedef struct {
        double     x;
        int        type, next, map;
} mask, *mask_ptr;

/* Psychoacoustic Model 2 Type Definitions */

typedef int        ICB[CBANDS];
typedef int        IHBLK[HBLKSIZE];
typedef FLOAT      F32[32];
typedef FLOAT      F2_32[2][32];
typedef FLOAT      FCB[CBANDS];
typedef FLOAT      FCBCB[CBANDS][CBANDS];
typedef FLOAT      FBLK[BLKSIZE];
typedef FLOAT      FHBLK[HBLKSIZE];
typedef FLOAT      F2HBLK[2][HBLKSIZE];
typedef FLOAT      F22HBLK[2][2][HBLKSIZE];
typedef double     DCB[CBANDS];

/***********************************************************************
*
*  Encoder Function Prototype Declarations
*
***********************************************************************/

/* The following functions are in the file "musicin.c" */

#ifdef        PROTO_ARGS
extern void   obtain_parameters(frame_params*, int*, unsigned long*,
                           char[MAX_NAME_SIZE], char[MAX_NAME_SIZE]);
extern void   parse_args(int, char**, frame_params*, int*, unsigned long*,
                           char[MAX_NAME_SIZE], char[MAX_NAME_SIZE]);
extern void   print_config(frame_params*, int*, unsigned long*,
                           char[MAX_NAME_SIZE], char[MAX_NAME_SIZE]);
static void   usage(void);
extern void   aiff_check(char*, IFF_AIFF*);
#else
extern void   obtain_parameters();
extern void   parse_args();
extern void   print_config();
static void   usage();
extern void   aiff_check();
#endif

/* The following functions are in the file "encode.c" */

#ifdef        PROTO_ARGS
extern unsigned long    read_samples(FILE*, short[2304], unsigned long,
                           unsigned long);
extern unsigned long    get_audio(FILE*, short[2][1152], unsigned long,
                           int, int);
extern void   read_ana_window(double[HAN_SIZE]);
extern void   window_subband(short**, double[HAN_SIZE], int);
extern void   create_ana_filter(double[SBLIMIT][64]);
extern void   filter_subband(double[HAN_SIZE], double[SBLIMIT]);
extern void   encode_info(frame_params*, Bit_stream_struc*);
extern double mod(double);
extern void   I_combine_LR(double[2][3][SCALE_BLOCK][SBLIMIT],
                           double[3][SCALE_BLOCK][SBLIMIT]);
extern void   II_combine_LR(double[2][3][SCALE_BLOCK][SBLIMIT],
                           double[3][SCALE_BLOCK][SBLIMIT], int);
extern void   I_scale_factor_calc(double[][3][SCALE_BLOCK][SBLIMIT],
                           unsigned int[][3][SBLIMIT], int);
extern void   II_scale_factor_calc(double[][3][SCALE_BLOCK][SBLIMIT],
                           unsigned int[][3][SBLIMIT], int, int);
extern void   pick_scale(unsigned int[2][3][SBLIMIT], frame_params*,
                           double[2][SBLIMIT]);
extern void   put_scale(unsigned int[2][3][SBLIMIT], frame_params*,
                           double[2][SBLIMIT]);
extern void   II_transmission_pattern(unsigned int[2][3][SBLIMIT],
                           unsigned int[2][SBLIMIT], frame_params*);
extern void   II_encode_scale(unsigned int[2][SBLIMIT],
                           unsigned int[2][SBLIMIT],
```

```
                           unsigned int[2][3][SBLIMIT], frame_params*,
                           Bit_stream_struc*);
extern void   I_encode_scale(unsigned int[2][3][SBLIMIT],
                           unsigned int[2][SBLIMIT], frame_params*,
                           Bit_stream_struc*);
extern int    II_bits_for_nonoise(double[2][SBLIMIT], unsigned int[2][SBLIMIT],
                           frame_params*);
extern void   II_main_bit_allocation(double[2][SBLIMIT],
                           unsigned int[2][SBLIMIT], unsigned int[2][SBLIMIT],
                           int*, frame_params*);
extern int    II_a_bit_allocation(double[2][SBLIMIT], unsigned int[2][SBLIMIT],
                           unsigned int[2][SBLIMIT], int*, frame_params*);
extern int    I_bits_for_nonoise(double[2][SBLIMIT], frame_params*);
extern void   I_main_bit_allocation(double[2][SBLIMIT],
                           unsigned int[2][SBLIMIT], int*, frame_params*);
extern int    I_a_bit_allocation(double[2][SBLIMIT], unsigned int[2][SBLIMIT],
                           int*, frame_params*);
extern void   I_subband_quantization(unsigned int[2][3][SBLIMIT],
                           double[2][3][SCALE_BLOCK][SBLIMIT], unsigned int[3][SBLIMIT],
                           double[3][SCALE_BLOCK][SBLIMIT], unsigned int[2][SBLIMIT],
                           unsigned int[2][3][SCALE_BLOCK][SBLIMIT], frame_params*);
extern void   II_subband_quantization(unsigned int[2][3][SBLIMIT],
                           double[2][3][SCALE_BLOCK][SBLIMIT], unsigned int[3][SBLIMIT],
                           double[3][SCALE_BLOCK][SBLIMIT], unsigned int[2][SBLIMIT],
                           unsigned int[2][3][SCALE_BLOCK][SBLIMIT], frame_params*);
extern void   II_encode_bit_alloc(unsigned int[2][SBLIMIT], frame_params*,
                           Bit_stream_struc*);
extern void   I_encode_bit_alloc(unsigned int[2][SBLIMIT], frame_params*,
                           Bit_stream_struc*);
extern void   I_sample_encoding(unsigned int[2][3][SCALE_BLOCK][SBLIMIT],
                           unsigned int[2][SBLIMIT], frame_params*,
                           Bit_stream_struc*);
extern void   II_sample_encoding(unsigned int[2][3][SCALE_BLOCK][SBLIMIT],
                           unsigned int[2][SBLIMIT], frame_params*,
                           Bit_stream_struc*);
extern void   encode_CRC(unsigned int, Bit_stream_struc*);
#else
extern unsigned long  read_samples();
extern unsigned long  get_audio();
extern void        read_ana_window();
extern void        window_subband();
extern void        create_ana_filter();
extern void        filter_subband();
extern void        encode_info();
extern double      mod();
extern void        I_combine_LR();
extern void        II_combine_LR();
extern void        I_scale_factor_calc();
extern void        II_scale_factor_calc();
extern void        pick_scale();
extern void        put_scale();
extern void        II_transmission_pattern();
extern void        II_encode_scale();
extern void        I_encode_scale();
extern int         II_bits_for_nonoise();
extern void        II_main_bit_allocation();
extern int         II_a_bit_allocation();
extern int         I_bits_for_nonoise();
extern void        I_main_bit_allocation();
extern int         I_a_bit_allocation();
extern void        I_subband_quantization();
extern void        II_subband_quantization();
extern void        II_encode_bit_alloc();
extern void        I_encode_bit_alloc();
extern void        I_sample_encoding();
extern void        II_sample_encoding();
extern void        encode_CRC();
#endif

/* The following functions are in the file "tonal.c" */

#ifdef     PROTO_ARGS
extern void        read_cbound(int, int);
extern void        read_freq_band(g_ptr*, int, int);
extern void        make_map(mask[HAN_SIZE], g_thres*);
extern double      add_db(double, double);
```

```
extern void        II_f_f_t(double[FFT_SIZE], mask[HAN_SIZE]);
extern void        II_hann_win(double[FFT_SIZE]);
extern void        II_pick_max(mask[HAN_SIZE], double[SBLIMIT]);
extern void        II_tonal_label(mask[HAN_SIZE], int*);
extern void        noise_label(mask*, int*, g_thres*);
extern void        subsampling(mask[HAN_SIZE], g_thres*, int*, int*);
extern void        threshold(mask[HAN_SIZE], g_thres*, int*, int*, int);
extern void        II_minimum_mask(g_thres*, double[SBLIMIT], int);
extern void        II_smr(double[SBLIMIT], double[SBLIMIT], double[SBLIMIT],
                           int);
extern void        II_Psycho_One(short[2][1152], double[2][SBLIMIT],
                           double[2][SBLIMIT], frame_params*);
extern void        I_f_f_t(double[FFT_SIZE/2], mask[HAN_SIZE/2]);
extern void        I_hann_win(double[FFT_SIZE/2]);
extern void        I_pick_max(mask[HAN_SIZE/2], double[SBLIMIT]);
extern void        I_tonal_label(mask[HAN_SIZE/2], int*);
extern void        I_minimum_mask(g_thres*, double[SBLIMIT]);
extern void        I_smr(double[SBLIMIT], double[SBLIMIT], double[SBLIMIT]);
extern void        I_Psycho_One(short[2][1152], double[2][SBLIMIT],
                           double[2][SBLIMIT], frame_params*);
#else
extern void        read_cbound();
extern void        read_freq_band();
extern void        make_map();
extern double      add_db();
extern void        II_f_f_t();
extern void        II_hann_win();
extern void        II_pick_max();
extern void        II_tonal_label();
extern void        noise_label();
extern void        subsampling();
extern void        threshold();
extern void        II_minimum_mask();
extern void        II_smr();
extern void        II_Psycho_One();
extern void        I_f_f_t();
extern void        I_hann_win();
extern void        I_pick_max();
extern void        I_tonal_label();
extern void        I_minimum_mask();
extern void        I_smr();
extern void        I_Psycho_One();
#endif

/* The following functions are in the file "psy.c" */

#ifdef     PROTO_ARGS
extern void        psycho_anal(short int*, short int[1056], int, int,
                           FLOAT[32], double);
#else
extern void        psycho_anal();
#endif

/* The following functions are in the file "subs.c" */

#ifdef     PROTO_ARGS
extern void        fft(FLOAT[BLKSIZE], FLOAT[BLKSIZE], FLOAT[BLKSIZE],
                           FLOAT[BLKSIZE], int );
#else
extern void        fft();
#endif
```

C.3.7 huffman.c

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
huffman.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
```

```
 *   changes made since last update:                                    *
 *   date   programmers                comment                          *
 *27.2.92   F.O.Witte                  (ITT Intermetall)                *
 *                     email: otto.witte@itt-sc.de    *
 *                     tel:   ++49 (761)517-125           *
 *                     fax:   ++49 (761)517-880           *
 *12.6.92   J. Pineda                  Added sign bit to decoder.       *
 * 08/24/93 M. Iwadare                 Changed for 1 pass decoding.     *
 *----------------------------------------------------------------------*
 *  7/14/94 Juergen Koller      Bug fixes in Layer III code             *
 ***********************************************************************/

#include "common.h"
#include "huffman.h"

HUFFBITS dmask = 1 << (sizeof(HUFFBITS)*8-1);
unsigned int hs = sizeof(HUFFBITS)*8;

struct huffcodetab ht[HTN];  /* array of all huffcodtable headers   */
                /* 0..31 Huffman code table 0..31   */
                /* 32,33 count1-tables              */

/* read the huffman encode table */
int read_huffcodetab(fi)
FILE *fi;
{

  char line[100],command[40],huffdata[40];
  unsigned int t,i,j,k,nn,x,y,n=0;
  unsigned int xl, yl, len;
  HUFFBITS h;
  int   hsize;

  hsize = sizeof(HUFFBITS)*8;
  do {
      fgets(line,99,fi);
  } while ((line[0] == '#') || (line[0] < ' ') );

  do {
    while ((line[0]=='#') || (line[0] < ' ')) {
      fgets(line,99,fi);
    }

    sscanf(line,"%s %s %u %u %u",command,ht[n].tablename,
                     &xl,&yl,&ht[n].linbits);
    if (strcmp(command,".end")==0)
      return n;
    else if (strcmp(command,".table")!=0) {
      fprintf(stderr,"huffman table %u data corrupted\n",n);
      return -1;
    }
    ht[n].linmax = (1<<ht[n].linbits)-1;

    sscanf(ht[n].tablename,"%u",&nn);
    if (nn != n) {
      fprintf(stderr,"wrong table number %u\n",n);
      return(-2);
    }

    ht[n].xlen = xl;
    ht[n].ylen = yl;

    do {
      fgets(line,99,fi);
    } while ((line[0] == '#') || (line[0] < ' '));

    sscanf(line,"%s %u",command,&t);
    if (strcmp(command,".reference")==0) {
      ht[n].ref   = t;
      ht[n].table = ht[t].table;
      ht[n].hlen  = ht[t].hlen;
      if ( (xl != ht[t].xlen) ||
           (yl != ht[t].ylen)  ) {
        fprintf(stderr,"wrong table %u reference\n",n);
        return (-3);
      };
```

```
      do {
        fgets(line,99,fi);
      } while ((line[0] == '#') || (line[0] < ' ') );
    }
    else {
      ht[n].ref  = -1;
      ht[n].table=(HUFFBITS *) calloc(xl*yl,sizeof(HUFFBITS));
      if (ht[n].table == NULL) {
         fprintf(stderr,"unsufficient heap error\n");
         return (-4);
      }
      ht[n].hlen=(unsigned char *) calloc(xl*yl,sizeof(unsigned char));
      if (ht[n].hlen == NULL) {
         fprintf(stderr,"unsufficient heap error\n");
         return (-4);
      }
      for (i=0; i<xl; i++) {
        for (j=0;j<yl; j++) {
      if (xl>1)
            sscanf(line,"%u %u %u %s",&x, &y, &len,huffdata);
      else
            sscanf(line,"%u %u %s",&x,&len,huffdata);
          h=0;k=0;
      while (huffdata[k]) {
            h <<= 1;
            if (huffdata[k] == '1')
              h++;
            else if (huffdata[k] != '0'){
              fprintf(stderr,"huffman-table %u bit error\n",n);
              return (-5);
            };
            k++;
          };
          if (k != len) {
           fprintf(stderr,
              "warning: wrong codelen in table %u, pos [%2u][%2u]\n",
           n,i,j);
          };
          ht[n].table[i*xl+j] = h;
          ht[n].hlen[i*xl+j] = (unsigned char) len;
      do {
            fgets(line,99,fi);
          } while ((line[0] == '#') || (line[0] < ' '));
        }
      }
    }
    n++;
  } while (1);
}

/* read the huffman decoder table */
int read_decoder_table(fi)
FILE *fi;
{
  int n,i,nn,t;
  unsigned int v0,v1;
  char command[100],line[100];
  for (n=0;n<HTN;n++) {
    /* .table number treelen xlen ylen linbits */
    do {
      fgets(line,99,fi);
    } while ((line[0] == '#') || (line[0] < ' '));

    sscanf(line,"%s %s %u %u %u %u",command,ht[n].tablename,
           &ht[n].treelen, &ht[n].xlen, &ht[n].ylen, &ht[n].linbits);
    if (strcmp(command,".end")==0)
      return n;
    else if (strcmp(command,".table")!=0) {
      fprintf(stderr,"huffman table %u data corrupted\n",n);
      return -1;
    }
    ht[n].linmax = (1<<ht[n].linbits)-1;

    sscanf(ht[n].tablename,"%u",&nn);
    if (nn != n) {
      fprintf(stderr,"wrong table number %u\n",n);
```

```
      return(-2);
    }
    do {
      fgets(line,99,fi);
    } while ((line[0] == '#') || (line[0] < ' '));

    sscanf(line,"%s %u",command,&t);
    if (strcmp(command,".reference")==0) {
      ht[n].ref   = t;
      ht[n].val   = ht[t].val;
      ht[n].treelen  = ht[t].treelen;
      if ( (ht[n].xlen != ht[t].xlen) ||
           (ht[n].ylen != ht[t].ylen)  ) {
        fprintf(stderr,"wrong table %u reference\n",n);
        return (-3);
      };
      while ((line[0] == '#') || (line[0] < ' ') ) {
        fgets(line,99,fi);
      }
    }
    else if (strcmp(command,".treedata")==0) {
      ht[n].ref  = -1;
      ht[n].val = (unsigned char (*)[2])
        calloc(2*(ht[n].treelen),sizeof(unsigned char));
      if (ht[n].val == NULL) {
    fprintf(stderr, "heaperror at table %d\n",n);
    exit (-10);
      }
      for (i=0;i<ht[n].treelen; i++) {
        fscanf(fi,"%x %x",&v0, &v1);
        ht[n].val[i][0]=(unsigned char)v0;
        ht[n].val[i][1]=(unsigned char)v1;
      }
      fgets(line,99,fi); /* read the rest of the line */
    }
    else {
      fprintf(stderr,"huffman decodertable error at table %d\n",n);
    }
  }
  return n;
}


/* do the huffman coding,  */
/* note! for counta,countb - the 4 bit value is passed in y, set x to 0 */
/* return value: 0-no error, 1 decode error                        */
void huffman_coder( x, y, h, bs)
unsigned int x;  /* x-value */
unsigned int y;  /* y-value */
struct huffcodetab *h;   /* pointer to huffman code record  */
Bit_stream_struc *bs;    /* pointer to open write bitstream        */
{
  HUFFBITS huffbits; /* data left aligned */
  HUFFBITS linbitsX;
  HUFFBITS linbitsY;
  unsigned int len;
  unsigned int xl1 = h->xlen-1;
  unsigned int yl1 = h->ylen-1;
  linbitsX = 0;
  linbitsY = 0;
  if (h->table == NULL) return;
  if (((x < xl1) || (xl1==0)) && (y < yl1)) {
    huffbits = h->table[x*(h->xlen)+y];
    len = h->hlen[x*(h->xlen)+y];
    putbits(bs,huffbits,len);
    return;
  }
  else if (x >= xl1) {
    linbitsX = x-xl1;
    if (linbitsX > h->linmax) {
      fprintf(stderr,"warning: Huffman X table overflow\n");
      linbitsX= h->linmax;
    };
    if (y >= yl1) {
      huffbits = h->table[(h->ylen)*(h->xlen)-1];
      len = h->hlen[(h->ylen)*(h->xlen)-1];
```

```
      putbits(bs,huffbits,len);
      linbitsY = y-yl1;
      if (linbitsY > h->linmax) {
        fprintf(stderr,"warning: Huffman Y table overflow\n");
        linbitsY = h->linmax;
      };
      if (h->linbits) {
        putbits(bs,linbitsX,h->linbits);
        putbits(bs,linbitsY,h->linbits);
      }
    }
    else { /* x>= h->xlen, y<h->ylen */
      huffbits = h->table[(h->ylen)*xl1+y];
      len = h->hlen[(h->ylen)*xl1+y];
      putbits(bs,huffbits,len);
      if (h->linbits) {
        putbits(bs,linbitsX,h->linbits);
      }
    }
  }
  else  { /* ((x < h->xlen) && (y>=h->ylen)) */
    huffbits = h->table[(h->ylen)*x+yl1];
    len = h->hlen[(h->ylen)*x+yl1];
    putbits(bs,huffbits,len);
    linbitsY = y-yl1;
    if (linbitsY > h->linmax) {
      fprintf(stderr,"warning: Huffman Y table overflow\n");
      linbitsY = h->linmax;
    };
    if (h->linbits) {
       putbits(bs,linbitsY,h->linbits);
    }
  }
}

/* do the huffman-decoding                          */
/* note! for counta,countb -the 4 bit value is returned in y, discard x */
int huffman_decoder(h, x, y, v, w)
struct huffcodetab *h;   /* pointer to huffman code record  */
/* unsigned */ int *x;   /* returns decoded x value         */
/* unsigned */ int *y;   /* returns decoded y value         */
int *v;
int *w;
{
  HUFFBITS level;
  int point = 0;
  int error = 1;
  level     = dmask;
  if (h->val == NULL) return 2;

  /* table 0 needs no bits */
  if ( h->treelen == 0)
  {  *x = *y = 0;
     return 0;
  }


  /* Lookup in Huffman table. */

  do {
    if (h->val[point][0]==0) {   /*end of tree*/
      *x = h->val[point][1] >> 4;
      *y = h->val[point][1] & 0xf;

      error = 0;
      break;
    }
    if (hget1bit()) {
      while (h->val[point][1] >= MXOFF) point += h->val[point][1];
      point += h->val[point][1];
    }
    else {
      while (h->val[point][0] >= MXOFF) point += h->val[point][0];
      point += h->val[point][0];
    }
    level >>= 1;
```

```
  } while (level  || (point < ht->treelen) );

  /* Check for error. */

  if (error) { /* set x and y to a medium value as a simple concealment */
    printf("Illegal Huffman code in data.\n");
    *x = (h->xlen-1 << 1);
    *y = (h->ylen-1 << 1);
  }

  /* Process sign encodings for quadruples tables. */

  if (h->tablename[0] == '3'
      && (h->tablename[1] == '2' || h->tablename[1] == '3')) {
     *v = (*y>>3) & 1;
     *w = (*y>>2) & 1;
     *x = (*y>>1) & 1;
     *y = *y & 1;

     /* v, w, x and y are reversed in the bitstream.
        switch them around to make test bistream work. */

/*   {int i=*v; *v=*y; *y=i; i=*w; *w=*x; *x=i;}  MI */

     if (*v)
        if (hget1bit() == 1) *v = -*v;
     if (*w)
        if (hget1bit() == 1) *w = -*w;
     if (*x)
        if (hget1bit() == 1) *x = -*x;
     if (*y)
        if (hget1bit() == 1) *y = -*y;
     }

  /* Process sign and escape encodings for dual tables. */

  else {

      /* x and y are reversed in the test bitstream.
         Reverse x and y here to make test bitstream work. */

/*    removed 11/11/92 -ag
        {int i=*x; *x=*y; *y=i;}
*/
     if (h->linbits)
       if ((h->xlen-1) == *x)
         *x += hgetbits(h->linbits);
     if (*x)
        if (hget1bit() == 1) *x = -*x;
     if (h->linbits)
       if ((h->ylen-1) == *y)
         *y += hgetbits(h->linbits);
     if (*y)
        if (hget1bit() == 1) *y = -*y;
     }

  return error;
}
```

C.3.8 huffman.h

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
huffman.h
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers                comment                        *
 *  27.2.92 F.O.Witte (ITT Intermetall)                           *
```

```
 *  8/24/93 M. Iwadare          Changed for 1 pass decoding.          *
 *  7/14/94 J. Koller        useless 'typedef' before huffcodetab  *
 *            removed                   *
 *********************************************************************/

#define HUFFBITS unsigned long int
#define HTN  34
#define MXOFF   250

struct huffcodetab {
  char tablename[3]; /*string, containing table_description */
  unsigned int xlen;     /*max. x-index+                      */
  unsigned int ylen; /*max. y-index+               */
  unsigned int linbits; /*number of linbits             */
  unsigned int linmax;   /*max number to be stored in linbits       */
  int ref;       /*a positive value indicates a reference*/
  HUFFBITS *table;   /*pointer to array[xlen][ylen]          */
  unsigned char *hlen;   /*pointer to array[xlen][ylen]             */
  unsigned char(*val)[2];/*decoder tree                         */
  unsigned int treelen;  /*length of decoder tree            */
};

extern struct huffcodetab ht[HTN];/* global memory block            */
                /* array of all huffcodtable headers       */
                /* 0..31 Huffman code table 0..31   */
                /* 32,33 count1-tables              */
#ifdef PROTO_ARGS

extern int read_huffcodetab(FILE *);
extern int read_decoder_table(FILE *);

extern void huffman_coder(unsigned int, unsigned int,
              struct huffcodetab *, Bit_stream_struc *);

extern int huffman_decoder(struct huffcodetab *,
               /* unsigned */ int *, /* unsigned */ int*, int*, int*);

#else

extern int read_huffcodetab();
extern int read_decoder_table();
extern void huffman_coder();
extern int huffman_decoder();

#endif
```

C.3.9 musicin.c

```
/**********************************************************************
Copyright (c) 1991 ISO/IEC JTC1 SC29 WG1, All Rights Reserved
musicin.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 3/01/91  Douglas Wong,       start of version 1.1 records          *
 *          Davis Pan                                                 *
 * 3/06/91  Douglas Wong,       rename: setup.h to endef.h            *
 *                              removed extraneous variables          *
 * 3/21/91  J.Georges Fritsch   introduction of the bit-stream        *
 *                              package. This package allows you      *
 *                              to generate the bit-stream in a       *
 *                              binary or ascii format                *
 * 3/31/91  Bill Aspromonte     replaced the read of the SB matrix    *
 *                              by an "code generated" one            *
 * 5/10/91  W. Joseph Carter    Ported to Macintosh and Unix.         *
 *                              Incorporated Jean-Georges Fritsch's   *
 *                              "bitstream.c" package.                *
```

```
 *                               Modified to strictly adhere to       *
 *                               encoded bitstream specs, including   *
 *                               "Berlin changes".                    *
 *                               Modified user interface dialog & code *
 *                               to accept any input & output         *
 *                               filenames desired.  Also added       *
 *                               de-emphasis prompt and final bail-out *
 *                               opportunity before encoding.         *
 *                               Added AIFF PCM sound file reading    *
 *                               capability.                          *
 *                               Modified PCM sound file handling to  *
 *                               process all incoming samples and fill *
 *                               out last encoded frame with zeros    *
 *                               (silence) if needed.                 *
 *                               Located and fixed numerous software  *
 *                               bugs and table data errors.          *
 * 27jun91  dpwe (Aware Inc)     Used new frame_params struct.        *
 *                               Clear all automatic arrays.          *
 *                               Changed some variable names,         *
 *                               simplified some code.                *
 *                               Track number of bits actually sent.  *
 *                               Fixed padding slot, stereo bitrate   *
 *                               Added joint-stereo : scales L+R.     *
 * 6/12/91  Earle Jennings       added fix for MS_DOS in obtain_param *
 * 6/13/91  Earle Jennings       added stack length adjustment before *
 *                               main for MS_DOS                      *
 * 7/10/91  Earle Jennings       conversion of all float to FLOAT     *
 *                               port to MsDos from MacIntosh completed*
 * 8/ 8/91  Jens Spille          Change for MS-C6.00                  *
 * 8/22/91  Jens Spille          new obtain_parameters()              *
 *10/ 1/91  S.I. Sudharsanan,    Ported to IBM AIX platform.          *
 *          Don H. Lee,                                               *
 *          Peter W. Farrett                                          *
 *10/ 3/91  Don H. Lee           implemented CRC-16 error protection  *
 *                               newly introduced functions are       *
 *                               I_CRC_calc, II_CRC_calc and encode_CRC*
 *                               Additions and revisions are marked   *
 *                               with "dhl" for clarity               *
 *11/11/91 Katherine Wang        Documentation of code.               *
 *                                 (variables in documentation are    *
 *                                 surround by the # symbol, and an '*'*
 *                                 denotes layer I or II versions)    *
 * 2/11/92  W. Joseph Carter     Ported new code to Macintosh.  Most  *
 *                               important fixes involved changing    *
 *                               16-bit ints to long or unsigned in   *
 *                               bit alloc routines for quant of 65535 *
 *                               and passing proper function args.    *
 *                               Removed "Other Joint Stereo" option  *
 *                               and made bitrate be total channel    *
 *                               bitrate, irrespective of the mode.   *
 *                               Fixed many small bugs & reorganized. *
 * 2/25/92  Masahiro Iwadare     made code cleaner and more consistent *
 * 8/07/92  Mike Coleman         make exit() codes return error status *
 *                               made slight changes for portability  *
 *19 aug 92 Soren H. Nielsen     Changed MS-DOS file name extensions. *
 * 8/25/92  Shaun Astarabadi     Replaced rint() function with explicit*
 *                               rounding for portability with MSDOS. *
 * 9/22/92  jddevine@aware.com   Fixed _scale_factor_calc() calls.    *
 *10/19/92  Masahiro Iwadare     added info->mode and info->mode_ext  *
 *                               updates for AIFF format files        *
 * 3/10/93  Kevin Peterson       In parse_args, only set non default  *
 *                               bit rate if specified in arg list.   *
 *                               Use return value from aiff_read_hdrs *
 *                               to fseek to start of sound data      *
 * 7/26/93  Davis Pan            fixed bug in printing info->mode_ext *
 *                               value for joint stereo condition     *
 * 8/27/93 Seymour Shlien,       Fixes in Unix and MSDOS ports,       *
 *         Daniel Lauzon, and                                         *
 *         Bill Truerniet                                             *
 **********************************************************************/

#ifdef MS_DOS
#include <dos.h>
#endif
#include "common.h"
#include "encoder.h"
```

```
/* Global variable definitions for "musicin.c" */

FILE               *musicin;
Bit_stream_struc   bs;
char               *programName;

/* Implementations */

/************************************************************************
/*
/* obtain_parameters
/*
/* PURPOSE:  Prompts for and reads user input for encoding parameters
/*
/* SEMANTICS:  The parameters read are:
/* - input and output filenames
/* - sampling frequency (if AIFF file, will read from the AIFF file header)
/* - layer number
/* - mode (stereo, joint stereo, dual channel or mono)
/* - psychoacoustic model (I or II)
/* - total bitrate, irrespective of the mode
/* - de-emphasis, error protection, copyright and original or copy flags
/*
/************************************************************************/

void
obtain_parameters(fr_ps,psy,num_samples,original_file_name,encoded_file_name)
frame_params    *fr_ps;
int             *psy;
unsigned long   *num_samples;
char            original_file_name[MAX_NAME_SIZE];
char            encoded_file_name[MAX_NAME_SIZE];
{
    int j;
    long int freq;
    int model, brt;
    char t[50];
    IFF_AIFF pcm_aiff_data;
    layer *info = fr_ps->header;
    long soundPosition;

    do  {
       printf("Enter PCM input file name <required>: ");
       gets(original_file_name);
       if (original_file_name[0] == NULL_CHAR)
       printf("PCM input file name is required.\n");
    } while (original_file_name[0] == NULL_CHAR);
    printf(">>> PCM input file name is: %s\n", original_file_name);

    if ((musicin = fopen(original_file_name, "rb")) == NULL) {
       printf("Could not find \"%s\".\n", original_file_name);
       exit(1);
    }

#ifdef  MS_DOS
    printf("Enter MPEG encoded output file name <%s>: ",
           new_ext(original_file_name, DFLT_EXT)); /* 92-08-19 shn */
#else
    printf("Enter MPEG encoded output file name <%s%s>: ",
           original_file_name, DFLT_EXT);
#endif

    gets(encoded_file_name);
    if (encoded_file_name[0] == NULL_CHAR) {
#ifdef  MS_DOS
        /* replace old extension with new one, 92-08-19 shn */
        strcpy(encoded_file_name,new_ext(original_file_name, DFLT_EXT));
#else
        strcat(strcpy(encoded_file_name, original_file_name), DFLT_EXT);
#endif
    }


    printf(">>> MPEG encoded output file name is: %s\n", encoded_file_name);
```

```
    open_bit_stream_w(&bs, encoded_file_name, BUFFER_SIZE);

    if ((soundPosition = aiff_read_headers(musicin, &pcm_aiff_data)) != -1) {

       printf(">>> Using Audio IFF sound file headers\n");

       aiff_check(original_file_name, &pcm_aiff_data);

       if (fseek(musicin, soundPosition, SEEK_SET) != 0) {
          printf("Could not seek to PCM sound data in \"%s\".\n",
                 original_file_name);
          exit(1);
       }

       info->sampling_frequency = SmpFrqIndex((long)pcm_aiff_data.sampleRate);
       printf(">>> %.f Hz sampling frequency selected\n",
              pcm_aiff_data.sampleRate);

       /* Determine number of samples in sound file */
#ifndef MS_DOS
       *num_samples = pcm_aiff_data.numChannels *
                      pcm_aiff_data.numSampleFrames;
#else
       *num_samples = (long)(pcm_aiff_data.numChannels) *
                      (long)(pcm_aiff_data.numSampleFrames);
#endif

    }
    else {    /* Not using Audio IFF sound file headers. */

       printf("What is the sampling frequency? <44100>[Hz]: ");
       gets(t);
       freq = atol(t);
       switch (freq) {
          case 48000 : info->sampling_frequency = 1;
              printf(">>> %ld Hz sampling freq selected\n", freq);
              break;
          case 44100 : info->sampling_frequency = 0;
              printf(">>> %ld Hz sampling freq selected\n", freq);
              break;
          case 32000 : info->sampling_frequency = 2;
              printf(">>> %ld Hz sampling freq selected\n", freq);
              break;
          default:    info->sampling_frequency = 0;
              printf(">>> Default 44.1 kHz samp freq selected\n");
       }

       if (fseek(musicin, 0, SEEK_SET) != 0) {
          printf("Could not seek to PCM sound data in \"%s\".\n",
                  original_file_name);
          exit(1);
       }

       /* Declare sound file to have "infinite" number of samples. */
       *num_samples = MAX_U_32_NUM;

    }

    printf("Which layer do you want to use?\n");
    printf("Available: Layer (1), Layer (<2>): ");
    gets(t);
    switch(*t){
       case '1': info->lay = 1; printf(">>> Using Layer %s\n",t); break;
       case '2': info->lay = 2; printf(">>> Using Layer %s\n",t); break;
       default:  info->lay = 2; printf(">>> Using default Layer 2\n"); break;
    }

    printf("Which mode do you want?\n");
    printf("Available: (<s>)tereo, (j)oint stereo, ");
    printf("(d)ual channel, s(i)ngle Channel: ");
    gets(t);
    switch(*t){
       case 's':
       case 'S':
          info->mode = MPG_MD_STEREO; info->mode_ext = 0;
          printf(">>> Using mode %s\n",t);
```

```
         break;
      case 'j':
      case 'J':
         info->mode = MPG_MD_JOINT_STEREO;
         printf(">>> Using mode %s\n",t);
         break;
      case 'd':
      case 'D':
         info->mode = MPG_MD_DUAL_CHANNEL; info->mode_ext = 0;
         printf(">>> Using mode %s\n",t);
         break;
      case 'i':
      case 'I':
         info->mode = MPG_MD_MONO; info->mode_ext = 0;
         printf(">>> Using mode %s\n",t);
         break;
      default:
         info->mode = MPG_MD_STEREO; info->mode_ext = 0;
         printf(">>> Using default stereo mode\n");
         break;
   }

   printf("Which psychoacoustic model do you want to use? <2>: ");
   gets(t);
   model = atoi(t);
   if (model > 2 || model < 1) {
      printf(">>> Default model 2 selected\n");
      *psy = 2;
   }
   else {
      *psy = model;
      printf(">>> Using psychoacoustic model %d\n", model);
   }

   printf("What is the total bitrate? <%u>[kbps]: ", DFLT_BRT);
   gets(t);
   brt = atoi(t);
   if (brt == 0) brt = -10;
   j=0;
   while (j<15) {
      if (bitrate[info->lay-1][j] == brt) break;
      j++;
   }
   if (j==15) {
      printf(">>> Using default %u kbps\n", DFLT_BRT);
      for (j=0;j<15;j++)
         if (bitrate[info->lay-1][j] == DFLT_BRT) {
            info->bitrate_index = j;
            break;
         }
   }
   else{
      info->bitrate_index = j;
      printf(">>> Bitrate = %d kbps\n", bitrate[info->lay-1][j]);
   }

   printf("What type of de-emphasis should the decoder use?\n");
   printf("Available: (<n>)one, (5)0/15 microseconds, (c)citt j.17: ");
   gets(t);
   if (*t != 'n' && *t != '5' && *t != 'c') {
      printf(">>> Using default no de-emphasis\n");
      info->emphasis = 0;
   }
   else {
      if (*t == 'n')      info->emphasis = 0;
      else if (*t == '5') info->emphasis = 1;
      else if (*t == 'c') info->emphasis = 3;
      printf(">>> Using de-emphasis %s\n",t);
   }

/*  Start 2. Part changes for CD Ver 3.2; jsp; 22-Aug-1991 */

   printf("Do you want to set the private bit? (y/<n>): ");
   gets(t);
   if (*t == 'y' || *t == 'Y') info->extension = 1;
   else                        info->extension = 0;
```

```
    if(info->extension) printf(">>> Private bit set\n");
    else                printf(">>> Private bit not set\n");

/*  End changes for CD Ver 3.2; jsp; 22-Aug-1991 */

    printf("Do you want error protection? (y/<n>): ");
    gets(t);
    if (*t == 'y' || *t == 'Y') info->error_protection = TRUE;
    else                        info->error_protection = FALSE;
    if(info->error_protection) printf(">>> Error protection used\n");
    else printf(">>> Error protection not used\n");

    printf("Is the material copyrighted? (y/<n>): ");
    gets(t);
    if (*t == 'y' || *t == 'Y') info->copyright = 1;
    else                        info->copyright = 0;
    if(info->copyright) printf(">>> Copyrighted material\n");
    else                printf(">>> Material not copyrighted\n");

    printf("Is this the original? (y/<n>): ");
    gets(t);
    if (*t == 'y' || *t == 'Y') info->original = 1;
    else                        info->original = 0;
    if(info->original) printf(">>> Original material\n");
    else               printf(">>> Material not original\n");

    printf("Do you wish to exit (last chance before encoding)? (y/<n>): ");
    gets(t);
    if (*t == 'y' || *t == 'Y') exit(0);
}

/************************************************************************
/*
/* parse_args
/*
/* PURPOSE:  Sets encoding parameters to the specifications of the
/* command line.  Default settings are used for parameters
/* not specified in the command line.
/*
/* SEMANTICS:  The command line is parsed according to the following
/* syntax:
/*
/* -l  is followed by the layer number
/* -m  is followed by the mode
/* -p  is followed by the psychoacoustic model number
/* -s  is followed by the sampling rate
/* -b  is followed by the total bitrate, irrespective of the mode
/* -d  is followed by the emphasis flag
/* -c  is followed by the copyright/no_copyright flag
/* -o  is followed by the original/not_original flag
/* -e  is followed by the error_protection on/off flag
/*
/* If the input file is in AIFF format, the sampling frequency is read
/* from the AIFF header.
/*
/* The input and output filenames are read into #inpath# and #outpath#.
/*
/************************************************************************/

void
parse_args(argc, argv, fr_ps, psy, num_samples, inPath, outPath)
int     argc;
char    **argv;
frame_params  *fr_ps;
int     *psy;
unsigned long *num_samples;
char    inPath[MAX_NAME_SIZE];
char    outPath[MAX_NAME_SIZE];
{
   FLOAT srate;
   int   brate;
   layer *info = fr_ps->header;
   int   err = 0, i = 0;
   IFF_AIFF pcm_aiff_data;
   long samplerate;
   long soundPosition;
```

```
/* preset defaults */
inPath[0] = '\0';   outPath[0] = '\0';
info->lay = DFLT_LAY;
switch(DFLT_MOD) {
   case 's': info->mode = MPG_MD_STEREO; info->mode_ext = 0; break;
   case 'd': info->mode = MPG_MD_DUAL_CHANNEL; info->mode_ext=0; break;
   case 'j': info->mode = MPG_MD_JOINT_STEREO; break;
   case 'm': info->mode = MPG_MD_MONO; info->mode_ext = 0; break;
   default:
      fprintf(stderr, "%s: Bad mode dflt %c\n", programName, DFLT_MOD);
      abort();
}
*psy = DFLT_PSY;
if((info->sampling_frequency = SmpFrqIndex((long)(1000*DFLT_SFQ))) < 0) {
   fprintf(stderr, "%s: bad sfrq default %.2f\n", programName, DFLT_SFQ);
   abort();
}
if((info->bitrate_index = BitrateIndex(info->lay, DFLT_BRT)) < 0) {
   fprintf(stderr, "%s: bad default bitrate %u\n", programName, DFLT_BRT);
   abort();
}
switch(DFLT_EMP) {
   case 'n': info->emphasis = 0; break;
   case '5': info->emphasis = 1; break;
   case 'c': info->emphasis = 3; break;
   default:
      fprintf(stderr, "%s: Bad emph dflt %c\n", programName, DFLT_EMP);
      abort();
}
info->copyright = 0; info->original = 0; info->error_protection = FALSE;

/* process args */
while(++i<argc && err == 0) {
   char c, *token, *arg, *nextArg;
   int  argUsed;

   token = argv[i];
   if(*token++ == '-') {
      if(i+1 < argc) nextArg = argv[i+1];
      else           nextArg = "";
      argUsed = 0;
      while(c = *token++) {
         if(*token /* NumericQ(token) */) arg = token;
         else                             arg = nextArg;
         switch(c) {
            case 'l':        info->lay = atoi(arg); argUsed = 1;
               if(info->lay<1 || info->lay>2) {
                  fprintf(stderr,"%s: -l layer must be 1 or 2, not %s\n",
                       programName, arg);
                  err = 1;
               }
               break;
            case 'm':        argUsed = 1;
               if (*arg == 's')
                 { info->mode = MPG_MD_STEREO; info->mode_ext = 0; }
               else if (*arg == 'd')
                 { info->mode = MPG_MD_DUAL_CHANNEL; info->mode_ext=0; }
               else if (*arg == 'j')
                 { info->mode = MPG_MD_JOINT_STEREO; }
               else if (*arg == 'm')
                 { info->mode = MPG_MD_MONO; info->mode_ext = 0; }
               else {
                 fprintf(stderr,"%s: -m mode must be s/d/j/m not %s\n",
                         programName, arg);
                 err = 1;
               }
               break;
            case 'p':        *psy = atoi(arg); argUsed = 1;
               if(*psy<1 || *psy>2) {
                  fprintf(stderr,"%s: -p model must be 1 or 2, not %s\n",
                          programName, arg);
                  err = 1;
               }
               break;
```

```
            case 's':
               argUsed = 1;
               srate = atof( arg );
               /* samplerate = rint( 1000.0 * srate ); $A  */
               samplerate = (long) (( 1000.0 * srate ) + 0.5);
               if( (info->sampling_frequency = SmpFrqIndex((long) samplerate)) < 0 )
                   err = 1;
               break;

            case 'b':
        argUsed = 1;
        brate = atoi(arg);
        if( (info->bitrate_index = BitrateIndex(info->lay, brate)) < 0)
           err=1;
       break;
            case 'd':        argUsed = 1;
               if (*arg == 'n')                    info->emphasis = 0;
               else if (*arg == '5')               info->emphasis = 1;
               else if (*arg == 'c')               info->emphasis = 3;
               else {
                  fprintf(stderr,"%s: -d emp must be n/5/c not %s\n",
                          programName, arg);
                  err = 1;
               }
               break;
             case 'c':       info->copyright = 1; break;
             case 'o':       info->original  = 1; break;
             case 'e':       info->error_protection = TRUE; break;
             default:        fprintf(stderr,"%s: unrec option %c\n",
                                     programName, c);
                             err = 1; break;
         }
         if(argUsed) {
            if(arg == token)    token = "";   /* no more from token */
            else                ++i;          /* skip arg we used */
            arg = ""; argUsed = 0;
         }
      }
   }
   else {
      if(inPath[0] == '\0')       strcpy(inPath, argv[i]);
      else if(outPath[0] == '\0') strcpy(outPath, argv[i]);
      else {
         fprintf(stderr,"%s: excess arg %s\n", programName, argv[i]);
         err = 1;
      }
   }
}

if(err || inPath[0] == '\0') usage();  /* never returns */

if(outPath[0] == '\0') {
   strcpy(outPath, inPath);
   strcat(outPath, DFLT_EXT);
}

if ((musicin = fopen(inPath, "rb")) == NULL) {
   printf("Could not find \"%s\".\n", inPath);
   exit(1);
}

open_bit_stream_w(&bs, outPath, BUFFER_SIZE);

if ((soundPosition = aiff_read_headers(musicin, &pcm_aiff_data)) != -1) {

   printf(">>> Using Audio IFF sound file headers\n");

   aiff_check(inPath, &pcm_aiff_data);

   if (fseek(musicin, soundPosition, SEEK_SET) != 0) {
      printf("Could not seek to PCM sound data in \"%s\".\n", inPath);
      exit(1);
   }

   info->sampling_frequency = SmpFrqIndex((long)pcm_aiff_data.sampleRate);
   printf(">>> %.f Hz sampling frequency selected\n",
```

```
             pcm_aiff_data.sampleRate);

      /* Determine number of samples in sound file */
#ifndef MS_DOS
      *num_samples = pcm_aiff_data.numChannels *
                     pcm_aiff_data.numSampleFrames;
#else
      *num_samples = (long)(pcm_aiff_data.numChannels) *
                     (long)(pcm_aiff_data.numSampleFrames);
#endif
      if ( pcm_aiff_data.numChannels == 1 ) {
        info->mode = MPG_MD_MONO;
        info->mode_ext = 0;
      }
   }
   else {    /* Not using Audio IFF sound file headers. */

      if (fseek(musicin, 0, SEEK_SET) != 0) {
         printf("Could not seek to PCM sound data in \"%s\".\n", inPath);
         exit(1);
      }

      /* Declare sound file to have "infinite" number of samples. */
      *num_samples = MAX_U_32_NUM;

   }

}

/************************************************************************
/*
/* print_config
/*
/* PURPOSE:  Prints the encoding parameters used
/*
/************************************************************************/

void
print_config(fr_ps, psy, num_samples, inPath, outPath)
frame_params *fr_ps;
int     *psy;
unsigned long *num_samples;
char    inPath[MAX_NAME_SIZE];
char    outPath[MAX_NAME_SIZE];
{
 layer *info = fr_ps->header;

   printf("Encoding configuration:\n");
   if(info->mode != MPG_MD_JOINT_STEREO)
      printf("Layer=%s   mode=%s   extn=%d   psy model=%d\n",
             layer_names[info->lay-1], mode_names[info->mode],
             info->mode_ext, *psy);
   else printf("Layer=%s   mode=%s   extn=data dependant   psy model=%d\n",
               layer_names[info->lay-1], mode_names[info->mode], *psy);
   printf("samp frq=%.1f kHz   total bitrate=%d kbps\n",
          s_freq[info->sampling_frequency],
          bitrate[info->lay-1][info->bitrate_index]);
   printf("de-emph=%d   c/right=%d   orig=%d   errprot=%d\n",
          info->emphasis, info->copyright, info->original,
          info->error_protection);
   printf("input file: '%s'   output file: '%s'\n", inPath, outPath);
}

/************************************************************************
/*
/* main
/*
/* PURPOSE:  MPEG I Encoder supporting layers 1 and 2, and
/* psychoacoustic models 1 (MUSICAM) and 2 (AT&T)
/*
/* SEMANTICS:  One overlapping frame of audio of up to 2 channels are
/* processed at a time in the following order:
/* (associated routines are in parentheses)
/*
/* 1.  Filter sliding window of data to get 32 subband
/* samples per channel.
```

```
/* (window_subband,filter_subband)
/*
/* 2.  If joint stereo mode, combine left and right channels
/* for subbands above #jsbound#.
/* (*_combine_LR)
/*
/* 3.  Calculate scalefactors for the frame, and if layer 2,
/* also calculate scalefactor select information.
/* (*_scale_factor_calc)
/*
/* 4.  Calculate psychoacoustic masking levels using selected
/* psychoacoustic model.
/* (*_Psycho_One, psycho_anal)
/*
/* 5.  Perform iterative bit allocation for subbands with low
/* mask_to_noise ratios using masking levels from step 4.
/* (*_main_bit_allocation)
/*
/* 6.  If error protection flag is active, add redundancy for
/* error protection.
/* (*_CRC_calc)
/*
/* 7.  Pack bit allocation, scalefactors, and scalefactor select
/* information (layer 2) onto bitstream.
/* (*_encode_bit_alloc,*_encode_scale,II_transmission_pattern)
/*
/* 8.  Quantize subbands and pack them into bitstream
/* (*_subband_quantization, *_sample_encoding)
/*
/************************************************************************/

main(argc, argv)
int     argc;
char    **argv;
{
typedef double SBS[2][3][SCALE_BLOCK][SBLIMIT];
    SBS  FAR        *sb_sample;
typedef double JSBS[3][SCALE_BLOCK][SBLIMIT];
    JSBS FAR        *j_sample;
typedef double IN[2][HAN_SIZE];
    IN   FAR        *win_que;
typedef unsigned int SUB[2][3][SCALE_BLOCK][SBLIMIT];
    SUB  FAR        *subband;

    frame_params fr_ps;
    layer info;
    char original_file_name[MAX_NAME_SIZE];
    char encoded_file_name[MAX_NAME_SIZE];
    short FAR **win_buf;
static short FAR buffer[2][1152];
static unsigned int bit_alloc[2][SBLIMIT], scfsi[2][SBLIMIT];
static unsigned int scalar[2][3][SBLIMIT], j_scale[3][SBLIMIT];
static double FAR ltmin[2][SBLIMIT], lgmin[2][SBLIMIT], max_sc[2][SBLIMIT];
    FLOAT snr32[32];
    short sam[2][1056];
    int whole_SpF, extra_slot = 0;
    double avg_slots_per_frame, frac_SpF, slot_lag;
    int model, stereo, error_protection;
static unsigned int crc;
    int i, j, k, adb;
    unsigned long bitsPerSlot, samplesPerFrame, frameNum = 0;
    unsigned long frameBits, sentBits = 0;
    unsigned long num_samples;

#ifdef  MACINTOSH
    console_options.nrows = MAC_WINDOW_SIZE;
    argc = ccommand(&argv);
#endif

    /* Most large variables are declared dynamically to ensure
       compatibility with smaller machines */

    sb_sample = (SBS FAR *) mem_alloc(sizeof(SBS), "sb_sample");
    j_sample = (JSBS FAR *) mem_alloc(sizeof(JSBS), "j_sample");
    win_que = (IN FAR *) mem_alloc(sizeof(IN), "Win_que");
    subband = (SUB FAR *) mem_alloc(sizeof(SUB),"subband");
```

```
    win_buf = (short FAR **) mem_alloc(sizeof(short *)*2, "win_buf");

    /* clear buffers */
    memset((char *) buffer, 0, sizeof(buffer));
    memset((char *) bit_alloc, 0, sizeof(bit_alloc));
    memset((char *) scalar, 0, sizeof(scalar));
    memset((char *) j_scale, 0, sizeof(j_scale));
    memset((char *) scfsi, 0, sizeof(scfsi));
    memset((char *) ltmin, 0, sizeof(ltmin));
    memset((char *) lgmin, 0, sizeof(lgmin));
    memset((char *) max_sc, 0, sizeof(max_sc));
    memset((char *) snr32, 0, sizeof(snr32));
    memset((char *) sam, 0, sizeof(sam));

    fr_ps.header = &info;
    fr_ps.tab_num = -1;             /* no table loaded */
    fr_ps.alloc = NULL;
    info.version = MPEG_AUDIO_ID;

    programName = argv[0];
    if(argc==1)     /* no command-line args */
       obtain_parameters(&fr_ps, &model, &num_samples,
                         original_file_name, encoded_file_name);
    else
       parse_args(argc, argv, &fr_ps, &model, &num_samples,
                  original_file_name, encoded_file_name);
    print_config(&fr_ps, &model, &num_samples,
                 original_file_name, encoded_file_name);

    hdr_to_frps(&fr_ps);
    stereo = fr_ps.stereo;
    error_protection = info.error_protection;

    if (info.lay == 1) { bitsPerSlot = 32; samplesPerFrame = 384;  }
    else               { bitsPerSlot = 8;  samplesPerFrame = 1152; }
    /* Figure average number of 'slots' per frame. */
    /* Bitrate means TOTAL for both channels, not per side. */
    avg_slots_per_frame = ((double)samplesPerFrame /
                           s_freq[info.sampling_frequency]) *
                          ((double)bitrate[info.lay-1][info.bitrate_index] /
                           (double)bitsPerSlot);
    whole_SpF = (int) avg_slots_per_frame;
    printf("slots/frame = %d\n",whole_SpF);
    frac_SpF  = avg_slots_per_frame - (double)whole_SpF;
    slot_lag  = -frac_SpF;
    printf("frac SpF=%.3f, tot bitrate=%d kbps, s freq=%.1f kHz\n",
           frac_SpF, bitrate[info.lay-1][info.bitrate_index],
           s_freq[info.sampling_frequency]);

    if (frac_SpF != 0)
       printf("Fractional number of slots, padding required\n");
    else info.padding = 0;

    while (get_audio(musicin, buffer, num_samples, stereo, info.lay) > 0) {

       fprintf(stderr, "{%4lu}", frameNum++); fflush(stderr);
       win_buf[0] = &buffer[0][0];
       win_buf[1] = &buffer[1][0];
       if (frac_SpF != 0) {
          if (slot_lag > (frac_SpF-1.0) ) {
             slot_lag -= frac_SpF;
             extra_slot = 0;
             info.padding = 0;
             /*  printf("No padding for this frame\n"); */
          }
          else {
             extra_slot = 1;
             info.padding = 1;
             slot_lag += (1-frac_SpF);
             /*  printf("Padding for this frame\n");    */
          }
       }
       adb = (whole_SpF+extra_slot) * bitsPerSlot;

       switch (info.lay) {
```

```
/**************************** Layer I *********************************/

          case 1 :
             for (j=0;j<SCALE_BLOCK;j++)
             for (k=0;k<stereo;k++) {
                window_subband(&win_buf[k], &(*win_que)[k][0], k);
                filter_subband(&(*win_que)[k][0], &(*sb_sample)[k][0][j][0]);
             }

             I_scale_factor_calc(*sb_sample, scalar, stereo);
             if(fr_ps.actual_mode == MPG_MD_JOINT_STEREO) {
                I_combine_LR(*sb_sample, *j_sample);
                I_scale_factor_calc(j_sample, &j_scale, 1);
             }

             put_scale(scalar, &fr_ps, max_sc);

             if (model == 1) I_Psycho_One(buffer, max_sc, ltmin, &fr_ps);
             else {
                for (k=0;k<stereo;k++) {
                   psycho_anal(&buffer[k][0],&sam[k][0], k, info.lay, snr32,
                               (FLOAT)s_freq[info.sampling_frequency]*1000);
                   for (i=0;i<SBLIMIT;i++) ltmin[k][i] = (double) snr32[i];
                }
             }

             I_main_bit_allocation(ltmin, bit_alloc, &adb, &fr_ps);

             if (error_protection) I_CRC_calc(&fr_ps, bit_alloc, &crc);

             encode_info(&fr_ps, &bs);

             if (error_protection) encode_CRC(crc, &bs);

             I_encode_bit_alloc(bit_alloc, &fr_ps, &bs);
             I_encode_scale(scalar, bit_alloc, &fr_ps, &bs);
             I_subband_quantization(scalar, *sb_sample, j_scale, *j_sample,
                                    bit_alloc, *subband, &fr_ps);
             I_sample_encoding(*subband, bit_alloc, &fr_ps, &bs);
             for (i=0;i<adb;i++) put1bit(&bs, 0);
          break;

/**************************** Layer 2 *********************************/

          case 2 :
             for (i=0;i<3;i++) for (j=0;j<SCALE_BLOCK;j++)
                for (k=0;k<stereo;k++) {
                   window_subband(&win_buf[k], &(*win_que)[k][0], k);
                   filter_subband(&(*win_que)[k][0], &(*sb_sample)[k][i][j][0]);
                }

                II_scale_factor_calc(*sb_sample, scalar, stereo, fr_ps.sblimit);
                pick_scale(scalar, &fr_ps, max_sc);
                if(fr_ps.actual_mode == MPG_MD_JOINT_STEREO) {
                   II_combine_LR(*sb_sample, *j_sample, fr_ps.sblimit);
                   II_scale_factor_calc(j_sample, &j_scale, 1, fr_ps.sblimit);
                }       /* this way we calculate more mono than we need */
                        /* but it is cheap */

                if (model == 1) II_Psycho_One(buffer, max_sc, ltmin, &fr_ps);
                else {
                   for (k=0;k<stereo;k++) {
                      psycho_anal(&buffer[k][0],&sam[k][0], k,
                                  info.lay, snr32,
                                  (FLOAT)s_freq[info.sampling_frequency]*1000);
                      for (i=0;i<SBLIMIT;i++) ltmin[k][i] = (double) snr32[i];
                   }
                }

                II_transmission_pattern(scalar, scfsi, &fr_ps);
                II_main_bit_allocation(ltmin, scfsi, bit_alloc, &adb, &fr_ps);

                if (error_protection)
                   II_CRC_calc(&fr_ps, bit_alloc, scfsi, &crc);

                encode_info(&fr_ps, &bs);
```

```
                if (error_protection) encode_CRC(crc, &bs);

                II_encode_bit_alloc(bit_alloc, &fr_ps, &bs);
                II_encode_scale(bit_alloc, scfsi, scalar, &fr_ps, &bs);
                II_subband_quantization(scalar, *sb_sample, j_scale,
                                      *j_sample, bit_alloc, *subband, &fr_ps);
                II_sample_encoding(*subband, bit_alloc, &fr_ps, &bs);
                for (i=0;i<adb;i++) put1bit(&bs, 0);
          break;

/**************************** Layer 3 *********************************/

          case 3 : break;

       }

       frameBits = sstell(&bs) - sentBits;
       if(frameBits%bitsPerSlot)   /* a program failure */
          fprintf(stderr,"Sent %ld bits = %ld slots plus %ld\n",
                  frameBits, frameBits/bitsPerSlot,
                  frameBits%bitsPerSlot);
       sentBits += frameBits;

    }

    close_bit_stream_w(&bs);

    printf("Avg slots/frame = %.3f; b/smp = %.2f; br = %.3f kbps\n",
           (FLOAT) sentBits / (frameNum * bitsPerSlot),
           (FLOAT) sentBits / (frameNum * samplesPerFrame),
           (FLOAT) sentBits / (frameNum * samplesPerFrame) *
           s_freq[info.sampling_frequency]);

    if (fclose(musicin) != 0){
       printf("Could not close \"%s\".\n", original_file_name);
       exit(2);
    }

#ifdef  MACINTOSH
    set_mac_file_attr(encoded_file_name, VOL_REF_NUM, CREATOR_ENCODE,
                      FILETYPE_ENCODE);
#endif

    printf("Encoding of \"%s\" with psychoacoustic model %d is finished\n",
           original_file_name, model);
    printf("The MPEG encoded output file name is \"%s\"\n",
            encoded_file_name);
    exit(0);
}

/************************************************************************
/*
/* usage
/*
/* PURPOSE:  Writes command line syntax to the file specified by #stderr#
/*
/************************************************************************/

static void usage()  /* print syntax & exit */
{
    fprintf(stderr,
    "usage: %s                         queries for all arguments, or\n",
            programName);
    fprintf(stderr,
    "       %s [-l lay][-m mode][-p psy][-s sfrq][-b br][-d emp]\n",
            programName);
    fprintf(stderr,
    "          [-c][-o][-e] inputPCM [outBS]\n");
    fprintf(stderr,"where\n");
    fprintf(stderr," -l lay   use layer <lay> coding   (dflt %4u)\n",DFLT_LAY);
    fprintf(stderr," -m mode  channel mode : s/d/j/m   (dflt %4c)\n",DFLT_MOD);
    fprintf(stderr," -p psy   psychoacoustic model 1/2 (dflt %4u)\n",DFLT_PSY);
    fprintf(stderr," -s sfrq  input smpl rate in kHz   (dflt %4.1f)\n",DFLT_SFQ);
    fprintf(stderr," -b br    total bitrate in kbps    (dflt %4u)\n",DFLT_BRT);
    fprintf(stderr," -d emp   de-emphasis n/5/c        (dflt %4c)\n",DFLT_EMP);
```

```
    fprintf(stderr," -c       mark as copyright\n");
    fprintf(stderr," -o       mark as original\n");
    fprintf(stderr," -e       add error protection\n");
    fprintf(stderr," inputPCM input PCM sound file (standard or AIFF)\n");
    fprintf(stderr," outBS    output bit stream of encoded audio (dflt inName+%s)\n",
            DFLT_EXT);
    exit(1);
}

/************************************************************************
/*
/* aiff_check
/*
/* PURPOSE:  Checks AIFF header information to make sure it is valid.
/*           Exits if not.
/*
/************************************************************************/

void aiff_check(file_name, pcm_aiff_data)
char *file_name;          /* Pointer to name of AIFF file */
IFF_AIFF *pcm_aiff_data;  /* Pointer to AIFF data structure */
{

#ifdef IFF_LONG
    if (pcm_aiff_data->sampleType != IFF_ID_SSND) {
#else
    if (strncmp(&pcm_aiff_data->sampleType,IFF_ID_SSND,4)) {
#endif
       printf("Sound data is not PCM in \"%s\".\n", file_name);
       exit(1);
    }

    if(SmpFrqIndex((long)pcm_aiff_data->sampleRate) < 0) {
       printf("in \"%s\".\n", file_name);
       exit(1);
    }

    if (pcm_aiff_data->sampleSize != sizeof(short) * BITS_IN_A_BYTE) {
        printf("Sound data is not %d bits in \"%s\".\n",
               sizeof(short) * BITS_IN_A_BYTE, file_name);
        exit(1);
    }

    if (pcm_aiff_data->numChannels != MONO &&
        pcm_aiff_data->numChannels != STEREO) {
       printf("Sound data is not mono or stereo in \"%s\".\n", file_name);
       exit(1);
    }

    if (pcm_aiff_data->blkAlgn.blockSize != 0) {
       printf("Block size is not %lu bytes in \"%s\".\n", 0, file_name);
       exit(1);
    }

    if (pcm_aiff_data->blkAlgn.offset != 0) {
       printf("Block offset is not %lu bytes in \"%s\".\n", 0, file_name);
       exit(1);
    }
}
```

C.3.10 musicout.c

```
/**********************************************************************
Copyright (c) 1991 MPEG/audio software simulation group, All Rights Reserved
musicout.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress             *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
```

```
 *   date   programmers                comment                      *
 * 2/25/91  Douglas Wong        start of version 1.0 records        *
 * 3/06/91  Douglas Wong        rename setup.h to dedef.h           *
 *                              removed extraneous variables        *
 *                              removed window_samples (now part of *
 *                              filter_samples)                     *
 * 3/07/91  Davis Pan           changed output file to "codmusic"   *
 * 5/10/91  Vish (PRISM)        Ported to Macintosh and Unix.       *
 *                              Incorporated new "out_fifo()" which *
 *                              writes out last incomplete buffer.  *
 *                              Incorporated all AIFF routines which *
 *                              are also compatible with SUN.       *
 *                              Incorporated user interface for     *
 *                              specifying sound file names.        *
 *                              Also incorporated user interface for *
 *                              writing AIFF compatible sound files. *
 * 27jun91  dpwe (Aware)        Added musicout and &sample_frames as *
 *                              args to out_fifo (were glob refs).  *
 *                              Used new 'frame_params' struct.     *
 *                              Clean,simplify, track clipped output *
 *                              and total bits/frame received.      *
 * 7/10/91  Earle Jennings      changed to floats to FLOAT          *
 *10/ 1/91  S.I. Sudharsanan,   Ported to IBM AIX platform.         *
 *          Don H. Lee,                                             *
 *          Peter W. Farrett                                        *
 *10/ 3/91  Don H. Lee          implemented CRC-16 error protection *
 *                              newly introduced functions are      *
 *                              buffer_CRC and recover_CRC_error    *
 *                              Additions and revisions are marked  *
 *                              with "dhl" for clarity              *
 * 2/11/92  W. Joseph Carter    Ported new code to Macintosh.  Most *
 *                              important fixes involved changing   *
 *                              16-bit ints to long or unsigned in  *
 *                              bit alloc routines for quant of 65535 *
 *                              and passing proper function args.   *
 *                              Removed "Other Joint Stereo" option *
 *                              and made bitrate be total channel   *
 *                              bitrate, irrespective of the mode.  *
 *                              Fixed many small bugs & reorganized. *
 *19 aug 92 Soren H. Nielsen    Changed MS-DOS file name extensions. *
 * 8/27/93 Seymour Shlien,      Fixes in Unix and MSDOS ports,      *
 *         Daniel Lauzon, and                                       *
 *         Bill Truerniet                                           *
 *------------------------------------------------------------------*
 * 4/23/92  J. Pineda           Added code for layer III.  LayerIII *
 *          Amit Gulati         decoding is currently performed in  *
 *                              two-passes for ease of sideinfo and *
 *                              maindata buffering and decoding.    *
 *                              The second (computation) pass is    *
 *                              activated with "decode -3 <outfile>" *
 * 10/25/92 Amit Gulati         Modified usage() for layerIII       *
 * 12/10/92 Amit Gulati         Changed processing order of re-order- *
 *                              -ing step.  Fixed adjustment of     *
 *                              main_data_end pointer to exclude    *
 *                              side information.                   *
 *  9/07/93 Toshiyuki Ishino    Integrated Layer III with Ver 3.9.  *
 *------------------------------------------------------------------*
 * 11/20/93 Masahiro Iwadare    Integrated Layer III with Ver 4.0.  *
 *------------------------------------------------------------------*
 *  7/14/94 Juergen Koller      Bug fixes in Layer III code         *
 * 11/04/94 Jon Rowlands        Prototype fixes                     *
 ********************************************************************/

#include        "common.h"
#include        "decoder.h"

/*******************************************************************
/*
/*        This part contains the MPEG I decoder for Layers I & II.
/*
/*******************************************************************/

/****************************************************************
/*
/*        For MS-DOS user (Turbo c) change all instance of malloc
/*        to _farmalloc and free to _farfree. Compiler model hugh
```

```
/*        Also make sure all the pointer specified are changed to far.
/*
/*****************************************************************/

/*******************************************************************
/*
/* Core of the Layer II decoder.  Default layer is Layer II.
/*
/*******************************************************************/

/* Global variable definitions for "musicout.c" */

char *programName;
int main_data_slots();
int side_info_slots();

/* Implementations */

main(argc, argv)
int argc;
char **argv;
{
/*typedef short PCM[2][3][SBLIMIT];*/
typedef short PCM[2][SSLIMIT][SBLIMIT];
    PCM FAR *pcm_sample;
typedef unsigned int SAM[2][3][SBLIMIT];
    SAM FAR *sample;
typedef double FRA[2][3][SBLIMIT];
    FRA FAR *fraction;
typedef double VE[2][HAN_SIZE];
    VE FAR *w;

    Bit_stream_struc  bs;
    frame_params      fr_ps;
    layer             info;
    FILE              *musicout;
    unsigned long     sample_frames;

    int               i, j, k, stereo, done=FALSE, clip, sync;
    int               error_protection, crc_error_count, total_error_count;
    unsigned int      old_crc, new_crc;
    unsigned int      bit_alloc[2][SBLIMIT], scfsi[2][SBLIMIT],
                      scale_index[2][3][SBLIMIT];
    unsigned long     bitsPerSlot, samplesPerFrame, frameNum = 0;
    unsigned long     frameBits, gotBits = 0;
    IFF_AIFF          pcm_aiff_data;
    char              encoded_file_name[MAX_NAME_SIZE];
    char              decoded_file_name[MAX_NAME_SIZE];
    char              t[50];
    int               need_aiff;
    int               need_esps;        /* MI */
    int topSb = 0;

III_scalefac_t III_scalefac;
III_side_info_t III_side_info;

#ifdef  MACINTOSH
    console_options.nrows = MAC_WINDOW_SIZE;
    argc = ccommand(&argv);
#endif

    /* Most large variables are declared dynamically to ensure
       compatibility with smaller machines */

    pcm_sample = (PCM FAR *) mem_alloc((long) sizeof(PCM), "PCM Samp");
    sample = (SAM FAR *) mem_alloc((long) sizeof(SAM), "Sample");
    fraction = (FRA FAR *) mem_alloc((long) sizeof(FRA), "fraction");
    w = (VE FAR *) mem_alloc((long) sizeof(VE), "w");

    fr_ps.header = &info;
    fr_ps.tab_num = -1;                /* no table loaded */
    fr_ps.alloc = NULL;
    for (i=0;i<HAN_SIZE;i++) for (j=0;j<2;j++) (*w)[j][i] = 0.0;

    programName = argv[0];
    if(argc==1) {        /* no command line args -> interact */
```

```
       do {
          printf ("Enter encoded file name <required>: ");
          gets (encoded_file_name);
          if (encoded_file_name[0] == NULL_CHAR)
             printf ("Encoded file name is required. \n");
       } while (encoded_file_name[0] == NULL_CHAR);
       printf (">>> Encoded file name is: %s \n", encoded_file_name);
#ifdef  MS_DOS
       printf ("Enter MPEG decoded file name <%s>: ",
               new_ext(encoded_file_name, DFLT_OPEXT)); /* 92-08-19 shn */
#else
       printf ("Enter MPEG decoded file name <%s%s>: ", encoded_file_name,
               DFLT_OPEXT);
#endif
       gets (decoded_file_name);
       if (decoded_file_name[0] == NULL_CHAR) {
#ifdef  MS_DOS
           /* replace old extension with new one, 92-08-19 shn */
           strcpy(decoded_file_name,new_ext(encoded_file_name, DFLT_OPEXT));
#else
           strcat (strcpy(decoded_file_name, encoded_file_name), DFLT_OPEXT);
#endif
       }
       printf (">>> MPEG decoded file name is: %s \n", decoded_file_name);

       printf(
          "Do you wish to write an AIFF compatible sound file ? (y/<n>) : ");
       gets(t);
       if (*t == 'y' || *t == 'Y') need_aiff = TRUE;
       else                        need_aiff = FALSE;
       if (need_aiff)
            printf(">>> An AIFF compatible sound file will be written\n");
       else printf(">>> A non-headered PCM sound file will be written\n");

       printf(
          "Do you wish to exit (last chance before decoding) ? (y/<n>) : ");
       gets(t);
       if (*t == 'y' || *t == 'Y') exit(0);
    }
    else {        /* interpret CL Args */
       int i=0, err=0;

       need_aiff = FALSE;
       need_esps = FALSE;    /* MI */
       encoded_file_name[0] = '\0';
       decoded_file_name[0] = '\0';

       while(++i<argc && err == 0) {
          char c, *token, *arg, *nextArg;
          int  argUsed;

          token = argv[i];
          if(*token++ == '-') {
             if(i+1 < argc) nextArg = argv[i+1];
             else           nextArg = "";
             argUsed = 0;
             while(c = *token++) {
                if(*token /* NumericQ(token) */) arg = token;
                else                             arg = nextArg;
                switch(c) {
                   case 's':  topSb = atoi(arg); argUsed = 1;
                      if(topSb<1 || topSb>SBLIMIT) {
                         fprintf(stderr, "%s: -s band %s not %d..%d\n",
                                 programName, arg, 1, SBLIMIT);
                         err = 1;
                      }
                      break;
                   case 'A':  need_aiff = TRUE; break;
                   case 'E':  need_esps = TRUE; break;      /* MI */
                   default:   fprintf(stderr,"%s: unrecognized option %c\n",
                                      programName, c);
                      err = 1; break;
                }
                if(argUsed) {
                   if(arg == token) token = ""; /* no more from token */
                   else             ++i; /* skip arg we used */
```

```
                arg = ""; argUsed = 0;
             }
          }
       }
       else {
          if(encoded_file_name[0] == '\0')
             strcpy(encoded_file_name, argv[i]);
          else
             if(decoded_file_name[0] == '\0')
                strcpy(decoded_file_name, argv[i]);
             else {
                fprintf(stderr,
                        "%s: excess arg %s\n", programName, argv[i]);
                err = 1;
             }
       }
    }

    if(err || encoded_file_name[0] == '\0') usage();  /* never returns */

    if(decoded_file_name[0] == '\0') {
       strcpy(decoded_file_name, encoded_file_name);
       strcat(decoded_file_name, DFLT_OPEXT);
    }

 }
 /* report results of dialog / command line */
 printf("Input file = '%s'  output file = '%s'\n",
        encoded_file_name, decoded_file_name);
 if(need_aiff) printf("Output file written in AIFF format\n");
 if(need_esps) printf("Output file written in ESPS format\n"); /* MI */

 if ((musicout = fopen(decoded_file_name, "w+b")) == NULL) {
    printf ("Could not create \"%s\".\n", decoded_file_name);
    exit(1);
 }

 open_bit_stream_r(&bs, encoded_file_name, BUFFER_SIZE);

 if (need_aiff)
    if (aiff_seek_to_sound_data(musicout) == -1) {
       printf("Could not seek to PCM sound data in \"%s\".\n",
              decoded_file_name);
       exit(1);
    }

 sample_frames = 0;

 while (!end_bs(&bs)) {

    sync = seek_sync(&bs, SYNC_WORD, SYNC_WORD_LNGTH);
    frameBits = sstell(&bs) - gotBits;
    if(frameNum > 0)        /* don't want to print on 1st loop; no lay */
       if(frameBits%bitsPerSlot)
          fprintf(stderr,"Got %ld bits = %ld slots plus %ld\n",
                  frameBits, frameBits/bitsPerSlot, frameBits%bitsPerSlot);
    gotBits += frameBits;

    if (!sync) {
       printf("Frame cannot be located\n");
       printf("Input stream may be empty\n");
       done = TRUE;
       /* finally write out the buffer */
       if (info.lay != 1) out_fifo(*pcm_sample, 3, &fr_ps, done,
                                   musicout, &sample_frames);
       else               out_fifo(*pcm_sample, 1, &fr_ps, done,
                                   musicout, &sample_frames);
       break;
    }

    decode_info(&bs, &fr_ps);
    hdr_to_frps(&fr_ps);
    stereo = fr_ps.stereo;
    error_protection = info.error_protection;
    crc_error_count = 0;
    total_error_count = 0;
```

```
       if(frameNum == 0) WriteHdr(&fr_ps, stdout);  /* printout layer/mode */

#ifdef ESPS
if (frameNum == 0 && need_esps) {
esps_write_header(musicout,(long) sample_frames, (double)
s_freq[info.sampling_frequency] * 1000,
(int) stereo, decoded_file_name );
} /* MI */
#endif

       fprintf(stderr, "{%4lu}", frameNum++); fflush(stderr);
       if (error_protection) buffer_CRC(&bs, &old_crc);

       switch (info.lay) {

          case 1: {
             bitsPerSlot = 32;        samplesPerFrame = 384;
             I_decode_bitalloc(&bs,bit_alloc,&fr_ps);
             I_decode_scale(&bs, bit_alloc, scale_index, &fr_ps);

             if (error_protection) {
                I_CRC_calc(&fr_ps, bit_alloc, &new_crc);
                if (new_crc != old_crc) {
                   crc_error_count++;
                   total_error_count++;
                   recover_CRC_error(*pcm_sample, crc_error_count,
                                     &fr_ps, musicout, &sample_frames);
                   break;
                }
                else crc_error_count = 0;
             }

             clip = 0;
             for (i=0;i<SCALE_BLOCK;i++) {
                I_buffer_sample(&bs,(*sample),bit_alloc,&fr_ps);
                I_dequantize_sample(*sample,*fraction,bit_alloc,&fr_ps);
                I_denormalize_sample((*fraction),scale_index,&fr_ps);
                if(topSb>0)        /* clear channels to 0 */
                   for(j=topSb; j<fr_ps.sblimit; ++j)
                      for(k=0; k<stereo; ++k)
                         (*fraction)[k][0][j] = 0;

                for (j=0;j<stereo;j++) {
                   clip += SubBandSynthesis (&((*fraction)[j][0][0]), j,
                                             &((*pcm_sample)[j][0][0]));
                }
                out_fifo(*pcm_sample, 1, &fr_ps, done,
                         musicout, &sample_frames);
             }
             if(clip > 0) printf("%d output samples clipped\n", clip);
             break;
          }

          case 2: {
             bitsPerSlot = 8;        samplesPerFrame = 1152;
             II_decode_bitalloc(&bs, bit_alloc, &fr_ps);
             II_decode_scale(&bs, scfsi, bit_alloc, scale_index, &fr_ps);

             if (error_protection) {
                II_CRC_calc(&fr_ps, bit_alloc, scfsi, &new_crc);
                if (new_crc != old_crc) {
                   crc_error_count++;
                   total_error_count++;
                   recover_CRC_error(*pcm_sample, crc_error_count,
                                     &fr_ps, musicout, &sample_frames);
                   break;
                }
                else crc_error_count = 0;
             }

             clip = 0;
             for (i=0;i<SCALE_BLOCK;i++) {
                II_buffer_sample(&bs,(*sample),bit_alloc,&fr_ps);
                II_dequantize_sample((*sample),bit_alloc,(*fraction),&fr_ps);
                II_denormalize_sample((*fraction),scale_index,&fr_ps,i>>2);
```

```
        if(topSb>0)        /* debug : clear channels to 0 */
           for(j=topSb; j<fr_ps.sblimit; ++j)
              for(k=0; k<stereo; ++k)
                 (*fraction)[k][0][j] =
                 (*fraction)[k][1][j] =
                 (*fraction)[k][2][j] = 0;

        for (j=0;j<3;j++) for (k=0;k<stereo;k++) {
           clip += SubBandSynthesis (&((*fraction)[k][j][0]), k,
                                     &((*pcm_sample)[k][j][0]));
        }
        out_fifo(*pcm_sample, 3, &fr_ps, done, musicout,
                 &sample_frames);
     }
     if(clip > 0) printf("%d samples clipped\n", clip);
     break;
  }

  case 3: {
     int nSlots;
     int gr, ch, ss, sb, main_data_end, flush_main ;
 int  bytes_to_discard ;
 static int frame_start = 0;
     bitsPerSlot = 8;        samplesPerFrame = 1152;

     III_get_side_info(&bs, &III_side_info, &fr_ps);
     nSlots = main_data_slots(fr_ps);
     for (; nSlots > 0; nSlots--)  /* read main data. */
        hputbuf((unsigned int) getbits(&bs,8), 8);
 main_data_end = hsstell() / 8; /*of privious frame*/
     if ( flush_main=(hsstell() % bitsPerSlot) ) {
        hgetbits((int)(bitsPerSlot - flush_main));
main_data_end ++;
 }
     bytes_to_discard = frame_start - main_data_end
                 - III_side_info.main_data_begin ;
     if( main_data_end > 4096 )
     {   frame_start -= 4096;
         rewindNbytes( 4096 );
     }

     frame_start += main_data_slots(fr_ps);
     if (bytes_to_discard < 0) {
 printf("Not enough main data to decode frame %d.  Frame discarded.\n",
               frameNum - 1); break;
     }
     for (; bytes_to_discard > 0; bytes_to_discard--) hgetbits(8);

     clip = 0;
     for (gr=0;gr<2;gr++) {
       double lr[2][SBLIMIT][SSLIMIT],ro[2][SBLIMIT][SSLIMIT];

       for (ch=0; ch<stereo; ch++) {
         long int is[SBLIMIT][SSLIMIT];   /* Quantized samples. */
         double xr[SBLIMIT][SSLIMIT];  /* Dequantized samples. */
         int part2_start;
         part2_start = hsstell();
         III_get_scale_factors(III_scalefac,&III_side_info,gr,ch,
    &fr_ps);
         III_hufman_decode(is, &III_side_info, ch, gr, part2_start,
                          &fr_ps);
         III_dequantize_sample(is, xr, &III_scalefac,
                          &(III_side_info.ch[ch].gr[gr]), ch, &fr_ps);
         III_reorder (xr,ro[ch],&(III_side_info.ch[ch].gr[gr]), &fr_ps);
       }
       III_stereo(ro,lr,III_scalefac,
                  &(III_side_info.ch[0].gr[gr]), &fr_ps);
       for (ch=0; ch<stereo; ch++) {
            double hybridIn[SBLIMIT][SSLIMIT];/* Hybrid filter input */
            double hybridOut[SBLIMIT][SSLIMIT];/* Hybrid filter out */
            double polyPhaseIn[SBLIMIT];     /* PolyPhase Input. */
            III_antialias(lr[ch], hybridIn, /* Antialias butterflies. */
                          &(III_side_info.ch[ch].gr[gr]), &fr_ps);
            for (sb=0; sb<SBLIMIT; sb++) { /* Hybrid synthesis. */
                III_hybrid(hybridIn[sb], hybridOut[sb], sb, ch,
                           &(III_side_info.ch[ch].gr[gr]), &fr_ps);
```

```
                    }
                    for (ss=0;ss<18;ss++) /*Frequency inversion for polyphase.*/
                       for (sb=0; sb<SBLIMIT; sb++)
                          if ((ss%2) && (sb%2))
                             hybridOut[sb][ss] = -hybridOut[sb][ss];
                    for (ss=0;ss<18;ss++) { /* Polyphase synthesis */
                        for (sb=0; sb<SBLIMIT; sb++)
                            polyPhaseIn[sb] = hybridOut[sb][ss];
                        clip += SubBandSynthesis (polyPhaseIn, ch,
                                                  &((*pcm_sample)[ch][ss][0]));
                        }
                    }
                /* Output PCM sample points for one granule. */
                out_fifo(*pcm_sample, 18, &fr_ps, done, musicout,
                         &sample_frames);
             }
             if(clip > 0) printf("%d samples clipped.\n", clip);
             break;
          }
       }
    }

    if (need_aiff) {
       pcm_aiff_data.numChannels       = stereo;
       pcm_aiff_data.numSampleFrames   = sample_frames;
       pcm_aiff_data.sampleSize        = 16;
       pcm_aiff_data.sampleRate        = s_freq[info.sampling_frequency]*1000;
#ifdef IFF_LONG
       pcm_aiff_data.sampleType        = IFF_ID_SSND;
#else
       strncpy(&pcm_aiff_data.sampleType,IFF_ID_SSND,4);
#endif
       pcm_aiff_data.blkAlgn.offset    = 0;
       pcm_aiff_data.blkAlgn.blockSize = 0;

       if (aiff_write_headers(musicout, &pcm_aiff_data) == -1) {
          printf("Could not write AIFF headers to \"%s\"\n",
                 decoded_file_name);
          exit(2);
       }
    }

    printf("Avg slots/frame = %.3f; b/smp = %.2f; br = %.3f kbps\n",
           (FLOAT) gotBits / (frameNum * bitsPerSlot),
           (FLOAT) gotBits / (frameNum * samplesPerFrame),
           (FLOAT) gotBits / (frameNum * samplesPerFrame) *
           s_freq[info.sampling_frequency]);

    close_bit_stream_r(&bs);
    fclose(musicout);

    /* for the correct AIFF header information */
    /*             on the Macintosh            */
    /* the file type and the file creator for  */
    /* Macintosh compatible Digidesign is set  */

#ifdef  MACINTOSH
    if (need_aiff) set_mac_file_attr(decoded_file_name, VOL_REF_NUM,
                                     CREATR_DEC_AIFF, FILTYP_DEC_AIFF);
    else           set_mac_file_attr(decoded_file_name, VOL_REF_NUM,
                                     CREATR_DEC_BNRY, FILTYP_DEC_BNRY);
#endif

    printf("Decoding of \"%s\" is finished\n", encoded_file_name);
    printf("The decoded PCM output file name is \"%s\"\n", decoded_file_name);
    if (need_aiff)
       printf("\"%s\" has been written with AIFF header information\n",
              decoded_file_name);

    exit( 0 );
}

static void usage()  /* print syntax & exit */
{
   fprintf(stderr,
      "usage: %s                         queries for all arguments, or\n",
```

```
        programName);
    fprintf(stderr,
       "       %s [-A][-s sb] inputBS [outPCM]\n", programName);
    fprintf(stderr,"where\n");
    fprintf(stderr," -A       write an AIFF output PCM sound file\n");
    fprintf(stderr," -s sb    resynth only up to this sb (debugging only)\n");
    fprintf(stderr," inputBS  input bit stream of encoded audio\n");
    fprintf(stderr," outPCM   output PCM sound file (dflt inName+%s)\n",
            DFLT_OPEXT);
    exit(1);
}
```

C.3.11 psy.c

```
/**********************************************************************
Copyright (c) 1991 ISO/IEC JTC1 SC29 WG1, All Rights Reserved
psy.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Davis Pan           start of version 1.0 records          *
 * 5/10/91  W. Joseph Carter    Ported to Macintosh and Unix.         *
 * 7/10/91  Earle Jennings      Ported to MsDos.                      *
 *                              replace of floats with FLOAT          *
 * 2/11/92  W. Joseph Carter    Fixed mem_alloc() arg for "absthr".   *
 * 7/24/92  M. Iwadare          HANN window coefficients modified.    *
 * 7/27/92  Masahiro Iwadare    Bug fix, FFT modification for Layer 3 *
 * 7/27/92  Masahiro Iwadare    Bug fix, "new", "old", and "oldest"   *
 *                              updates                               *
 * 8/07/92  Mike Coleman        Bug fix, read_absthr()                *
 **********************************************************************/

#include "common.h"
#include "encoder.h"
FILE    *fpo;   /* file pointer */
void psycho_anal(buffer,savebuf,chn,lay,snr32,sfreq)
short int *buffer;
short int savebuf[1056];
int   chn, lay;
FLOAT snr32[32];
double sfreq;        /* to match prototype : float args are always double */
{
 unsigned int   i, j, k;
 FLOAT          r_prime, phi_prime;
 FLOAT          freq_mult, bval_lo, minthres, sum_energy;
 double         tb, temp1, temp2, temp3;

/* The static variables "r", "phi_sav", "new", "old" and "oldest" have    */
/* to be remembered for the unpredictability measure.  For "r" and        */
/* "phi_sav", the first index from the left is the channel select and     */
/* the second index is the "age" of the data.                             */

 static int     new = 0, old = 1, oldest = 0;
 static int     init = 0, flush, sync_flush, syncsize, sfreq_idx;

/* The following static variables are constants.                           */

 static double  nmt = 5.5;

 static FLOAT   crit_band[27] = {0,  100,  200, 300, 400, 510, 630,  770,
                              920, 1080, 1270,1480,1720,2000,2320, 2700,
                             3150, 3700, 4400,5300,6400,7700,9500,12000,
                            15500,25000,30000};

 static FLOAT   bmax[27] = {20.0, 20.0, 20.0, 20.0, 20.0, 17.0, 15.0,
                           10.0,  7.0,  4.4,  4.5,  4.5,  4.5,  4.5,
                            4.5,  4.5,  4.5,  4.5,  4.5,  4.5,  4.5,
                            4.5,  4.5,  4.5,  3.5,  3.5,  3.5};
```

```
/* The following pointer variables point to large areas of memory         */
/* dynamically allocated by the mem_alloc() function.  Dynamic memory     */
/* allocation is used in order to avoid stack frame or data area          */
/* overflow errors that otherwise would have occurred at compile time     */
/* on the Macintosh computer.                                             */

 FLOAT          *grouped_c, *grouped_e, *nb, *cb, *ecb, *bc;
 FLOAT          *wsamp_r, *wsamp_i, *phi, *energy;
 FLOAT          *c, *fthr;
 F32            *snrtmp;

 static int     *numlines;
 static int     *partition;
 static FLOAT   *cbval, *rnorm;
 static FLOAT   *window;
 static FLOAT   *absthr;
 static double  *tmn;
 static FCB     *s;
 static FHBLK   *lthr;
 static F2HBLK  *r, *phi_sav;

/* These dynamic memory allocations simulate "automatic" variables        */
/* placed on the stack.  For each mem_alloc() call here, there must be    */
/* a corresponding mem_free() call at the end of this function.           */

 grouped_c = (FLOAT *) mem_alloc(sizeof(FCB), "grouped_c");
 grouped_e = (FLOAT *) mem_alloc(sizeof(FCB), "grouped_e");
 nb = (FLOAT *) mem_alloc(sizeof(FCB), "nb");
 cb = (FLOAT *) mem_alloc(sizeof(FCB), "cb");
 ecb = (FLOAT *) mem_alloc(sizeof(FCB), "ecb");
 bc = (FLOAT *) mem_alloc(sizeof(FCB), "bc");
 wsamp_r = (FLOAT *) mem_alloc(sizeof(FBLK), "wsamp_r");
 wsamp_i = (FLOAT *) mem_alloc(sizeof(FBLK), "wsamp_i");
 phi = (FLOAT *) mem_alloc(sizeof(FBLK), "phi");
 energy = (FLOAT *) mem_alloc(sizeof(FBLK), "energy");
 c = (FLOAT *) mem_alloc(sizeof(FHBLK), "c");
 fthr = (FLOAT *) mem_alloc(sizeof(FHBLK), "fthr");
 snrtmp = (F32 *) mem_alloc(sizeof(F2_32), "snrtmp");

 if(init==0){

/* These dynamic memory allocations simulate "static" variables placed    */
/* in the data space.  Each mem_alloc() call here occurs only once at     */
/* initialization time.  The mem_free() function must not be called.      */

     numlines = (int *) mem_alloc(sizeof(ICB), "numlines");
     partition = (int *) mem_alloc(sizeof(IHBLK), "partition");
     fpo = fopen("out.dat", "wb");
    if(fpo==NULL) {
         puts("\t The attempt to open the output file failed.\n");
         exit(-1);}
     cbval = (FLOAT *) mem_alloc(sizeof(FCB), "cbval");
     rnorm = (FLOAT *) mem_alloc(sizeof(FCB), "rnorm");
     window = (FLOAT *) mem_alloc(sizeof(FBLK), "window");
     absthr = (FLOAT *) mem_alloc(sizeof(FHBLK), "absthr");
     tmn = (double *) mem_alloc(sizeof(DCB), "tmn");
     s = (FCB *) mem_alloc(sizeof(FCBCB), "s");
     lthr = (FHBLK *) mem_alloc(sizeof(F2HBLK), "lthr");
     r = (F2HBLK *) mem_alloc(sizeof(F22HBLK), "r");
     phi_sav = (F2HBLK *) mem_alloc(sizeof(F22HBLK), "phi_sav");

     i = sfreq + 0.5;
     switch(i){
        case 32000: sfreq_idx = 0; break;
        case 44100: sfreq_idx = 1; break;
        case 48000: sfreq_idx = 2; break;
        default:    printf("error, invalid sampling frequency: %d Hz\n",i);
        exit(-1);
     }
     printf("absthr[][] sampling frequency index: %d\n",sfreq_idx);
     read_absthr(absthr, sfreq_idx);
     if(lay==1){
        flush = 384;
        syncsize = 1024;
        sync_flush = 576;
```

```
     }
     else {
        flush = 384*3.0/2.0;
        syncsize = 1056;
        sync_flush = syncsize - flush;
     }
/* calculate HANN window coefficients */
/*   for(i=0;i<BLKSIZE;i++)window[i]=0.5*(1-cos(2.0*PI*i/(BLKSIZE-1.0))); */
     for(i=0;i<BLKSIZE;i++)window[i]=0.5*(1-cos(2.0*PI*(i-0.5)/BLKSIZE));
/* reset states used in unpredictability measure */
     for(i=0;i<HBLKSIZE;i++){
        r[0][0][i]=r[1][0][i]=r[0][1][i]=r[1][1][i]=0;
        phi_sav[0][0][i]=phi_sav[1][0][i]=0;
        phi_sav[0][1][i]=phi_sav[1][1][i]=0;
        lthr[0][i] = 60802371420160.0;
        lthr[1][i] = 60802371420160.0;
     }
/*****************************************************************************
 * Initialization: Compute the following constants for use later            *
 *    partition[HBLKSIZE] = the partition number associated with each        *
 *                          frequency line                                   *
 *    cbval[CBANDS]       = the center (average) bark value of each          *
 *                          partition                                        *
 *    numlines[CBANDS]    = the number of frequency lines in each partition  *
 *    tmn[CBANDS]         = tone masking noise                               *
 *****************************************************************************/
/* compute fft frequency multiplicand */
     freq_mult = sfreq/BLKSIZE;

/* calculate fft frequency, then bval of each line (use fthr[] as tmp storage)*/
     for(i=0;i<HBLKSIZE;i++){
        temp1 = i*freq_mult;
        j = 1;
        while(temp1>crit_band[j])j++;
        fthr[i]=j-1+(temp1-crit_band[j-1])/(crit_band[j]-crit_band[j-1]);
     }
     partition[0] = 0;
/* temp2 is the counter of the number of frequency lines in each partition */
     temp2 = 1;
     cbval[0]=fthr[0];
     bval_lo=fthr[0];
     for(i=1;i<HBLKSIZE;i++){
        if((fthr[i]-bval_lo)>0.33){
           partition[i]=partition[i-1]+1;
           cbval[partition[i-1]] = cbval[partition[i-1]]/temp2;
           cbval[partition[i]] = fthr[i];
           bval_lo = fthr[i];
           numlines[partition[i-1]] = temp2;
           temp2 = 1;
        }
        else {
           partition[i]=partition[i-1];
           cbval[partition[i]] += fthr[i];
           temp2++;
        }
     }
     numlines[partition[i-1]] = temp2;
     cbval[partition[i-1]] = cbval[partition[i-1]]/temp2;

/************************************************************************
 * Now compute the spreading function, s[j][i], the value of the spread-*
 * ing function, centered at band j, for band i, store for later use    *
 ************************************************************************/
     for(j=0;j<CBANDS;j++){
        for(i=0;i<CBANDS;i++){
           temp1 = (cbval[i] - cbval[j])*1.05;
           if(temp1>=0.5 && temp1<=2.5){
              temp2 = temp1 - 0.5;
              temp2 = 8.0 * (temp2*temp2 - 2.0 * temp2);
           }
           else temp2 = 0;
           temp1 += 0.474;
           temp3 = 15.811389+7.5*temp1-17.5*sqrt((double) (1.0+temp1*temp1));
           if(temp3 <= -100) s[i][j] = 0;
           else {
              temp3 = (temp2 + temp3)*LN_TO_LOG10;
```

```
              s[i][j] = exp(temp3);
           }
        }
     }

  /* Calculate Tone Masking Noise values */
     for(j=0;j<CBANDS;j++){
        temp1 = 15.5 + cbval[j];
        tmn[j] = (temp1>24.5) ? temp1 : 24.5;
  /* Calculate normalization factors for the net spreading functions */
        rnorm[j] = 0;
        for(i=0;i<CBANDS;i++){
           rnorm[j] += s[j][i];
        }
     }
     init++;
 }

/************************* End of Initialization *****************************/
 switch(lay) {
  case 1:
  case 2:
     for(i=0; i<lay; i++){
/*****************************************************************************
 * Net offset is 480 samples (1056-576) for layer 2; this is because one must*
 * stagger input data by 256 samples to synchronize psychoacoustic model with*
 * filter bank outputs, then stagger so that center of 1024 FFT window lines *
 * up with center of 576 "new" audio samples.                                *
 *                                                                           *
 * For layer 1, the input data still needs to be staggered by 256 samples,   *
 * then it must be staggered again so that the 384 "new" samples are centered*
 * in the 1024 FFT window.  The net offset is then 576 and you need 448 "new"*
 * samples for each iteration to keep the 384 samples of interest centered   *
 *****************************************************************************/
        for(j=0; j<syncsize; j++){
           if(j<(sync_flush))savebuf[j] = savebuf[j+flush];
           else savebuf[j] = *buffer++;
           if(j<BLKSIZE){
/**window data with HANN window***********************************************/
              wsamp_r[j] = window[j]*((FLOAT) savebuf[j]);
              wsamp_i[j] = 0;
           }
        }
/**Compute FFT****************************************************************/
        fft(wsamp_r,wsamp_i,energy,phi,1024);
/*****************************************************************************
 * calculate the unpredictability measure, given energy[f] and phi[f]        *
 *****************************************************************************/
/*only update data "age" pointers after you are done with both channels      */
/*for layer 1 computations, for the layer 2 double computations, the pointers*/
/*are reset automatically on the second pass                                 */
         if(lay==2 || (lay==1 && chn==0) ){
           if(new==0){new = 1; oldest = 1;}
           else {new = 0; oldest = 0;}
           if(old==0)old = 1; else old = 0;
        }
        for(j=0; j<HBLKSIZE; j++){
           r_prime = 2.0 * r[chn][old][j] - r[chn][oldest][j];
           phi_prime = 2.0 * phi_sav[chn][old][j] - phi_sav[chn][oldest][j];
           r[chn][new][j] = sqrt((double) energy[j]);
           phi_sav[chn][new][j] = phi[j];
temp1=r[chn][new][j] * cos((double) phi[j]) - r_prime * cos((double) phi_prime);
temp2=r[chn][new][j] * sin((double) phi[j]) - r_prime * sin((double) phi_prime);
           temp3=r[chn][new][j] + fabs((double)r_prime);
           if(temp3 != 0)c[j]=sqrt(temp1*temp1+temp2*temp2)/temp3;
           else c[j] = 0;
        }
/*****************************************************************************
 * Calculate the grouped, energy-weighted, unpredictability measure,         *
 * grouped_c[], and the grouped energy. grouped_e[]                          *
 *****************************************************************************/
        for(j=1;j<CBANDS;j++){
           grouped_e[j] = 0;
           grouped_c[j] = 0;
        }
        grouped_e[0] = energy[0];
```

```
        grouped_c[0] = energy[0]*c[0];
        for(j=1;j<HBLKSIZE;j++){
           grouped_e[partition[j]] += energy[j];
           grouped_c[partition[j]] += energy[j]*c[j];
        }

/*****************************************************************************
 * convolve the grouped energy-weighted unpredictability measure             *
 * and the grouped energy with the spreading function, s[j][k]               *
 *****************************************************************************/
        for(j=0;j<CBANDS;j++){
           ecb[j] = 0;
           cb[j] = 0;
           for(k=0;k<CBANDS;k++){
              if(s[j][k] != 0.0){
                 ecb[j] += s[j][k]*grouped_e[k];
                 cb[j] += s[j][k]*grouped_c[k];
              }
           }
           if(ecb[j] !=0)cb[j] = cb[j]/ecb[j];
           else cb[j] = 0;
        }

/*****************************************************************************
 * Calculate the required SNR for each of the frequency partitions           *
 *         this whole section can be accomplished by a table lookup          *
 *****************************************************************************/
        for(j=0;j<CBANDS;j++){
           if(cb[j]<.05)cb[j]=0.05;
           else if(cb[j]>.5)cb[j]=0.5;
           tb = -0.434294482*log((double) cb[j])-0.301029996;
       cb[j]=tb;
           bc[j] = tmn[j]*tb + nmt*(1.0-tb);
           k = cbval[j] + 0.5;
           bc[j] = (bc[j] > bmax[k]) ? bc[j] : bmax[k];
           bc[j] = exp((double) -bc[j]*LN_TO_LOG10);
        }

/*****************************************************************************
 * Calculate the permissible noise energy level in each of the frequency     *
 * partitions. Include absolute threshold and pre-echo controls              *
 *         this whole section can be accomplished by a table lookup          *
 *****************************************************************************/
        for(j=0;j<CBANDS;j++)
           if(rnorm[j] && numlines[j])
              nb[j] = ecb[j]*bc[j]/(rnorm[j]*numlines[j]);
           else nb[j] = 0;
        for(j=0;j<HBLKSIZE;j++){
/*temp1 is the preliminary threshold */
           temp1=nb[partition[j]];
           temp1=(temp1>absthr[j])?temp1:absthr[j];
/*do not use pre-echo control for layer 2 because it may do bad things to the*/
/*  MUSICAM bit allocation algorithm                                          */
           if(lay==1){
              fthr[j] = (temp1 < lthr[chn][j]) ? temp1 : lthr[chn][j];
              temp2 = temp1 * 0.00316;
              fthr[j] = (temp2 > fthr[j]) ? temp2 : fthr[j];
           }
           else fthr[j] = temp1;
           lthr[chn][j] = LXMIN*temp1;
        }

/*****************************************************************************
 * Translate the 512 threshold values to the 32 filter bands of the coder    *
 *****************************************************************************/
        for(j=0;j<193;j += 16){
           minthres = 60802371420160.0;
           sum_energy = 0.0;
           for(k=0;k<17;k++){
              if(minthres>fthr[j+k])minthres = fthr[j+k];
              sum_energy += energy[j+k];
           }
           snrtmp[i][j/16] = sum_energy/(minthres * 17.0);
           snrtmp[i][j/16] = 4.342944819 * log((double)snrtmp[i][j/16]);
        }
        for(j=208;j<(HBLKSIZE-1);j += 16){
```

```
          minthres = 0.0;
          sum_energy = 0.0;
          for(k=0;k<17;k++){
             minthres += fthr[j+k];
             sum_energy += energy[j+k];
          }
          snrtmp[i][j/16] = sum_energy/minthres;
          snrtmp[i][j/16] = 4.342944819 * log((double)snrtmp[i][j/16]);
       }
/*****************************************************************************
 * End of Psychoacuostic calculation loop                                    *
 *****************************************************************************/
     }
     for(i=0; i<32; i++){
        if(lay==2)
           snr32[i]=(snrtmp[0][i]>snrtmp[1][i])?snrtmp[0][i]:snrtmp[1][i];
        else snr32[i]=snrtmp[0][i];
     }
     break;
  case 3:
     printf("layer 3 is not currently supported\n");
     break;
  default:
     printf("error, invalid MPEG/audio coding layer: %d\n",lay);
 }
for(j=0;j<16;j++)
  for(i=0;i<32;i++){
    k = i + 32*j;
    fprintf(fpo,"%3d,enr:thr:abthr,%8.4f,%8.4f,%8.4f",
    k, 4.342944819 * log((double)energy[k]),
    4.342944819 * log((double)nb[partition[k]]),
    4.342944819 * log((double)fthr[k]));
    if(k<32) fprintf(fpo,",snr,%8.4f", snr32[k]);
    if(k<CBANDS) fprintf(fpo,",prte:sprd_prte:tnidx,%8.4f,%8.4f,%8.6f\n",
            4.342944819 * log((double)grouped_e[k]),
            4.342944819 * log((double)ecb[k]), cb[k]);
    else fprintf(fpo,"\n");
}

/* These mem_free() calls must correspond with the mem_alloc() calls     */
/* used at the beginning of this function to simulate "automatic"        */
/* variables placed on the stack.                                        */

 mem_free((void **) &grouped_c);
 mem_free((void **) &grouped_e);
 mem_free((void **) &nb);
 mem_free((void **) &cb);
 mem_free((void **) &ecb);
 mem_free((void **) &bc);
 mem_free((void **) &wsamp_r);
 mem_free((void **) &wsamp_i);
 mem_free((void **) &phi);
 mem_free((void **) &energy);
 mem_free((void **) &c);
 mem_free((void **) &fthr);
 mem_free((void **) &snrtmp);
}

/*****************************************************************************
routine to read in absthr table from a file.
*****************************************************************************/

void read_absthr(absthr, table)
FLOAT *absthr;
int table;
{
 FILE *fp;
 long j,index;
 float a;
 char t[80];
 char ta[16];

 strcpy( ta, "absthr_0" );

 switch(table){
    case 0 : ta[7] = '0';
```

```
            break;
    case 1 : ta[7] = '1';
            break;
    case 2 : ta[7] = '2';
            break;
    default : printf("absthr table: Not valid table number\n");
 }
 if(!(fp = OpenTableFile(ta) ) ){
    printf("Please check %s table\n", ta);
    exit(1);
 }
 fgets(t, 150, fp);
 sscanf(t, "table %ld", &index);
 if(index != table){
    printf("error in absthr table %s",ta);
    exit(1);
 }
 for(j=0; j<HBLKSIZE; j++){
    fgets(t,80,fp);
    sscanf(t,"%f", &a);
    absthr[j] =  a;
 }
 fclose(fp);
}
```

C.3.12 subs.c

```
/**********************************************************************
Copyright (c) 1991 ISO/IEC JTC1 SC29 WG1, All Rights Reserved
subs.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Davis Pan           start of version 1.0 records          *
 * 5/10/91  W. Joseph Carter    Ported to Macintosh and Unix.         *
 * 7/10/91  Earle Jennings      Ported to MsDos from Macintosh        *
 *                              Replacement of one float with FLOAT   *
 * 2/11/92  W. Joseph Carter    Added type casting to memset() args.  *
 * 4/27/92  Masahiro Iwadare    Added 256 point version for Layer III *
 **********************************************************************/

#include "common.h"
#include "encoder.h"

/*****************************************************************************
 ************************** Start of Subroutines *****************************
 *****************************************************************************/

/*****************************************************************************
 * FFT computes fast fourier transform of BLKSIZE samples of data            *
 *   uses decimation-in-frequency algorithm described in "Digital            *
 *   Signal Processing" by Oppenheim and Schafer, refer to pages 304         *
 *   (flow graph) and 330-332 (Fortran program in problem 5)                 *
 *   to get the inverse fft, change line 20 from                             *
 *                 w_imag[L] = -sin(PI/le1);                                 *
 *                          to                                               *
 *                 w_imag[L] = sin(PI/le1);                                  *
 *                                                                           *
 *   required constants:                                                     *
 *         #define      PI          3.14159265358979                         *
 *         #define      BLKSIZE     1024                                     *
 *         #define      LOGBLKSIZE  10                                       *
 *         #define      BLKSIZE_S   256                                      *
 *         #define      LOGBLKSIZE_S 8                                       *
 *                                                                           *
 *****************************************************************************/
```

```
#define      BLKSIZE_S   256
#define      LOGBLKSIZE_S 8

void fft(x_real,x_imag, energy, phi, N)
FLOAT x_real[BLKSIZE], x_imag[BLKSIZE], energy[BLKSIZE], phi[BLKSIZE];
int N;
{
 int     M,MM1;
 static int     init=0;
 int     NV2, NM1, MP;
 static double  w_real[2][LOGBLKSIZE], w_imag[2][LOGBLKSIZE];
 int            i,j,k,L;
 int            ip, le,le1;
 double         t_real, t_imag, u_real, u_imag;

 if(init==0) {
    memset((char *) w_real, 0, sizeof(w_real));  /* preset statics to 0 */
    memset((char *) w_imag, 0, sizeof(w_imag));  /* preset statics to 0 */
    M = LOGBLKSIZE;
    for(L=0; L<M; L++){
       le = 1 << (M-L);
       le1 = le >> 1;
       w_real[0][L] = cos(PI/le1);
       w_imag[0][L] = -sin(PI/le1);
    }
    M = LOGBLKSIZE_S;
    for(L=0; L<M; L++){
       le = 1 << (M-L);
       le1 = le >> 1;
       w_real[1][L] = cos(PI/le1);
       w_imag[1][L] = -sin(PI/le1);
    }
    init++;
 }
 switch(N) {
    case BLKSIZE:
            M = LOGBLKSIZE;
            MP = 0;
            break;
    case BLKSIZE_S:
            M = LOGBLKSIZE_S;
            MP = 1;
            break;
    default: printf("Error: Bad FFT Size in subs.c\n");
            exit(-1);
 }
 MM1 = M-1;
 NV2 = N >> 1;
 NM1 = N - 1;
 for(L=0; L<MM1; L++){
    le = 1 << (M-L);
    le1 = le >> 1;
    u_real = 1;
    u_imag = 0;
    for(j=0; j<le1; j++){
       for(i=j; i<N; i+=le){
          ip = i + le1;
          t_real = x_real[i] + x_real[ip];
          t_imag = x_imag[i] + x_imag[ip];
          x_real[ip] = x_real[i] - x_real[ip];
          x_imag[ip] = x_imag[i] - x_imag[ip];
          x_real[i] = t_real;
          x_imag[i] = t_imag;
          t_real = x_real[ip];
          x_real[ip] = x_real[ip]*u_real - x_imag[ip]*u_imag;
          x_imag[ip] = x_imag[ip]*u_real + t_real*u_imag;
       }
       t_real = u_real;
       u_real = u_real*w_real[MP][L] - u_imag*w_imag[MP][L];
       u_imag = u_imag*w_real[MP][L] + t_real*w_imag[MP][L];
    }
 }
 /* special case: L = M-1; all Wn = 1 */
 for(i=0; i<N; i+=2){
    ip = i + 1;
    t_real = x_real[i] + x_real[ip];
```

```
    t_imag = x_imag[i] + x_imag[ip];
    x_real[ip] = x_real[i] - x_real[ip];
    x_imag[ip] = x_imag[i] - x_imag[ip];
    x_real[i] = t_real;
    x_imag[i] = t_imag;
    energy[i] = x_real[i]*x_real[i] + x_imag[i]*x_imag[i];
    if(energy[i] <= 0.0005){phi[i] = 0;energy[i] = 0.0005;}
    else phi[i] = atan2((double) x_imag[i],(double) x_real[i]);
    energy[ip] = x_real[ip]*x_real[ip] + x_imag[ip]*x_imag[ip];
    if(energy[ip] == 0)phi[ip] = 0;
    else phi[ip] = atan2((double) x_imag[ip],(double) x_real[ip]);
 }
 /* this section reorders the data to the correct ordering */
 j = 0;
 for(i=0; i<NM1; i++){
    if(i<j){
/* use this section only if you need the FFT in complex number form *
 * (and in the correct ordering)                                    */
       t_real = x_real[j];
       t_imag = x_imag[j];
       x_real[j] = x_real[i];
       x_imag[j] = x_imag[i];
       x_real[i] = t_real;
       x_imag[i] = t_imag;
/* reorder the energy and phase, phi                                        */
       t_real = energy[j];
       energy[j] = energy[i];
       energy[i] = t_real;
       t_real = phi[j];
       phi[j] = phi[i];
       phi[i] = t_real;
    }
    k=NV2;
    while(k<=j){
       j = j-k;
       k = k >> 1;
    }
    j = j+k;
 }
}
```

C.3.13 tonal.c

```
/**********************************************************************
Copyright (c) 1991 ISO/IEC JTC1 SC29 WG1, All Rights Reserved
tonal.c
**********************************************************************/
/**********************************************************************
 * MPEG/audio coding/decoding software, work in progress              *
 *   NOT for public distribution until verified and approved by the   *
 *   MPEG/audio committee.  For further information, please contact   *
 *   Davis Pan, 708-538-5671, e-mail: pan@ukraine.corp.mot.com        *
 *                                                                    *
 * VERSION 4.3                                                        *
 *   changes made since last update:                                  *
 *   date   programmers         comment                               *
 * 2/25/91  Douglas Wong        start of version 1.1 records          *
 * 3/06/91  Douglas Wong        rename: setup.h to endef.h            *
 *                              updated I_psycho_one and II_psycho_one*
 * 3/11/91  W. J. Carter        Added Douglas Wong's updates dated    *
 *                              3/9/91 for I_Psycho_One() and for     *
 *                              II_Psycho_One().                      *
 * 5/10/91  W. Joseph Carter    Ported to Macintosh and Unix.         *
 *                              Located and fixed numerous software   *
 *                              bugs and table data errors.           *
 * 6/11/91  Davis Pan           corrected several bugs                *
 *                              based on comments from H. Fuchs       *
 * 01jul91  dpwe (Aware Inc.)   Made pow() args float                 *
 *                              Removed logical bug in I_tonal_label: *
 *                              Sometimes *tone returned == STOP      *
 * 7/10/91  Earle Jennings      no change necessary in port to MsDos  *
 * 11sep91  dpwe@aware.com      Subtracted 90.3dB from II_f_f_t peaks *
 * 10/1/91  Peter W. Farrett    Updated II_Psycho_One(),I_Psycho_One()*
 *                              to include comments.                  *
 *11/29/91  Masahiro Iwadare    Bug fix regarding POWERNORM           *
```

```
 *                               fixed several other miscellaneous bugs*
 * 2/11/92  W. Joseph Carter     Ported new code to Macintosh.  Most   *
 *                               important fixes involved changing     *
 *                               16-bit ints to long or unsigned in    *
 *                               bit alloc routines for quant of 65535 *
 *                               and passing proper function args.     *
 *                               Removed "Other Joint Stereo" option   *
 *                               and made bitrate be total channel     *
 *                               bitrate, irrespective of the mode.    *
 *                               Fixed many small bugs & reorganized.  *
 * 2/12/92  Masahiro Iwadare     Fixed some potential bugs in          *
 *          Davis Pan            subsampling()                         *
 * 2/25/92  Masahiro Iwadare     Fixed some more potential bugs        *
 * 6/24/92  Tan Ah Peng          Modified window for FFT               *
 *                               (denominator N-1 to N)                *
 *                               Updated all critical band rate &      *
 *                               absolute threshold tables and critical*
 *                               boundaries for use with Layer I & II  *
 *                               Corrected boundary limits for tonal   *
 *                               component computation                 *
 *                               Placement of non-tonal component at   *
 *                               geometric mean of critical band       *
 *                               (previous placement method commented  *
 *                                out - can be used if desired)        *
 * 3/01/93  Mike Li              Infinite looping fix in noise_label() *
 * 3/19/93  Jens Spille          fixed integer overflow problem in     *
 *                               psychoacoutic model 1                 *
 * 3/19/93  Giorgio Dimino       modifications to better account for   *
 *                               tonal and non-tonal components        *
 * 5/28/93 Sriram Jayasimha      "London" mod. to psychoacoustic model1*
 * 8/05/93 Masahiro Iwadare      noise_label modification "option"     *
 * 1/21/94 Seymore Shlien        fixed another infinite looping problem*
 **********************************************************************/

#include "common.h"
#include "encoder.h"
#define LONDON                  /* enable "LONDON" modification */
#define MAKE_SENSE              /* enable "MAKE_SENSE" modification */
#define MI_OPTION               /* enable "MI_OPTION" modification */
/**********************************************************************/
/*
/*        This module implements the psychoacoustic model I for the
/* MPEG encoder layer II. It uses simplified tonal and noise masking
/* threshold analysis to generate SMR for the encoder bit allocation
/* routine.
/*
/**********************************************************************/

int crit_band;
int FAR *cbound;
int sub_size;

void read_cbound(lay,freq)  /* this function reads in critical */
int lay, freq;              /* band boundaries                 */
{
 int i,j,k;
 FILE *fp;
 char r[16], t[80];

 strcpy(r, "2cb1");
 r[0] = (char) lay + '0';
 r[3] = (char) freq + '0';
 if( !(fp = OpenTableFile(r)) ){       /* check boundary values */
    printf("Please check %s boundary table\n",r);
    exit(1);
 }
 fgets(t,80,fp);               /* read input for critical bands */
 sscanf(t,"%d\n",&crit_band);
 cbound = (int FAR *) mem_alloc(sizeof(int) * crit_band, "cbound");
 for(i=0;i<crit_band;i++){   /* continue to read input for */
    fgets(t,80,fp);            /* critical band boundaries   */
    sscanf(t,"%d %d\n",&j, &k);
    if(i==j) cbound[j] = k;
    else {                     /* error */
       printf("Please check index %d in cbound table %s\n",i,r);
       exit(1);
```

```
    }
 }
 fclose(fp);
}

void read_freq_band(ltg,lay,freq)  /* this function reads in   */
int lay, freq;                     /* frequency bands and bark */
g_ptr FAR *ltg;                /* values                   */
{
 int i,j, k;
 double b,c;
 FILE *fp;
 char r[16], t[80];

 strcpy(r, "2th1");
 r[0] = (char) lay + '0';
 r[3] = (char) freq + '0';
 if( !(fp = OpenTableFile(r)) ){   /* check freq. values  */
    printf("Please check frequency and cband table %s\n",r);
    exit(1);
 }
 fgets(t,80,fp);              /* read input for freq. subbands */
 sscanf(t,"%d\n",&sub_size);
 *ltg = (g_ptr FAR ) mem_alloc(sizeof(g_thres) * sub_size, "ltg");
 (*ltg)[0].line = 0;          /* initialize global masking threshold */
 (*ltg)[0].bark = 0;
 (*ltg)[0].hear = 0;
 for(i=1;i<sub_size;i++){    /* continue to read freq. subband */
    fgets(t,80,fp);          /* and assign                     */
    sscanf(t,"%d %d %lf %lf\n",&j, &k, &b, &c);
    if(i == j){
       (*ltg)[j].line = k;
       (*ltg)[j].bark = b;
       (*ltg)[j].hear = c;
    }
    else {                   /* error */
       printf("Please check index %d in freq-cb table %s\n",i,r);
       exit(1);
    }
 }
 fclose(fp);
}

void make_map(power, ltg)       /* this function calculates the */
mask FAR power[HAN_SIZE];   /* global masking threshold     */
g_thres FAR *ltg;
{
 int i,j;

 for(i=1;i<sub_size;i++) for(j=ltg[i-1].line;j<=ltg[i].line;j++)
    power[j].map = i;
}

double add_db(a,b)
double a,b;
{
 a = pow(10.0,a/10.0);
 b = pow(10.0,b/10.0);
 return 10 * log10(a+b);
}

/*****************************************************************/
/*
/*        Fast Fourier transform of the input samples.
/*
/*****************************************************************/

void II_f_f_t(sample, power)      /* this function calculates an */
double FAR sample[FFT_SIZE];  /* FFT analysis for the freq.  */
mask FAR power[HAN_SIZE];     /* domain                      */
{
 int i,j,k,L,l=0;
 int ip, le, le1;
 double t_r, t_i, u_r, u_i;
 static int M, MM1, init = 0, N;
 double *x_r, *x_i, *energy;
```

```
 static int *rev;
 static double *w_r, *w_i;

 x_r = (double *) mem_alloc(sizeof(DFFT), "x_r");
 x_i = (double *) mem_alloc(sizeof(DFFT), "x_i");
 energy = (double *) mem_alloc(sizeof(DFFT), "energy");
 for(i=0;i<FFT_SIZE;i++) x_r[i] = x_i[i] = energy[i] = 0;
 if(!init){
    rev = (int *) mem_alloc(sizeof(IFFT), "rev");
    w_r = (double *) mem_alloc(sizeof(D10), "w_r");
    w_i = (double *) mem_alloc(sizeof(D10), "w_i");
    M = 10;
    MM1 = 9;
    N = FFT_SIZE;
    for(L=0;L<M;L++){
       le = 1 << (M-L);
       le1 = le >> 1;
       w_r[L] = cos(PI/le1);
       w_i[L] = -sin(PI/le1);
    }
    for(i=0;i<FFT_SIZE;rev[i] = l,i++) for(j=0,l=0;j<10;j++){
       k=(i>>j) & 1;
       l |= (k<<9-j);
    }
    init = 1;
 }
 memcpy( (char *) x_r, (char *) sample, sizeof(double) * FFT_SIZE);
 for(L=0;L<MM1;L++){
    le = 1 << (M-L);
    le1 = le >> 1;
    u_r = 1;
    u_i = 0;
    for(j=0;j<le1;j++){
       for(i=j;i<N;i+=le){
          ip = i + le1;
          t_r = x_r[i] + x_r[ip];
          t_i = x_i[i] + x_i[ip];
          x_r[ip] = x_r[i] - x_r[ip];
          x_i[ip] = x_i[i] - x_i[ip];
          x_r[i] = t_r;
          x_i[i] = t_i;
          t_r = x_r[ip];
          x_r[ip] = x_r[ip] * u_r - x_i[ip] * u_i;
          x_i[ip] = x_i[ip] * u_r + t_r * u_i;
       }
       t_r = u_r;
       u_r = u_r * w_r[L] - u_i * w_i[L];
       u_i = u_i * w_r[L] + t_r * w_i[L];
    }
 }
 for(i=0;i<N;i+=2){
    ip = i + 1;
    t_r = x_r[i] + x_r[ip];
    t_i = x_i[i] + x_i[ip];
    x_r[ip] = x_r[i] - x_r[ip];
    x_i[ip] = x_i[i] - x_i[ip];
    x_r[i] = t_r;
    x_i[i] = t_i;
    energy[i] = x_r[i] * x_r[i] + x_i[i] * x_i[i];
 }
 for(i=0;i<FFT_SIZE;i++) if(i<rev[i]){
    t_r = energy[i];
    energy[i] = energy[rev[i]];
    energy[rev[i]] = t_r;
 }
 for(i=0;i<HAN_SIZE;i++){    /* calculate power density spectrum */
    if (energy[i] < 1E-20) energy[i] = 1E-20;
    power[i].x = 10 * log10(energy[i]) + POWERNORM;
    power[i].next = STOP;
    power[i].type = FALSE;
 }
 mem_free((void **) &x_r);
 mem_free((void **) &x_i);
 mem_free((void **) &energy);
}
```

```
/*****************************************************************/
/*
/*         Window the incoming audio signal.
/*
/*****************************************************************/

void II_hann_win(sample)          /* this function calculates a  */
double FAR sample[FFT_SIZE];  /* Hann window for PCM (input) */
{                                 /* samples for a 1024-pt. FFT  */
 register int i;
 register double sqrt_8_over_3;
 static int init = 0;
 static double FAR *window;

 if(!init){  /* calculate window function for the Fourier transform */
    window = (double FAR *) mem_alloc(sizeof(DFFT), "window");
    sqrt_8_over_3 = pow(8.0/3.0, 0.5);
    for(i=0;i<FFT_SIZE;i++){
       /* Hann window formula */
       window[i]=sqrt_8_over_3*0.5*(1-cos(2.0*PI*i/(FFT_SIZE)))/FFT_SIZE;
    }
    init = 1;
 }
 for(i=0;i<FFT_SIZE;i++) sample[i] *= window[i];
}

/******************************************************************/
/*
/*        This function finds the maximum spectral component in each
/* subband and return them to the encoder for time-domain threshold
/* determination.
/*
/******************************************************************/
#ifndef LONDON
void II_pick_max(power, spike)
double FAR spike[SBLIMIT];
mask FAR power[HAN_SIZE];
{
 double max;
 int i,j;

 for(i=0;i<HAN_SIZE;spike[i>>4] = max, i+=16)      /* calculate the      */
 for(j=0, max = DBMIN;j<16;j++)                    /* maximum spectral   */
    max = (max>power[i+j].x) ? max : power[i+j].x; /* component in each  */
}                                                  /* subband from bound */
                                                   /* 4-16               */
#else
void II_pick_max(power, spike)
double FAR spike[SBLIMIT];
mask FAR power[HAN_SIZE];
{
 double sum;
 int i,j;

 for(i=0;i<HAN_SIZE;spike[i>>4] = 10.0*log10(sum), i+=16)
                                                   /* calculate the      */
 for(j=0, sum = pow(10.0,0.1*DBMIN);j<16;j++)      /* sum of spectral   */
   sum += pow(10.0,0.1*power[i+j].x);              /* component in each  */
}                                                  /* subband from bound */
                                                   /* 4-16               */
#endif

/*****************************************************************/
/*
/*        This function labels the tonal component in the power
/* spectrum.
/*
/*****************************************************************/

void II_tonal_label(power, tone)  /* this function extracts (tonal) */
mask FAR power[HAN_SIZE];     /* sinusoidals from the spectrum  */
int *tone;
{
 int i,j, last = LAST, first, run, last_but_one = LAST; /* dpwe */
 double max;
```

```
 *tone = LAST;
 for(i=2;i<HAN_SIZE-12;i++){
    if(power[i].x>power[i-1].x && power[i].x>=power[i+1].x){
       power[i].type = TONE;
       power[i].next = LAST;
       if(last != LAST) power[last].next = i;
       else first = *tone = i;
       last = i;
    }
 }
 last = LAST;
 first = *tone;
 *tone = LAST;
 while(first != LAST){               /* the conditions for the tonal          */
    if(first<3 || first>500) run = 0;/* otherwise k+/-j will be out of bounds */
    else if(first<63) run = 2;       /* components in layer II, which         */
    else if(first<127) run = 3;      /* are the boundaries for calc.          */
    else if(first<255) run = 6;      /* the tonal components                  */
    else run = 12;
    max = power[first].x - 7;        /* after calculation of tonal   */
    for(j=2;j<=run;j++)              /* components, set to local max */
       if(max < power[first-j].x || max < power[first+j].x){
          power[first].type = FALSE;
          break;
       }
    if(power[first].type == TONE){   /* extract tonal components */
       int help=first;
       if(*tone==LAST) *tone = first;
       while((power[help].next!=LAST)&&(power[help].next-first)<=run)
          help=power[help].next;
       help=power[help].next;
       power[first].next=help;
       if((first-last)<=run){
          if(last_but_one != LAST) power[last_but_one].next=first;
       }
       if(first>1 && first<500){     /* calculate the sum of the */
          double tmp;                /* powers of the components */
          tmp = add_db(power[first-1].x, power[first+1].x);
          power[first].x = add_db(power[first].x, tmp);
       }
       for(j=1;j<=run;j++){
          power[first-j].x = power[first+j].x = DBMIN;
          power[first-j].next = power[first+j].next = STOP;
          power[first-j].type = power[first+j].type = FALSE;
       }
       last_but_one=last;
       last = first;
       first = power[first].next;
    }
    else {
       int ll;
       if(last == LAST); /* *tone = power[first].next; dpwe */
       else power[last].next = power[first].next;
       ll = first;
       first = power[first].next;
       power[ll].next = STOP;
    }
 }
}

/****************************************************************/
/*
/*        This function groups all the remaining non-tonal
/* spectral lines into critical band where they are replaced by
/* one single line.
/*
/****************************************************************/

void noise_label(power, noise, ltg)
g_thres FAR *ltg;
mask FAR *power;
int *noise;
{
 int i,j, centre, last = LAST;
 double index, weight, sum;
                              /* calculate the remaining spectral */
```

```
 for(i=0;i<crit_band-1;i++){  /* lines for non-tonal components   */
     for(j=cbound[i],weight = 0.0,sum = DBMIN;j<cbound[i+1];j++){
        if(power[j].type != TONE){
           if(power[j].x != DBMIN){
              sum = add_db(power[j].x,sum);
/* the line below and others under the "MAKE_SENSE" condition are an alternate
   interpretation of "geometric mean". This approach may make more sense but
   it has not been tested with hardware. */
#ifdef MAKE_SENSE
/* weight += pow(10.0, power[j].x/10.0) * (ltg[power[j].map].bark-i);
   bad code [SS] 21-1-93
 */
    weight += pow(10.0,power[j].x/10.0) * (double) (j-cbound[i]) /
     (double) (cbound[i+1]-cbound[i]);  /* correction */
#endif
              power[j].x = DBMIN;
           }
        }   /*  check to see if the spectral line is low dB, and if  */
     }      /* so replace the center of the critical band, which is */
            /* the center freq. of the noise component              */

#ifdef MAKE_SENSE
     if(sum <= DBMIN)  centre = (cbound[i+1]+cbound[i]) /2;
     else {
        index = weight/pow(10.0,sum/10.0);
        centre = cbound[i] + (int) (index * (double) (cbound[i+1]-cbound[i]) );
     }
#else
     index = (double)( ((double)cbound[i]) * ((double)(cbound[i+1]-1)) );
     centre = (int)(pow(index,0.5)+0.5);
#endif

    /* locate next non-tonal component until finished; */
    /* add to list of non-tonal components             */
#ifdef MI_OPTION
     /* Masahiro Iwadare's fix for infinite looping problem? */
     if(power[centre].type == TONE)
       if (power[centre+1].type == TONE) centre++; else centre--;
#else
     /* Mike Li's fix for infinite looping problem */
     if(power[centre].type == FALSE) centre++;

     if(power[centre].type == NOISE){
       if(power[centre].x >= ltg[power[i].map].hear){
         if(sum >= ltg[power[i].map].hear) sum = add_db(power[j].x,sum);
         else
         sum = power[centre].x;
       }
     }
#endif
     if(last == LAST) *noise = centre;
     else {
        power[centre].next = LAST;
        power[last].next = centre;
     }
     power[centre].x = sum;
     power[centre].type = NOISE;
     last = centre;
 }
}

/****************************************************************/
/*
/*        This function reduces the number of noise and tonal
/* component for further threshold analysis.
/*
/****************************************************************/

void subsampling(power, ltg, tone, noise)
mask FAR power[HAN_SIZE];
g_thres FAR *ltg;
int *tone, *noise;
{
 int i, old;

 i = *tone; old = STOP;    /* calculate tonal components for */
```

```
 while(i!=LAST){           /* reduction of spectral lines    */
    if(power[i].x < ltg[power[i].map].hear){
       power[i].type = FALSE;
       power[i].x = DBMIN;
       if(old == STOP) *tone = power[i].next;
       else power[old].next = power[i].next;
    }
    else old = i;
    i = power[i].next;
 }
 i = *noise; old = STOP;    /* calculate non-tonal components for */
 while(i!=LAST){            /* reduction of spectral lines        */
    if(power[i].x < ltg[power[i].map].hear){
       power[i].type = FALSE;
       power[i].x = DBMIN;
       if(old == STOP) *noise = power[i].next;
       else power[old].next = power[i].next;
    }
    else old = i;
    i = power[i].next;
 }
 i = *tone; old = STOP;
 while(i != LAST){                              /* if more than one */
    if(power[i].next == LAST)break;             /* tonal component  */
    if(ltg[power[power[i].next].map].bark -     /* is less than .5  */
       ltg[power[i].map].bark < 0.5) {          /* bark, take the   */
       if(power[power[i].next].x > power[i].x ){/* maximum          */
          if(old == STOP) *tone = power[i].next;
          else power[old].next = power[i].next;
          power[i].type = FALSE;
          power[i].x = DBMIN;
          i = power[i].next;
       }
       else {
          power[power[i].next].type = FALSE;
          power[power[i].next].x = DBMIN;
          power[i].next = power[power[i].next].next;
          old = i;
       }
    }
    else {
      old = i;
      i = power[i].next;
    }
 }
}

/****************************************************************/
/*
/*        This function calculates the individual threshold and
/* sum with the quiet threshold to find the global threshold.
/*
/****************************************************************/

void threshold(power, ltg, tone, noise, bit_rate)
mask FAR power[HAN_SIZE];
g_thres FAR *ltg;
int *tone, *noise, bit_rate;
{
 int k, t;
 double dz, tmps, vf;

 for(k=1;k<sub_size;k++){
    ltg[k].x = DBMIN;
    t = *tone;          /* calculate individual masking threshold for */
    while(t != LAST){   /* components in order to find the global     */
       if(ltg[k].bark-ltg[power[t].map].bark >= -3.0 && /*threshold (LTG)*/
          ltg[k].bark-ltg[power[t].map].bark <8.0){
          dz = ltg[k].bark-ltg[power[t].map].bark; /* distance of bark value*/
          tmps = -1.525-0.275*ltg[power[t].map].bark - 4.5 + power[t].x;
             /* masking function for lower & upper slopes */
          if(-3<=dz && dz<-1) vf = 17*(dz+1)-(0.4*power[t].x +6);
          else if(-1<=dz && dz<0) vf = (0.4 *power[t].x + 6) * dz;
          else if(0<=dz && dz<1) vf = (-17*dz);
          else if(1<=dz && dz<8) vf = -(dz-1) * (17-0.15 *power[t].x) - 17;
          tmps += vf;
```

```
         ltg[k].x = add_db(ltg[k].x, tmps);
      }
      t = power[t].next;
   }

   t = *noise;        /* calculate individual masking threshold  */
   while(t != LAST){  /* for non-tonal components to find LTG    */
      if(ltg[k].bark-ltg[power[t].map].bark >= -3.0 &&
         ltg[k].bark-ltg[power[t].map].bark <8.0){
         dz = ltg[k].bark-ltg[power[t].map].bark; /* distance of bark value */
         tmps = -1.525-0.175*ltg[power[t].map].bark -0.5 + power[t].x;
            /* masking function for lower & upper slopes */
         if(-3<=dz && dz<-1) vf = 17*(dz+1)-(0.4*power[t].x +6);
         else if(-1<=dz && dz<0) vf = (0.4 *power[t].x + 6) * dz;
         else if(0<=dz && dz<1) vf = (-17*dz);
         else if(1<=dz && dz<8) vf = -(dz-1) * (17-0.15 *power[t].x) - 17;
         tmps += vf;
         ltg[k].x = add_db(ltg[k].x, tmps);
      }
      t = power[t].next;
   }
   if(bit_rate<96)ltg[k].x = add_db(ltg[k].hear, ltg[k].x);
   else ltg[k].x = add_db(ltg[k].hear-12.0, ltg[k].x);
 }
}

/****************************************************************/
/*
/*        This function finds the minimum masking threshold and
/* return the value to the encoder.
/*
/****************************************************************/

void II_minimum_mask(ltg,ltmin,sblimit)
g_thres FAR *ltg;
double FAR ltmin[SBLIMIT];
int sblimit;
{
 double min;
 int i,j;

 j=1;
 for(i=0;i<sblimit;i++)
    if(j>=sub_size-1)                   /* check subband limit, and       */
       ltmin[i] = ltg[sub_size-1].hear; /* calculate the minimum masking  */
    else {                              /* level of LTMIN for each subband*/
       min = ltg[j].x;
       while(ltg[j].line>>4 == i && j < sub_size){
       if(min>ltg[j].x)  min = ltg[j].x;
       j++;
    }
    ltmin[i] = min;
 }
}

/*****************************************************************/
/*
/*        This procedure is called in musicin to pick out the
/* smaller of the scalefactor or threshold.
/*
/*****************************************************************/

void II_smr(ltmin, spike, scale, sblimit)
double FAR spike[SBLIMIT], scale[SBLIMIT], ltmin[SBLIMIT];
int sblimit;
{
 int i;
 double max;

 for(i=0;i<sblimit;i++){                     /* determine the signal   */
    max = 20 * log10(scale[i] * 32768) - 10; /* level for each subband */
    if(spike[i]>max) max = spike[i];         /* for the maximum scale  */
    max -= ltmin[i];                         /* factors                */
    ltmin[i] = max;
 }
}
```

```
/*****************************************************************/
/*
/*        This procedure calls all the necessary functions to
/* complete the psychoacoustic analysis.
/*
/*****************************************************************/

void II_Psycho_One(buffer, scale, ltmin, fr_ps)
short FAR buffer[2][1152];
double FAR scale[2][SBLIMIT], ltmin[2][SBLIMIT];
frame_params *fr_ps;
{
 layer *info = fr_ps->header;
 int   stereo = fr_ps->stereo;
 int   sblimit = fr_ps->sblimit;
 int k,i, tone=0, noise=0;
 static char init = 0;
 static int off[2] = {256,256};
 double *sample;
 DSBL *spike;
 static D1408 *fft_buf;
 static mask_ptr FAR power;
 static g_ptr FAR ltg;

 sample = (double *) mem_alloc(sizeof(DFFT), "sample");
 spike = (DSBL *) mem_alloc(sizeof(D2SBL), "spike");
     /* call functions for critical boundaries, freq. */
 if(!init){  /* bands, bark values, and mapping */
    fft_buf = (D1408 *) mem_alloc((long) sizeof(D1408) * 2, "fft_buf");
    power = (mask_ptr FAR ) mem_alloc(sizeof(mask) * HAN_SIZE, "power");
    read_cbound(info->lay,info->sampling_frequency);
    read_freq_band(&ltg,info->lay,info->sampling_frequency);
    make_map(power,ltg);
    for (i=0;i<1408;i++) fft_buf[0][i] = fft_buf[1][i] = 0;
    init = 1;
 }
 for(k=0;k<stereo;k++){  /* check pcm input for 3 blocks of 384 samples */
    for(i=0;i<1152;i++) fft_buf[k][(i+off[k])%1408]= (double)buffer[k][i]/SCALE;
    for(i=0;i<FFT_SIZE;i++) sample[i] = fft_buf[k][(i+1216+off[k])%1408];
    off[k] += 1152;
    off[k] %= 1408;
                            /* call functions for windowing PCM samples,*/
    II_hann_win(sample);    /* location of spectral components in each  */
    for(i=0;i<HAN_SIZE;i++) power[i].x = DBMIN;  /*subband with labeling*/
    II_f_f_t(sample, power);                     /*locate remaining non-*/
    II_pick_max(power, &spike[k][0]);            /*tonal sinusoidals,   */
    II_tonal_label(power, &tone);                /*reduce noise & tonal */
    noise_label(power, &noise, ltg);             /*components, find     */
    subsampling(power, ltg, &tone, &noise);      /*global & minimal     */
    threshold(power, ltg, &tone, &noise,         /*threshold, and sgnl- */
      bitrate[info->lay-1][info->bitrate_index]/stereo); /*to-mask ratio*/
    II_minimum_mask(ltg, &ltmin[k][0], sblimit);
    II_smr(&ltmin[k][0], &spike[k][0], &scale[k][0], sblimit);
 }
 mem_free((void **) &sample);
 mem_free((void **) &spike);
}

/*******************************************************************/
/*
/*        This module implements the psychoacoustic model I for the
/* MPEG encoder layer I. It uses simplified tonal and noise masking
/* threshold analysis to generate SMR for the encoder bit allocation
/* routine.
/*
/*******************************************************************/

/*****************************************************************/
/*
/*        Fast Fourier transform of the input samples.
/*
/*****************************************************************/

void I_f_f_t(sample, power)         /* this function calculates */
double FAR sample[FFT_SIZE/2];  /* an FFT analysis for the  */
```

```
mask FAR power[HAN_SIZE/2];     /* freq. domain             */
{
 int i,j,k,L,l=0;
 int ip, le, le1;
 double t_r, t_i, u_r, u_i;
 static int M, MM1, init = 0, N;
 double *x_r, *x_i, *energy;
 static int *rev;
 static double *w_r, *w_i;

 x_r = (double *) mem_alloc(sizeof(DFFT2), "x_r");
 x_i = (double *) mem_alloc(sizeof(DFFT2), "x_i");
 energy = (double *) mem_alloc(sizeof(DFFT2), "energy");
 for(i=0;i<FFT_SIZE/2;i++) x_r[i] = x_i[i] = energy[i] = 0;
 if(!init){
    rev = (int *) mem_alloc(sizeof(IFFT2), "rev");
    w_r = (double *) mem_alloc(sizeof(D9), "w_r");
    w_i = (double *) mem_alloc(sizeof(D9), "w_i");
    M = 9;
    MM1 = 8;
    N = FFT_SIZE/2;
    for(L=0;L<M;L++){
       le = 1 << (M-L);
       le1 = le >> 1;
       w_r[L] = cos(PI/le1);
       w_i[L] = -sin(PI/le1);
    }
    for(i=0;i<FFT_SIZE/2;rev[i] = l,i++) for(j=0,l=0;j<9;j++){
       k=(i>>j) & 1;
       l |= (k<<8-j);
    }
    init = 1;
 }
 memcpy( (char *) x_r, (char *) sample, sizeof(double) * FFT_SIZE/2);
 for(L=0;L<MM1;L++){
    le = 1 << (M-L);
    le1 = le >> 1;
    u_r = 1;
    u_i = 0;
    for(j=0;j<le1;j++){
       for(i=j;i<N;i+=le){
          ip = i + le1;
          t_r = x_r[i] + x_r[ip];
          t_i = x_i[i] + x_i[ip];
          x_r[ip] = x_r[i] - x_r[ip];
          x_i[ip] = x_i[i] - x_i[ip];
          x_r[i] = t_r;
          x_i[i] = t_i;
          t_r = x_r[ip];
          x_r[ip] = x_r[ip] * u_r - x_i[ip] * u_i;
          x_i[ip] = x_i[ip] * u_r + t_r * u_i;
       }
       t_r = u_r;
       u_r = u_r * w_r[L] - u_i * w_i[L];
       u_i = u_i * w_r[L] + t_r * w_i[L];
    }
 }
 for(i=0;i<N;i+=2){
    ip = i + 1;
    t_r = x_r[i] + x_r[ip];
    t_i = x_i[i] + x_i[ip];
    x_r[ip] = x_r[i] - x_r[ip];
    x_i[ip] = x_i[i] - x_i[ip];
    x_r[i] = t_r;
    x_i[i] = t_i;
    energy[i] = x_r[i] * x_r[i] + x_i[i] * x_i[i];
 }
 for(i=0;i<FFT_SIZE/2;i++) if(i<rev[i]){
    t_r = energy[i];
    energy[i] = energy[rev[i]];
    energy[rev[i]] = t_r;
 }
 for(i=0;i<HAN_SIZE/2;i++){                      /* calculate power  */
    if(energy[i] < 1E-20) energy[i] = 1E-20;    /* density spectrum */
       power[i].x = 10 * log10(energy[i]) + POWERNORM;
       power[i].next = STOP;
```

```
       power[i].type = FALSE;
 }
 mem_free((void **) &x_r);
 mem_free((void **) &x_i);
 mem_free((void **) &energy);
}

/*****************************************************************/
/*
/*          Window the incoming audio signal.
/*
/*****************************************************************/

void I_hann_win(sample)             /* this function calculates a  */
double FAR sample[FFT_SIZE/2];  /* Hann window for PCM (input) */
{                                   /* samples for a 512-pt. FFT   */
 register int i;
 register double sqrt_8_over_3;
 static int init = 0;
 static double FAR *window;

 if(!init){  /* calculate window function for the Fourier transform */
    window = (double FAR *) mem_alloc(sizeof(DFFT2), "window");
    sqrt_8_over_3 = pow(8.0/3.0, 0.5);
    for(i=0;i<FFT_SIZE/2;i++){
      /* Hann window formula */
      window[i]=sqrt_8_over_3*0.5*(1-cos(2.0*PI*i/(FFT_SIZE/2)))/(FFT_SIZE/2);
    }
    init = 1;
 }
 for(i=0;i<FFT_SIZE/2;i++) sample[i] *= window[i];
}

/*******************************************************************/
/*
/*         This function finds the maximum spectral component in each
/* subband and return them to the encoder for time-domain threshold
/* determination.
/*
/*******************************************************************/
#ifndef LONDON
void I_pick_max(power, spike)
double FAR spike[SBLIMIT];
mask FAR power[HAN_SIZE/2];
{
 double max;
 int i,j;

 /* calculate the spectral component in each subband */
 for(i=0;i<HAN_SIZE/2;spike[i>>3] = max, i+=8)
    for(j=0, max = DBMIN;j<8;j++) max = (max>power[i+j].x) ? max : power[i+j].x;
}
#else
void I_pick_max(power, spike)
double FAR spike[SBLIMIT];
mask FAR power[HAN_SIZE];
{
 double sum;
 int i,j;

 for(i=0;i<HAN_SIZE/2;spike[i>>3] = 10.0*log10(sum), i+=8)
                                                   /* calculate the      */
 for(j=0, sum = pow(10.0,0.1*DBMIN);j<8;j++)       /* sum of spectral   */
   sum += pow(10.0,0.1*power[i+j].x);              /* component in each  */
}                                                  /* subband from bound */
#endif
/*****************************************************************/
/*
/*         This function labels the tonal component in the power
/* spectrum.
/*
/*****************************************************************/

void I_tonal_label(power, tone)     /* this function extracts   */
mask FAR power[HAN_SIZE/2];     /* (tonal) sinusoidals from */
int *tone;                          /* the spectrum             */
```

```
{
 int i,j, last = LAST, first, run;
 double max;
 int last_but_one= LAST;

 *tone = LAST;
 for(i=2;i<HAN_SIZE/2-6;i++){
    if(power[i].x>power[i-1].x && power[i].x>=power[i+1].x){
       power[i].type = TONE;
       power[i].next = LAST;
       if(last != LAST) power[last].next = i;
       else first = *tone = i;
       last = i;
    }
 }
 last = LAST;
 first = *tone;
 *tone = LAST;
 while(first != LAST){                 /* conditions for the tonal     */
    if(first<3 || first>250) run = 0; /* otherwise k+/-j will be out of bounds*/
    else if(first<63) run = 2;        /* components in layer I, which */
    else if(first<127) run = 3;       /* are the boundaries for calc.   */
    else run = 6;                     /* the tonal components          */
    max = power[first].x - 7;
    for(j=2;j<=run;j++)  /* after calc. of tonal components, set to loc.*/
       if(max < power[first-j].x || max < power[first+j].x){   /* max   */
          power[first].type = FALSE;
          break;
       }
    if(power[first].type == TONE){    /* extract tonal components */
       int help=first;
       if(*tone == LAST) *tone = first;
       while((power[help].next!=LAST)&&(power[help].next-first)<=run)
          help=power[help].next;
       help=power[help].next;
       power[first].next=help;
       if((first-last)<=run){
          if(last_but_one != LAST) power[last_but_one].next=first;
       }
       if(first>1 && first<255){     /* calculate the sum of the */
          double tmp;                /* powers of the components */
          tmp = add_db(power[first-1].x, power[first+1].x);
          power[first].x = add_db(power[first].x, tmp);
       }
       for(j=1;j<=run;j++){
          power[first-j].x = power[first+j].x = DBMIN;
          power[first-j].next = power[first+j].next = STOP; /*dpwe: 2nd was .x*/
          power[first-j].type = power[first+j].type = FALSE;
       }
       last_but_one=last;
       last = first;
       first = power[first].next;
    }
    else {
       int ll;
       if(last == LAST) ; /* *tone = power[first].next; dpwe */
       else power[last].next = power[first].next;
       ll = first;
       first = power[first].next;
       power[ll].next = STOP;
    }
 }
}

/*****************************************************************/
/*
/*        This function finds the minimum masking threshold and
/* return the value to the encoder.
/*
/*****************************************************************/

void I_minimum_mask(ltg,ltmin)
g_thres FAR *ltg;
double FAR ltmin[SBLIMIT];
{
 double min;
```

```
 int i,j;

 j=1;
 for(i=0;i<SBLIMIT;i++)
    if(j>=sub_size-1)                   /* check subband limit, and       */
       ltmin[i] = ltg[sub_size-1].hear; /* calculate the minimum masking  */
    else {                              /* level of LTMIN for each subband*/
       min = ltg[j].x;
       while(ltg[j].line>>3 == i && j < sub_size){
          if (min>ltg[j].x)  min = ltg[j].x;
          j++;
       }
       ltmin[i] = min;
    }
}

/*****************************************************************/
/*
/*        This procedure is called in musicin to pick out the
/* smaller of the scalefactor or threshold.
/*
/*****************************************************************/

void I_smr(ltmin, spike, scale)
double FAR spike[SBLIMIT], scale[SBLIMIT], ltmin[SBLIMIT];
{
 int i;
 double max;

 for(i=0;i<SBLIMIT;i++){                      /* determine the signal   */
    max = 20 * log10(scale[i] * 32768) - 10;  /* level for each subband */
    if(spike[i]>max) max = spike[i];          /* for the scalefactor    */
    max -= ltmin[i];
    ltmin[i] = max;
 }
}

/****************************************************************/
/*
/*        This procedure calls all the necessary functions to
/* complete the psychoacoustic analysis.
/*
/****************************************************************/

void I_Psycho_One(buffer, scale, ltmin, fr_ps)
short FAR buffer[2][1152];
double FAR scale[2][SBLIMIT], ltmin[2][SBLIMIT];
frame_params *fr_ps;
{
 int stereo = fr_ps->stereo;
 the_layer info = fr_ps->header;
 int k,i, tone=0, noise=0;
 static char init = 0;
 static int off[2] = {256,256};
 double *sample;
 DSBL *spike;
 static D640 *fft_buf;
 static mask_ptr FAR power;
 static g_ptr FAR ltg;

 sample = (double *) mem_alloc(sizeof(DFFT2), "sample");
 spike = (DSBL *) mem_alloc(sizeof(D2SBL), "spike");
            /* call functions for critical boundaries, freq. */
 if(!init){ /* bands, bark values, and mapping              */
    fft_buf = (D640 *) mem_alloc(sizeof(D640) * 2, "fft_buf");
    power = (mask_ptr FAR ) mem_alloc(sizeof(mask) * HAN_SIZE/2, "power");
    read_cbound(info->lay,info->sampling_frequency);
    read_freq_band(&ltg,info->lay,info->sampling_frequency);
    make_map(power,ltg);
    for(i=0;i<640;i++) fft_buf[0][i] = fft_buf[1][i] = 0;
    init = 1;
 }
 for(k=0;k<stereo;k++){    /* check PCM input for a block of */
    for(i=0;i<384;i++)     /* 384 samples for a 512-pt. FFT  */
       fft_buf[k][(i+off[k])%640]= (double) buffer[k][i]/SCALE;
    for(i=0;i<FFT_SIZE/2;i++)
```

```
      sample[i] = fft_buf[k][(i+448+off[k])%640];
    off[k] += 384;
    off[k] %= 640;
                        /* call functions for windowing PCM samples,   */
    I_hann_win(sample); /* location of spectral components in each     */
    for(i=0;i<HAN_SIZE/2;i++) power[i].x = DBMIN;   /* subband with    */
    I_f_f_t(sample, power);              /* labeling, locate remaining */
    I_pick_max(power, &spike[k][0]);     /* non-tonal sinusoidals,     */
    I_tonal_label(power, &tone);         /* reduce noise & tonal com., */
    noise_label(power, &noise, ltg);     /* find global & minimal      */
    subsampling(power, ltg, &tone, &noise);  /* threshold, and sgnl-   */
    threshold(power, ltg, &tone, &noise,     /* to-mask ratio          */
      bitrate[info->lay-1][info->bitrate_index]/stereo);
    I_minimum_mask(ltg, &ltmin[k][0]);
    I_smr(&ltmin[k][0], &spike[k][0], &scale[k][0]);
 }
 mem_free((void **) &sample);
 mem_free((void **) &spike);
}
```

Annex D

Bibliography

[1] Arun N. Netravali & Barry G. Haskell *Digital Pictures, representation and compression* Plenum Press, 1988.

[2] Didier Le Gall *MPEG: A Video Compression Standard for Multimedia Applications* Trans ACM, April 1991.

[3] C Loeffler, A Ligtenberg, G S Moschytz *Practical fast 1-D DCT algorithms with 11 multiplications* Proceedings IEEE ICASSP-89, Vol. 2, pp 988-991, Feb. 1989.

[4] IEC Standard Publication 461, Second edition 1986 *Time and control code for video tape recorders.*

[5] CCITT Recommendation H.261 *Codec for audiovisual services at px64 kbit/s* Geneva, 1990.

[6] ISO/IEC DIS 10918-1 *Digital compression and coding of continuous-tone still images - Part 1: Requirements and guidelines.*

[7] E Viscito and C Gonzales *A Video Compression Algorithm with Adaptive Bit Allocation and Quantization*, Proc SPIE Visual Communications and Image Proc '91 Boston MA November 10-15 Vol 1605 205, 1991.

[8] A Puri and R Aravind *Motion Compensated Video Coding with Adaptive Perceptual Quantization,* IEEE Trans on Circuits and Systems for Video Technology, Vol 1 pp 351 Dec 1991.

[9] C. Fogg, P. Au, S. Eckart, T. Hanamura, K. Oosa, B. Quandt, H. Watanabe *ISO/IEC Software Implementation of MPEG-1 Video*, IS&T/SPIE Symposium on Electronic Imaging: Science and Technology -- Digital Video Compression on Personal Computers: Algorithms and Technologies, San Jose, CA, SPIE Vol. 2187 No. 25, Feb. 6-10, 1994..

Annex E
(informative)

List of patent holders

The user's attention is called to the possibility that - for some of the processes specified in this part of ISO/IEC 11172 - compliance with this International Standard may require use of an invention covered by patent rights.

By publication of this part of ISO/IEC 11172, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. However, each company listed in this annex has filed with the Information Technology Task Force (ITTF) a statement of willingness to grant a license under such rights that they hold on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license.

Information regarding such patents can be obtained from :

AT&T
32 Avenue of the Americas
New York
NY 10013-2412
USA

Aware
1 Memorial Drive
Cambridge
02142 Massachusetts
USA

Bellcore
290 W Mount Pleasant Avenue
Livingston
NJ 07039
USA

The British Broadcasting Corporation
Broadcasting House
London
W1A 1AA
United Kingdom

British Telecommunications plc
Intellectual Property Unit
13th Floor
151 Gower Street
London
WC1E 6BA
United Kingdom

CCETT
4 Rue du Clos-Courtel
BP 59

F-35512
Cesson-Sevigne Cedex
France

CNET
38-40 Rue du General Leclerc
F-92131 Issy-les-Moulineaux
France

Compression Labs, Incorporated
2860 Junction Avenue
San Jose
CA 95134
USA

CSELT
Via G Reiss Romoli 274
I-10148 Torino
Italy

CompuSonics Corporation
PO Box 61017
Palo Alto
CA 94306
USA

Daimler Benz AG
PO Box 800 230
Epplestrasse 225
D-7000 Stuttgart 80
Germany

Dornier Gmbh
An der Bundesstrasse 31
D-7990 Friedrichshafen1
Germany

Fraunhofer Gesselschaft zur Foerderung der Angerwandten Forschung e.V.
Leonrodstrasse 54
8000 Muenchen 19
Germany

Hitachi Ltd
6 Kanda-Surugadai 4 chome
Chiyoda-ku
Tokyo 101
Japan

Institut für Rundfunktechnik Gmbh
Florianmühlstraße 60
8000 München 45
Germany

International Business Machines Corporation
Armonk

New York 10504
USA

KDD Corporation
2-3-2 Nishishinjuku
Shinjuku-ku
Tokyo
Japan

Licentia Patent-Verwaltungs-Gmbh
Theodor-Stern-Kai &
D-6000 Frankfurt 70
Germany

Massachusetts Institute of Technology
20 Ames Street
Cambridge
Massachusetts 02139
USA

Matsushita Electric Industrial Co. Ltd
1006 Oaza-Kadoma
Kadoma
Osaka 571
Japan

Mitsubishi Electric Corporation
2-3 Marunouchi
2-Chome
Chiyoda-Ku
Tokyo
100 Japan

NEC Corporation
7-1 Shiba 5-Chome
Minato-ku
Tokyo
Japan

Nippon Hoso Kyokai
2-2-1 Jin-nan
Shibuya-ku
Tokyo 150-01
Japan

Philips Electronics NV
Groenewoudseweg 1
5621 BA Eindhoven
The Netherlands

Pioneer Electronic Corporation
4-1 Meguro 1-Chome
Meguro-ku
Tokyo 153
Japan

Ricoh Co, Ltd
1-3-6 Nakamagome
Ohta-ku
Tokyo 143
Japan

Sony Corporation
6-7-35 Kitashinagawa
Shinagawa-ku
Tokyo 141
Japan

Symbionics
St John's Innovation Centre
Cowley Road
Cambridge
CB4 4WS
United Kingdom

Telefunken Fernseh und Rundfunk GmbH
Gottinger Chaussee
D-3000 Hannover 91
Germany

Toppan Printing Co, Ltd
1-5-1 Taito
Taito-ku
Tokyo 110
Japan

Toshiba Corporation
1-1 Shibaru 1-Chome
Minato-ku
Tokyo 105
Japan

Victor Company of Japan Ltd
12 Moriya-cho 3 chome
Kanagawa-ku
Yokohama
Kanagawa 221
Japan