

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

5 **ISO/IEC JTC1/SC29/WG11** **N0805**

6  
7 **11 November 1994**

8  
9 **Proposed Draft Technical Report**

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  

---

20 **INFORMATION TECHNOLOGY -**

21 **GENERIC CODING OF MOVING PICTURES AND**  
22 **ASSOCIATED AUDIO**  
23 **Recommendation H.262**

---

24 **ISO/IEC 13818-5**

---

25 **Proposed Draft Technical Report**  

---

26

## **CONTENTS**

## Foreword

The ITU-T (the ITU Telecommunication Standardisation Sector) is a permanent organ of the International Telecommunications Union (ITU). The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to developing telecommunication standards on a world-wide basis.

The World Telecommunication Standardisation Conference, which meets every four years, establishes the program of work arising from the review of existing questions and new questions among other things. The approval of new or revised Recommendations by members of the ITU-T is covered by the procedure laid down in the ITU-T Resolution No. 1 (Helsinki 1993). The proposal for Recommendation is accepted if 70% or more of the replies from members indicate approval.

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.

This specification is a committee draft that is being submitted for approval to the ITU-T, ISO-IEC/JTC1 SC29. It was prepared jointly by SC29/WG11, also known as MPEG (Moving Pictures Expert Group), and the Experts Group for ATM Video Coding in the ITU-T SG15. MPEG was formed in 1988 to establish standards for coding of moving pictures and associated audio for various applications such as digital storage media, distribution and communication. The Experts Group for ATM Video Coding was formed in 1990 to develop video coding standard(s) appropriate for B-ISDN using ATM transport.

In this specification Annex A, Annex B and Annex C contain normative requirements and are an integral part of this specification. Annex D, Annex E, Annex F and Annex G are informative and contain no normative requirements.

## ISO/IEC

This International Standard is published in four Parts.

13818-1 systems	specifies the system coding of the specification. It defines a multiplexed structure for combining audio and video data and means of representing the timing information needed to replay synchronised sequences in real-time.
13818-2 video	specifies the coded representation of video data and the decoding process required to reconstruct pictures.
13818-3 audio	specifies the coded representation of audio data.
13818-4 conformance	specifies the procedures for determining the characteristics of coded bitstreams and for testing compliance with the requirements stated in 13818-1, 13818-2 and 13818-3.
13818-5 technical report	.Software simulation of Parts 1, 2, and 3 of 13818

# **I Introduction**

## **Purpose**

This Part of this specification was developed in response to the growing need for a generic coding method of moving pictures and of associated sound for various applications such as digital storage media, television broadcasting and communication. The use of this specification means that motion video can be manipulated as a form of computer data and can be stored on various storage media, transmitted and received over existing and future networks and distributed on existing and future broadcasting channels.

## **Scope**

This Recommendation | International Standard specifies the coded representation of picture information for digital storage media and digital video communication and specifies the decoding process. The representation supports constant bitrate transmission, variable bitrate transmission, random access, channel hopping, scalable decoding, bitstream editing, as well as special functions such as fast forward playback, fast reverse playback, slow motion, pause and still pictures. This Recommendation | International Standard is forward compatible with ISO/IEC 11172-2 and upward or downward compatible with EDTV, HDTV, SDTV formats.

This Recommendation | International Standard is primarily applicable to digital storage media, video broadcast and communication. The storage media may be directly connected to the decoder, or via communications means such as busses, LANs, or telecommunications links.

## Normative references

The following ITU-T Recommendations and International Standards contain provisions which through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard 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. The TSB (Telecommunication Standardisation Bureau) maintains a list of currently valid ITU-T Recommendations.

Recommendations and reports of the CCIR, 1990

XVIIth Plenary Assembly, Dusseldorf, 1990 Volume XI - Part 1

Broadcasting Service (Television) Rec. 601-2 "Encoding parameters of digital television for studios"

CCIR Volume X and XI Part 3 Recommendation 648: Recording of audio signals.

CCIR Volume X and XI Part 3 Report 955-2: Sound broadcasting by satellite for portable and mobile receivers, including Annex IV Summary description of advanced digital system II.

ISO/IEC 11172 (1993) "Information technology Ñ Coding of moving picture and associated audio for digital storage media at up to about 1,5 Mbit/s"

IEEE Standard Specifications for the Implementations of 8 by 8 Inverse Discrete Cosine Transform, IEEE Std 1180-1990, December 6, 1990.

IEC Publication 908:198, "CD Digital Audio System"

IEC Standard Publication 461 Second edition, 1986 "Time and control code for video tape recorders"

ITU-T Recommendation H.261 (Formerly CCITT Recommendation H.261) ÒCodec for audiovisual services at px64 kbit/s" Geneva, 1990

ISO/IEC 10918-1 | ITU-T Rec. T.81 (JPEG) "Digital compression and coding of continuous-tone still images"

### 3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

1. **ac coefficient [video]:** Any DCT coefficient for which the frequency in one or both dimensions is non-zero.
2. **AC coefficient:** Any DCT coefficient for which the frequency in one or both dimensions is non-zero.
3. **access unit [system]:** A coded representation of a presentation unit. 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.
4. **adaptive bit allocation [audio]:** The assignment of bits to subbands in a time and frequency varying fashion according to a psychoacoustic model.
5. **adaptive multichannel prediction [audio]:** A method of multichannel data reduction exploiting statistical inter-channel dependencies.
6. **adaptive noise allocation [audio]:** The assignment of coding noise to frequency bands in a time and frequency varying fashion according to a psychoacoustic model.
7. **adaptive segmentation [audio]:** A subdivision of the digital representation of an audio signal in variable segments of time.
8. **alias [audio]:** Mirrored signal component resulting from sub-Nyquist sampling.
9. **almost-compliant:** decoder which does not pass all bitstream tests.
10. **analysis filterbank [audio]:** Filterbank in the encoder that transforms a broadband PCM audio signal into a set of subsampled subband samples.
11. **audio access unit [audio]:** For Layers I and II, an audio access unit is defined as the smallest part of the encoded bit stream which can be decoded by itself, where decoded means "fully reconstructed sound". For Layer III, an audio access unit is part of the bit stream that is decodable with the use of previously acquired main information.
12. **audio buffer [audio]:** A buffer in the system target decoder for storage of compressed audio data.
13. **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
14. **B-field picture [video]:** A field structure B-Picture.
15. **B-frame picture [video]:** A frame structure B-Picture.
16. **B-picture; bidirectionally predictive-coded picture [video]:** A picture that is coded using motion compensated prediction from past and/or future reference fields or frames.
17. **backward compatibility:** A newer coding standard is backward compatible with an older coding standard if decoders designed to operate with the older coding standard are able to continue to operate by decoding all or part of a bitstream produced according to the newer coding standard.
18. **backward motion vector [video]:** A motion vector that is used for motion compensation from a reference picture at a later time in display order.
19. **backward motion vector [video]:** A motion vector that is used for motion compensation from a reference frame or reference field at a later time in display order.
20. **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.
21. **bidirectionally predictive-coded picture; B-picture [video]:** A picture that is coded using motion compensated prediction from a past and/or future reference picture.
22. **bitrate:** The rate at which the coded bitstream is delivered from the storage medium to the input of a decoder.
23. **block [video]:** An 8-row by 8-column matrix of samples, or 64 DCT coefficients (source, quantised or dequantised).
24. **block companding [audio]:** Normalising of the digital representation of an audio signal within a certain time period.
25. **bottom field [video]:** One of two fields that comprise a frame. Each line of a bottom field is spatially located immediately below the corresponding line of the top field.
26. **bound [audio]:** The lowest subband in which intensity stereo coding is used.
27. **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.
28. **byte:** Sequence of 8-bits.
29. **center channel [audio]:** An audio presentation channel used to stabilise the central component of the frontal stereo image.

30. **channel [audio]:** A sequence of data representing an audio signal being transported.
31. **channel:** A digital medium that stores or transports a CD 13818 bit stream.
32. **chroma simulcast:** A type of scalability (which is a subset of SNR scalability) where the enhancement layer (s) contain only coded refinement data for the DC coefficients, and all the data for the AC coefficients, of the chrominance components.
33. **chrominance format [video]:** Defines the number of chrominance blocks in a macroblock.
34. **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.
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.
36. **coding parameters:** The set of user-definable parameters that characterise a coded video bitstream. Bitstreams are characterised by coding parameters. Decoders are characterised by the bitstreams that they are capable of decoding.
37. **coded audio bit stream [audio]:** A coded representation of an audio signal as specified in this part of the CD.[video]
38. **coded B-frame [video]:** A B-frame picture or a pair of B-field pictures.
39. **coded frame [video]:** A coded frame is a coded I-frame, a coded P-frame or a coded B-frame.
40. **coded I-frame [video]:** An I-frame picture or a pair of field pictures, where the first one is an I-picture and the second one is an I-picture or a P-picture.
41. **coded order [video]:** The order in which the pictures are stored and decoded. This order is not necessarily the same as the display order.
42. **coded order [video]:** The order in which the pictures are transmitted and decoded. This order is not necessarily the same as the display order.
43. **coded P-frame [video]:** A P-frame picture or a pair of P-field pictures.
44. **coded picture [video]:** A coded picture is made of a picture header, the optionnal extensions immediately following it, and the following picture data. A coded picture may be a frame picture or a field picture.
45. **coded representation:** A data element as represented in its encoded form.
46. **coded video bit stream [video]:** A coded representation of a series of one or more pictures as specified in this CD.
47. **coding parameters [video]:** The set of user-definable parameters that characterise a coded video bit stream. Bit streams are characterised by coding parameters. Decoders are characterised by the bit streams that they are capable of decoding.
48. **component [video]:** A matrix, block or single pel from one of the three matrices (luminance and two chrominance) that make up a picture.
49. **component [video]:** A matrix, block or single sample from one of the three matrices (luminance and two chrominance) that make up a picture.
50. **compression:** Reduction in the number of bits used to represent an item of data.
51. **constant bitrate coded video [video]:** A compressed video bit stream with a constant average bitrate.
52. **constant bitrate:** Operation where the bitrate is constant from start to finish of the coded bitstream.
53. **constrained parameters [video]:** The values of the set of coding parameters defined in 2.4.3.2 of ISO/IEC 11172-2.
54. **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.
55. **CRC:** The Cyclic Redundancy Check to verify the correctness of data.
56. **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.
57. **D-Picture [video]:** A type of picture that shall not be used except in ISO/IEC 11172-2.
58. **data element:** An item of data as represented before encoding and after decoding.
59. **data partitioning:** A method for dividing a bitstream into two separate bitstreams for error resilience purposes. the two bitstreams have to be recombined before decoding.
60. **DC coefficient:** The DCT coefficient for which the frequency is zero in both dimensions.
61. **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.
62. **dc-coefficient [video]:** The DCT coefficient for which the frequency is zero in both dimensions.
63. **DCT coefficient:** The amplitude of a specific cosine basis function.

64. **de-emphasis [audio]**: Filtering applied to an audio signal after storage or transmission to undo a linear distortion due to emphasis.
65. **decoded stream**: The decoded reconstruction of a compressed bit stream.
66. **decoder input buffer [video]**: The first-in first-out (FIFO) buffer specified in the video buffering verifier.
67. **decoder input rate [video]**: The data rate specified in the video buffering verifier and encoded in the coded video bit stream.
68. **decoder**: An embodiment of a decoding process.
69. **decoding (process)**: The process defined in this specification that reads an input coded bitstream and produces decoded pictures or audio samples.
70. **decoding time-stamp; DTS [system]**: A field that may be present in a PES packet header that indicates the time that an access unit is decoded in the system target decoder.
71. **dequantisation [video]**: The process of rescaling the quantised DCT coefficients after their representation in the bit stream has been decoded and before they are presented to the inverse DCT.
72. **dequantisation**: The process of rescaling the quantised DCT coefficients after their representation in the bitstream has been decoded and before they are presented to the inverse DCT.
73. **digital storage media; DSM**: A digital storage or transmission device or system.
74. **discrete cosine transform; DCT**: 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 of this specification.
75. **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.
76. **downmix [audio]**: A matrixing of n channels to obtain less than n channels.
77. **dual channel mode [audio]**: A mode, where two audio channels with independent programme contents (e.g. bilingual) are encoded within one bit stream. The coding process is the same as for the stereo mode.
78. **dynamic crosstalk [audio]**: A method of multichannel data reduction in which stereo-irrelevant signal components are copied to another channel.
79. **dynamic transmission channel switching [audio]**: A method of multichannel data reduction by allocating the most orthogonal signal components to the transmission channels.
80. **editing**: The process by which one or more compressed bit streams are manipulated to produce a new compressed bit stream. Conforming edited bit streams must meet the requirements defined in this CD.
81. **elementary stream; ES [system]**: A generic term for one of the coded video, coded audio or other coded bit streams.
82. **emphasis [audio]**: Filtering applied to an audio signal before storage or transmission to improve the signal-to-noise ratio at high frequencies.
83. **encoder**: An embodiment of an encoding process.
84. **encoding (process)**: A process, not specified in this specification, that reads a stream of input pictures or audio samples and produces a valid coded bitstream as defined in this specification.
85. **entitlement control message; ECM**: Entitlement Control Messages are private conditional access information which specify control words and possibly other, typically stream-specific, scrambling and and/or control parameters.
86. **entitlement management message; EMM**: Entitlement Management Messages are private conditional access information which specify the authorization levels or the services of specific decoders. They may be addressed to single decoders or groups of decoders.
87. **entropy coding**: Variable length lossless coding of the digital representation of a signal to reduce redundancy.
88. **evil bitstreams**: bitstreams orthogonal to reality.
89. **fast reverse playback**: The process of displaying the picture sequence in the reverse of display order faster than real-time.
90. **fast forward playback [video]**: The process of displaying a sequence, or parts of a sequence, of pictures in display-order faster than real-time.
91. **FFT**: Fast Fourier Transformation. A fast algorithm for performing a discrete Fourier transform (an orthogonal transform).
92. **field [video]**: For an interlaced video signal, a field is the assembly of alternate lines of a frame. Therefore an interlaced frame is composed of two fields, a top field and a bottom field.
93. **field period [video]**: The reciprocal of twice the frame rate.
94. **field picture; field structure picture [video]**: A field structure picture is a coded picture with picture\_structure is equal to "Top field" or "Bottom field".
95. **filterbank [audio]**: A set of band-pass filters covering the entire audio frequency range.



96. **fixed segmentation [audio]**: A subdivision of the digital representation of an audio signal into fixed segments of time.
97. **flag**: A variable which can take one of only the two values defined in this specification.
98. **forbidden**: The term "forbidden" when used in the clauses defining the coded bit stream indicates that the value shall never be used. This is usually to avoid emulation of start codes.
99. **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.
100. **forward compatibility**: A newer coding standard is forward compatible with an older coding standard if decoders designed to operate with the newer coding standard are able to decode bitstreams of the older coding standard.
101. **forward motion vector [video]**: A motion vector that is used for motion compensation from a reference picture at an earlier time in display order.
102. **frame [audio]**: A part of the audio signal that corresponds to audio PCM samples from an Audio Access Unit.
103. **frame [video]**: A frame contains lines of spatial information of a video signal. For progressive video, these lines contain samples starting from one time instant and continuing through successive lines to the bottom of the frame. For interlaced video a frame consists of two fields, a top field and a bottom field. One of these fields will commence one field period later than the other.
104. **frame period**: The reciprocal of the frame rate.
105. **frame picture; frame structure picture [video]**: A frame structure picture is a coded picture with picture\_structure is equal to "Frame".
106. **frame rate**: The rate at which frames are be output from the decoding process.
107. **free format [audio]**: Any bitrate other than the defined bitrates that is less than the maximum valid bitrate for each layer.
108. **future reference frame (field)**: A future reference frame(field) is a reference frame(field) that occurs at a later time than the current picture in display order.
109. **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.
110. **granules [Layer III] [audio]**: 576 frequency lines that carry their own side information.
111. **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 ISO/IEC 13818-2.
112. **Hann window [audio]**: A time function applied sample-by-sample to a block of audio samples before Fourier transformation.
113. **header**: A block of data in the coded bitstream containing the coded representation of a number of data elements pertaining to the coded data that follow the header in the bitstream.
114. **Huffman coding**: A specific method for entropy coding.
115. **hybrid filterbank [audio]**: A serial combination of subband filterbank and MDCT.
116. **hybrid scalability**: Hybrid scalability is the combination of two (or more) types of scalability.
117. **I-field picture [video]**: A field structure I-Picture.
118. **I-frame picture [video]**: A frame structure I-Picture.
119. **I-picture; intra-coded picture [video]**: A picture coded using information only from itself.
120. **IMDCT [audio]**: Inverse Modified Discrete Cosine Transform.
121. **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.
122. **interlace [video]**: The property of conventional television frames where alternating lines of the frame represent different instances in time. In an interlaced frame, one of the field is meant to be displayed first. This field is called the first field. The first field can be the top field or the bottom field of the frame.
123. **intra coding [video]**: Coding of a macroblock or picture that uses information only from that macroblock or picture.
124. **intra-coded picture; I-picture [video]**: A picture coded using information only from itself.
125. **ISO/IEC 11172 (multiplexed) stream [system]**: A bit stream composed of zero or more elementary streams combined in the manner defined in ISO/IEC 11172-1.
126. **ISO/IEC 13818 (multiplexed) stream [system]**: A bit stream composed of 0 or more elementary streams combined in the manner defined in Part 1 of this Recommendation | International Standard.
127. **joint stereo coding [audio]**: Any method that exploits stereophonic irrelevance or stereophonic redundancy.

- 128. **joint stereo mode [audio]**: A mode of the audio coding algorithm using joint stereo coding.
- 129. **layer [audio]**: One of the levels in the coding hierarchy of the audio system defined in this part of the CD.
- 130. **layer [video and systems]**: One of the levels in the data hierarchy of the video and system specifications defined in Part 1 and Part 2.
- 131. **level** : A defined set of constraints on the values which may be taken by the parameters of this specification within a particular profile. A profile may contain one or more levels.
- 132. **low frequency enhancement channel [audio]**: A limited bandwidth channel for low frequency audio effects in a multichannel system.
- 133. **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.
- 134. **macroblock [video]**: The four 8 by 8 blocks of luminance data and the two (for 4:2:0 chrominance format), four (for 4:2:2 chrominance format) or eight (for 4:4:4 chrominance format) 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 sample data and sometimes to the coded representation of the sample values and other data elements defined in the macroblock header of the syntax defined in this part of this specification. The usage is clear from the context.
- 135. **mapping [audio]**: Conversion of an audio signal from time to frequency domain by subband filtering and/or by MDCT.
- 136. **masking [audio]**: A property of the human auditory system by which an audio signal cannot be perceived in the presence of another audio signal .
- 137. **masking threshold [audio]**: A function in frequency and time below which an audio signal cannot be perceived by the human auditory system.
- 138. **MDCT [audio]**: Modified Discrete Cosine Transform which correspond to the Time Domain Aliasing Cancellation Filter Bank.
- 139. **motion compensation [video]**: The use of motion vectors to improve the efficiency of the prediction of sample values. The prediction uses motion vectors to provide offsets into the past and/or future reference frames or reference fields containing previously decoded sample values that are used to form the prediction error signal.
- 140. **motion estimation [video]**: The process of estimating motion vectors during the encoding process.
- 141. **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.
- 142. **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.
- 143. **multichannel [audio]**: A combination of audio channels used to create a spatial sound field.
- 144. **multilingual [audio]**: A presentation of dialogue in more than one language.
- 145. **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.
- 146. **non-tonal component [audio]**: A noise-like component of an audio signal.
- 147. **Nyquist sampling**: Sampling at or above twice the maximum bandwidth of a signal.
- 148. **P-field picture [video]**: A field structure P-Picture.
- 149. **P-frame picture [video]**: A frame structure P-Picture.
- 150. **P-picture; predictive-coded picture [video]**: A picture that is coded using motion compensated prediction from past reference fields or frame.
- 151. **pack [system]**: A pack consists of a pack header followed by zero or more packets. It is a layer in the system coding syntax described in clause **Error! Reference source not found.** on page 44 this Recommendation | International Standard.
- 152. **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 clause [XX] of this Recommendation | International Standard.
- 153. **packet data [system]**: Contiguous bytes of data from an elementary stream present in a packet.
- 154. **packet header [system]**: The data structure used to convey information about the elementary stream data contained in the packet data.
- 155. **packet identifier; PID [system]**: A unique integer value used to associate elementary streams of a program in a single or multi-program Transport Stream as described in [XXXX]
- 156. **padding [audio]**: A method to adjust the average length of an audio frame in time to the duration of the corresponding PCM samples, by conditionally adding a slot to the audio frame.

- 157.**parameter**: A variable within the syntax of this specification which may take one of a large range of values. A variable which can take one of only two values is a flag and not a parameter.
- 158.**past reference frame (field)**: A past reference frame(field) is a reference frame(field) that occurs at an earlier time than the current picture in display order.
- 159.**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.
- 160.**pel [video]**: Picture element.
- 161.**pel aspect ratio [video]**: The ratio of the nominal vertical height of pel on the display to its nominal horizontal width.
- 162.**PES [system]**: An abbreviation for Packetized Elementary Stream.
- 163.**PES packet [system]**: The data structure used to carry elementary stream data. It consists of a PES packet header followed by PES packet payload and is described in clause [XXX]
- 164.**PES Stream [system]**: A PES Stream consists of PES packets, all of whose payloads consist of data from a single elementary stream, and all of which have the same stream\_id. Specific semantic constraints apply.
- 165.**picture period [video]**: The reciprocal of the picture rate.
- 166.**picture rate [video]**: The nominal rate at which pictures should be output from the decoding process.
- 167.**picture**: 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. For progressive video, a picture is identical to a frame, while for interlaced video, a picture can refer to a frame, or the top field or the bottom field of the frame depending on the context.
- 168.**polyphase filterbank [audio]**: A set of equal bandwidth filters with special phase interrelationships, allowing for an efficient implementation of the filterbank.
- 169.**prediction [audio]**: The use of a predictor to provide an estimate of the subband sample in one channel from the subband samples in other channels.
- 170.**prediction [video]**: The use of a predictor to provide an estimate of the pel value or data element currently being decoded.
- 171.**prediction error [video]**: The difference between the actual value of a pel or data element and its predictor.
- 172.**prediction**: The use of a predictor to provide an estimate of the sample value or data element currently being decoded.
- 173.**predictive-coded picture; P-picture [video]**: A picture that is coded using motion compensated prediction from past reference frames or reference fields.
- 174.**predictor**: A linear combination of previously decoded sample values or data elements.
- 175.**presentation channel [audio]**: audio channels at the output of the decoder corresponding to the loudspeaker positions left, center, right, left surround and right surround.
- 176.**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.
- 177.**presentation unit; PU [system]**: A decoded audio access unit or a decoded picture.
- 178.**profile**: A defined subset of the syntax of this specification.
- 179.**program [system]**: A program is a collection of elementary streams with a common time base.
- 180.**Program Specific Information; PSI [system]**: PSI consists of normative data which is necessary for the demultiplexing of Transport Streams and the successful regeneration of programs and is described in clause [XX]. One case of PSI, the non-mandatory network information table, is privately defined.
- 181.**progressive [video]**: The property of film frames where all the samples of the frame represent the same instances in time.
- 182.**psychoacoustic model [audio]**: A mathematical model of the masking behaviour of the human auditory system.
- 183.**quantisation matrix [video]**: A set of sixty-four 8-bit values used by the dequantiser.
- 184.**quantised DCT coefficients**: DCT coefficients before dequantisation. A variable length coded representation of quantised DCT coefficients is transmitted as part of the compressed video bitstream.
- 185.**quantiser scalefactor [video]**: A data element represented in the bit stream and used by the decoding process to scale the dequantisation.
- 186.**random access**: The process of beginning to read and decode the coded bit stream at an arbitrary point.
- 187.**reconstructed frame [video]**: A reconstructed frame consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. A reconstructed frame is obtained by decoding a coded frame.
- 188.**reconstructed picture [video]**: A reconstructed picture is obtained by decoding a coded picture. A reconstructed picture is either a reconstructed frame (when decoding a frame picture), or one field of a

- reconstructed frame (when decoding a field picture). If the coded picture is a field picture, then the reconstructed picture is the top field or the bottom field of the reconstructed frame.
- 189.**reference field [video]**: A reference field is one field of a reconstructed frame. Reference fields are used for forward and backward prediction when P-pictures and B-pictures are decoded. Note that when field P-pictures are decoded, prediction of the second field P-picture of a coded frame uses the first reconstructed field of the same coded frame as a reference field.
- 190.**reference frame [video]**: A reference frame is a reconstructed frame that was coded in the form of a coded I-frame or a coded P-frame. Reference frames are used for forward and backward prediction when P-pictures and B-pictures are decoded.
- 191.**reference picture [video]**: Reference pictures are the nearest adjacent I- or P-pictures to the current picture in display order.
- 192.**reorder buffer [video]**: A buffer in the system target decoder for storage of a reconstructed I-picture or a reconstructed P-picture.
- 193.**requantisation [audio]**: Decoding of coded subband samples in order to recover the original quantised values.
- 194.**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.
- 195.**reverse playback [video]**: The process of displaying the picture sequence in the reverse of display order.
- 196.**sample aspect ratio [video]**: (abbreviated to **SAR**). This specifies the distance between samples. It is defined (for the purposes of this specification) as the vertical displacement of the lines of luminance samples in a frame divided by the horizontal displacement of the luminance samples. Thus its units are (metres per line) ÷ (metres per sample)
- 197.**scalability**: Scalability is the ability of a decoder to decode an ordered set of bitstreams to produce a reconstructed sequence. Moreover, useful video is output when subsets are decoded. The minimum subset that can thus be decoded is the first bitstream in the set which is called the base layer. Each of the other bitstreams in the set is called an enhancement layer. When addressing a specific enhancement layer, "lower layer" refer to the bitstream which precedes the enhancement layer.
- 198.**scalefactor [audio]**: Factor by which a set of values is scaled before quantisation.
- 199.**scalefactor band [audio]**: A set of frequency lines in Layer III which are scaled by one scalefactor.
- 200.**scalefactor index [audio]**: A numerical code for a scalefactor.
- 201.**sequence header [video]**: A block of data in the coded bit stream containing the coded representation of a number of data elements.
- 202.**side information**: Information in the bitstream necessary for controlling the decoder.
- 203.**skipped macroblock [video]**: A macroblock for which no data are stored.
- 204.**slice**: A series of macroblocks.
- 205.**slot [audio]**: A slot is an elementary part in the bit stream. In Layer I a slot equals four bytes, in Layers II and III one byte.
- 206.**SNR scalability [video]**: A type of scalability where the enhancement layer (s) contain only coded refinement data for the DCT coefficients of the lower layer.
- 207.**source stream**: A single non-multiplexed stream of samples before compression coding.
- 208.**spatial scalability**: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer without using motion vectors. The layers can have different frame sizes, frame rates or chrominance formats
- 209.**spreading function [audio]**: A function that describes the frequency spread of masking effects.
- 210.**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 structures in the coding syntax.
- 211.**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.
- 212.**stereo mode [audio]**: Mode, where two audio channels which form a stereo pair (left and right) are encoded within one bit stream. The coding process is the same as for the dual channel mode.
- 213.**stereo-irrelevant [audio]**: a portion of a stereophonic audio signal which does not contribute to spatial perception.
- 214.**stuffing (bits); stuffing (bytes)** : Code-words that may be inserted into the compressed bit stream that are discarded in the decoding process. Their purpose is to increase the bitrate of the stream.
- 215.**subband [audio]**: Subdivision of the audio frequency band.
- 216.**subband filterbank [audio]**: A set of band filters covering the entire audio frequency range. In this part of the CD, the subband filterbank is a polyphase filterbank.

- 217.**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.
- 218.**surround channel [audio]**: An audio presentation channel added to the front channels (L and R or L, R, and C) to enhance the spatial perception.
- 219.**syncword [audio]**: A 12-bit code embedded in the audio bit stream that identifies the start of a frame.
- 220.**synthesis filterbank [audio]**: Filterbank in the decoder that reconstructs a PCM audio signal from subband samples.
- 221.**system header [system]**: The system header is a data structure defined in clause of this Recommendation | International Standard that carries information summarizing the system characteristics of the ISO/IEC 13818 multiplexed stream.
- 222.**system target decoder; STD [system]**: A hypothetical reference model of a decoding process used to describe the semantics of an ISO/IEC 13818 multiplexed bit stream.
- 223.**temporal scalability [video]**: A type of scalability where an enhancement layer also uses predictions from sample data derived from a lower layer using motion vectors. The layers have identical frame size, and chrominance formats, but can have different frame rates.
- 224.**time-stamp [system]**: A term that indicates the time of an event.
- 225.**tonal component [audio]**: A sinusoid-like component of an audio signal.
- 226.**top field [video]**: One of two fields that comprise a frame. Each line of a top field is spatially located immediately above the corresponding line of the bottom field.
- 227.**~~Transport Stream packet~~transport packet header [system]**: A data structure used to convey information about the Transport Stream payload.
- 228.**triplet [audio]**: A set of 3 consecutive subband samples from one subband. A triplet from each of the 32 subbands forms a granule.
- 229.**variable bitrate**: Operation where the bitrate varies with time during the decoding of a compressed bit stream.
- 230.**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.
- 231.**video buffering verifier; VBV**: 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.
- 232.**video sequence**: The highest syntactic structure of coded video bitstreams. It contains a series of one or more coded frames.
- 233.**zig-zag scanning order [video]**: A specific sequential ordering of the DCT coefficients from (approximately) the lowest spatial frequency to the highest.

## 4 Symbols and Abbreviations

CPB	Constrained Parameters Bitstreams
CBP	Coded Block Pattern
DCT	Discrete Cosine Transform
DFD [video]	Displaced frame difference. Macroblock prediction error.
FDCT	Forward DCT
FLC	Fixed Length Code
IDCT	Inverse DCT
M	Distance between successive P pictures
Macroblock row	A continuous strip of macroblocks along a common horizontal strip.
MB [video]	Macroblock
N	Distance 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

## **6 Systems Simulation**

The systems source code is listed in Annex A. The current program demultiplexes program and transport layer bitstreams, and provides some verification and analysis of streams.

[Future version will include encoder]

## **7 Video Simulation**

The video source code is listed in Annex B.

The video encoder has the following characteristics:

- generates constant rate bitstreams
- performs adaptive quantization based on spatial activity measure normalized against activity of previous picture.
- rate control adjusted locally by buffer fullness model
- global bit allocation determined on a combination of target bit rate and moving average scene complexity.
- MPEG-1 and MPEG-2 bitstreams
- Progressive or Interlaced sequences
- Progressive or Interlaced Frames
- Frame macroblock prediction Frame pictures
- Field macroblock prediction in Frame pictures
- Field macroblock prediction in Field pictures
- 16x8 macroblock prediction in Field pictures
- Dual Prime prediction in both Frame and Field pictures.
- Linear or Non-linear macroblock quantization scale factors
- Constant B-P frame sequence

The decoder follows all normative guidelines of 13818-2 for Main Profile, and performs most modes of Spatial, SNR Profiles. Experimental modes such as Data Partitioning are also included. The decoder also forms the basis of a verifier.

## 8 Audio Simulation

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 features supported by the code are detailed in table 8.1.

**Table 8.1 List of audio simulation features**

Feature	Encoder	Decoder
layer 1	no	no
layer 2	yes	yes
bit rates	all	all
sample rates	32, 44.1, 48	32, 44.1, 48
multichannel mode	yes	yes
low sampling frequency mode	no	no
multilingual/commentary channels	no	no
channel configurations	3/2	3/2
transmission channel allocations	<del>0all</del>	all
dematrix procedures	0	0
dynamic crosstalk	no	no
prediction	<del>nozero-order</del>	<del>nozero-order</del>
phantom coding of center channel	yes	yes
extension bitstream	<del>noyes</del>	<del>noyes</del>
ancillary data	no	yes
CRC	yes	yes
predistortion	yes	not applicable
layer 3	no	no
compilation environment	Unix	Unix

### User information

The MPEG-2/audio Layer 2 software package consists of:

- 21 data files tables
- 9 source files (\*.c)
- 3 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 in 3/2 multichannel configuration, sampled at 32, 44.1, or 48 kHz with 16 bits per sample. For 3/2 configuration data the samples should be interleaved in the order L,R,C,Ls,Rs. 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.



## 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.

The decoder program has a very crude implementation of bitstream synchword detection. It may not be able to correctly decode valid bitstreams which have false synchword patterns in the ancillary data portion of the bitstream.

## 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. [##### amend for MPEG-2]

```
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)
```

## 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, [##### amend for MPEG-2]

```
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 and "II" for layer 2
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)
```

**Bitstream verification procedure**

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

Filename	Contents
m384.mpg	Layer 2
512tc.mpg, 512tc.ext	Layer 2 with extension bitstream
m512.mpg, m512.ext	Layer 2 with extension bitstream
pred384.mpg	Layer 2 with zero-order prediction
m384pc.mpg	Layer 2 with phantom coding of center channel
m512dypc.mpg, m512dypc.ext	Layer 2 with extension bitstream, dynamic crosstalk and phantom coding of center channel

[insert description of verification procedure]

## Annex A

### Systems - code listings

#### A.1 Introduction

This Annex contains the C source code listings for the CD 13818-5 Systems codec.

**Table A.1 List of systms files**

filename	description
readme	informative file
commondm.c	common routines for the CRC_32_Checking, Descriptors and the DSM_CC Decoder routines
commondm.h	common constants
makefile	example makefile for GNU gcc Unix and MS-DOS compilers
mpeg2bs.c	Main routine.
mpeg2pr.c	contains the Program_demultiplexor for both Program and Transport streams
mpeg2sys.c	contains all the variables defined globally
mpeg2sys.h	common constants
mpeg2tr.c	This is the file that contains programs needed for the transport streams

#### A.2 readme

```

/* *****
* README: Programs for the MPEG-2 System Demultiplexor for      *
* both Transport and Program Streams.                            *
*****
* Originator: Munsu A. Haque; Date: September 26th, 1994.      *
* Hyundai Electronics America, San Jose, CA.                    *
* e-mail: munsu@heava.com                                       *
* Version: 1.3                                                  *
*****

```

makefile: it is just a one line command for compilation.

mpeg2Sys.h: This is the file that contains all the variables defined globally. This file contains lots of variables that may be local to the routines. These are kept as they are for the time being.  
At the top, if "#define TS\_STREAM" is in, the compiled program generates the Demultiplexor for the Transport Stream, while "#define PS\_STREAM" does the same for the Program Stream.

mpeg2Sys.c: Main routine.

mpeg2Pr.c: This is the file that contains the Program\_demultiplexor for both Program and Transport streams.

mpeg2Tr.c: This is the file that contains programs needed for the transport streams.

commonDmux.c & commonDmux.h: Some common routines for the CRC\_32\_Checking, Descriptors and the DSM\_CC Decoder routines.

mpeg2BS.c: Description of various Bitstreams under Testing.

```

** *****

```

Testing:

- 1) Program\_Stream: Two streams are tested with this program. These are "Matsuhita.ps" and "Hyundai.ps". They worked perfectly.

2) Transport\_Stream: Many bitstreams are used to verify the Transport Stream Demultiplexor. The names of these are available in the program "mpeg2BS.c". Please find the related information there. In order to work with any one bitstream, please specify the corresponding as follows:

```
/* Conformance Bitstream at Norway Meeting */
#define SA      0 /* CONFORM BS: Worked */
#define DIVICOM 0 /* CONFORM BS: Worked */
#define TT1     0 /* CONFORM BS: PMAP ? mux_buff-descriptor */
#define TERACOM 1 /* CONFORM BS: */
```

All the conformance bitstreams are passed, except the TT1 bitstream (Tandberg Television A/S). It was found that this bitstream utilized an old descriptor definition (before the Atlanta DIS) for the "Multiplex-buffer-utilization" descriptor. The bitstream contained 1 byte for the information after the descriptor\_length, which should be 3 according to the new DIS. Also many other bitstreams are tested whose results you may find in the same program.

NOTE: Because of no "transport\_stream\_end\_code" unlike Program\_stream, these codes are not smart enough yet to extract a few hundred bytes of the PES Packets at the end of the bitstream file. As this is end-case problem, it is overlooked now. So the decoded "video only" or "audio only" raw bitstreams files will have shorter sizes than those provided by the MPEG Conformance Bitstream for comparison.

\*\*\*\*\*/

### A.3 commondm.c

```
/******
 * commonDmux.c: Some common routines are present here.
 * Included are: 32-bit CRC Checking;
 * All 13 or more descriptors mentioned in the System DIS, June, 1994;
 * and DSM_CC Decoder.
 *****/
 * Originator: Munsu A. Haque; Date: September 13th, 1994.
 * Hyundai Electronics America, San Jose, CA.
 * e-mail: munsu@heava.com
 *****/
*/
#include <stdio.h>
#include <math.h>
#include "commonDmux.h"
#include "mpeg2Sys.h"

/* 32-bit CRC Checking : It works on byte by byte data */
int Check_CRC(first,last)
unsigned char *first,*last;
{
    register unsigned char data;
    register shift_reg;
    register unsigned char *word_addr;
    register count;

    printf("CHECKING CRC_32 ..... \n");

    /* Preset the shift_register to '1's */
```

```

shift_reg = 0xffffffff ;

for(word_addr=first;word_addr<last-1;word_addr++)
{
    data = *word_addr ;
    for(count=0;count<8;count++)
    {
        if ((data^shift_reg) < 0)
        {
            shift_reg = shift_reg << 1 ;
            shift_reg = shift_reg ^ 0x04c11db7 ;
        }
        else
            shift_reg = shift_reg << 1 ;
        data = data << 1 ;
    }
}

word_addr = last - 1;
data = *word_addr ;
data = ~data ;

for(count=0;count<8;count++)
{
    if ((data^shift_reg) < 0)
    {
        shift_reg = shift_reg << 1 ;
        shift_reg = shift_reg ^ 0x04c11db7 ;
    }
    else
        shift_reg = shift_reg << 1 ;
    data = data << 1 ;
}

/* OR shift_regs, '0' = no errors and '1' = errors */
return(shift_reg != 0x00000000) ;
}

void descriptor_ps(trace)
int trace;
{
    unsigned int i,j,n1,n2,temp,temp1;

    /* Get a byte */
    descriptor_tag = System_buffer[bytes_out_System_buffer++];
    if((descriptor_tag >= 64) && (descriptor_tag < 256))
        descriptor_tag = 64;
    /* Get another byte */
    descriptor_length = System_buffer[bytes_out_System_buffer++];

    if(descriptor_length) {
        switch(descriptor_tag) {
            case 2: /* Video_Stream_Descriptor */
                if(trace) printf("Video Stream Descriptor Information... \n");
                temp = System_buffer[bytes_out_System_buffer++];
                multiple_frame_rate_flag = temp >> 7;
                frame_rate_code = (temp & 0x78) >> 3;
                MPEG_2_flag = (temp & 0x4) >> 2;

```

```

        constrained_parameter_flag = (temp & 0x2) >> 1;
        still_picture_flag = temp & 0x1;
        n1 = 1;
        if(MPEG_2_flag) {
            profile_and_level_indication
                = System_buffer[bytes_out_System_buffer++];
            temp = System_buffer[bytes_out_System_buffer++];
            chroma_format = (temp & 0xc0) >> 6;
            frame_rate_extension_flag = (temp & 0x20) >> 5;
            /* next 5-bits are reserved */
            n1 += 2;
        }
        break;

case 3: /* Audio_Stream_Descriptor */
    if(trace) printf("Audio Stream Descriptor Information.. \n");
    temp = System_buffer[bytes_out_System_buffer++];
    free_format_flag = temp >> 7;
    audio_ID = (temp & 0x40) >> 6;
    audio_Layer = (temp & 0x30) >> 4;
    n1 = 1;
    /* next 4-bits are reserved */
    break;

case 4: /* Hierarchy_Descriptor */
    if(trace) printf("Hierarchy Descriptor Information.. \n");
    temp = System_buffer[bytes_out_System_buffer++];
    hierarchy_type = temp & 0xf;
    temp = System_buffer[bytes_out_System_buffer++];
    hierarchy_layer_index = temp & 0x3f;
    temp = System_buffer[bytes_out_System_buffer++];
    hierarchy_embedded_layer = temp & 0x3f;
    temp = System_buffer[bytes_out_System_buffer++];
    hierarchy_priority = temp & 0x3f;
    n1 = 4;
    break;

case 5: /* Registration_Descriptor */
    if(trace) printf("Registration Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    reg_format_identifier = temp1;
    n1 = descriptor_length - 4;
    for(i=0; i<n1; i++)
        reg_additional_id_info[i]
            = System_buffer[bytes_out_System_buffer++];
    n1 = descriptor_length;
    break;

case 6: /* Data_Stream_Alignment_Descriptor */
    if(trace) printf("Data Stream Alignment Descriptor Information.. \n");
    alignment_type = System_buffer[bytes_out_System_buffer++];
    n1 = 1;
    break;

case 7: /* Target_Background_Grid_Descriptor */

```

```

    if(trace) printf("Target Background Grid Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    TBG_horizontal_size = (temp1 >> 18) & 0x3fff;
    TBG_vertical_size = (temp1 & 0x3fff0) >> 4;
    TBG_pel_aspect_ratio = temp1 & 0xf;
    n1 = 4;
    break;

case 8: /* Video_Window_Descriptor */
    if(trace) printf("Video Window Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    VW_horizontal_offset = (temp1 >> 18) & 0x3fff;
    VW_vertical_offset = (temp1 & 0x3fff0) >> 4;
    VW_window_priority = temp1 & 0xf;
    n1 = 4;
    break;

case 9: /* CA_Descriptor */
    if(trace) printf("Conditional Access Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    CA_system_ID = temp1;
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    CAT_PID = temp1 & 0x1fff;
    n1 = descriptor_length - 4;
    for(i=0; i<n1; i++)
        CA_Desc_private_data_byte[i]
            = System_buffer[bytes_out_System_buffer++];

    n1 = descriptor_length;
    break;

case 10: /* ISO_639_Language_Descriptor */
    if(trace) printf("ISO 639 Language Descriptor Information.. \n");
    n1 = descriptor_length - 1;
    n2 = n1 / 3;
    for(j=0; j<n2; j++) {
        temp1 = 0;
        for(i=0; i<3; i++) {
            temp = System_buffer[bytes_out_System_buffer++];
            temp1 = (temp1 << 8) | temp;
        }
        ISO_639_Language_code[j] = temp1;
    }

```

```

    /* Is there any stuffing ? */
    n1 = n1 - (n2 * 3);
    for(i=0; i<n1; i++)
        temp = System_buffer[bytes_out_System_buffer++];

    ISO_audio_type = System_buffer[bytes_out_System_buffer++];
    n1 = descriptor_length;
    break;

case 11: /* System_Clock_Descriptor */
    if(trace) printf("System Clock Descriptor Information.. \n");
    temp = System_buffer[bytes_out_System_buffer++];
    external_clock_reference_indicator = temp >> 7;
    clock_accuracy_integer = temp & 0x3f;
    temp = System_buffer[bytes_out_System_buffer++];
    clock_accuracy_exponent = temp >> 5;
    /* next 5-bits are reserved */
    n1 = 2;
    break;

case 12: /* Multiplex_Buffer_Utilization_Descriptor */
    if(trace) printf("Multiplex Buffer Utilization Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    mdv_valid_flag = temp1 >> 15;
    multiplex_delay_variation = temp1 & 0x7fff;
    temp = System_buffer[bytes_out_System_buffer++];
    multiplex_strategy = temp >> 5;
    /* next 5-bits are reserved */
    n1 = 3;
    break;

case 13: /* Copyright_Descriptor */
    if(trace) printf("Copyright Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    copyright_identifier = temp1;
    n1 = descriptor_length - 4;
    for(i=0; i<n1; i++)
        additional_copyright_info[i]
            = System_buffer[bytes_out_System_buffer++];
    n1 = descriptor_length;
    break;

case 14: /* Maximum_Bitrate_Descriptor */
    if(trace) printf("Maximum Bitrate Descriptor Information.. \n");
    temp1 = 0;
    for(i=0; i<3; i++) {
        temp = System_buffer[bytes_out_System_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    maximum_bitrate = temp1 >> 2;

```



```

        /* next 2-bits are reserved */
        n1 = 3;
        break;

    case 64: /* Private_Data_Indicator_Descriptor */
        if(trace) printf("Private Data Indicator Descriptor Information.. \n");
        temp1 = 0;
        for(i=0; i<4; i++) {
            temp = System_buffer[bytes_out_System_buffer++];
            temp1 = (temp1 << 8) | temp;
        }
        private_data_indicator = temp1;
        n1 = 4;
        break;

    default: /* Undefined Descriptor */
        if(trace) printf("Descriptor is not defined .. \n");
        n1 = 0;
        break;
} /* Close the Switch */

if(n1 < descriptor_length) {
    printf("Stuffing bytes may be present in the descriptor .. \n");
    for(i=n1; i<descriptor_length; i++)
        temp = System_buffer[bytes_out_System_buffer++];
}
}

}

#ifdef TS_STREAM

/*****
/*
    The following routine is valid for Descriptor
/* descType, the Input Parameter, describes the descriptor type
/* as follows: descType = stream_type (0x00 to 0xff) and
/* 257 for CAS, 258 for TS_PMAP_PInfo, 260 for PS_PSM_PInfo
*****/
void descriptor_ts(descType,bufData)
unsigned int descType;
unsigned int *bufData;
{
    unsigned int i,j,n1,n2,temp,temp1;

    /* Get a byte */
    descriptor_tag = getSplicedByte(bufData);
    if((descriptor_tag >= 64) && (descriptor_tag < 256))
        descriptor_tag = 64;
    /* Get another byte */
    descriptor_length = getSplicedByte(bufData);

    if(descriptor_length) {
        switch(descriptor_tag) {
            case 2: /* Video_Stream_Descriptor */
#ifdef debugS
                printf("Video Stream Descriptor Information... \n");
#endif
                temp = getSplicedByte(bufData);

```

```

    multiple_frame_rate_flag = temp >> 7;
    frame_rate_code = (temp & 0x78) >> 3;
    MPEG_2_flag = (temp & 0x4) >> 2;
    constrained_parameter_flag = (temp & 0x2) >> 1;
    still_picture_flag = temp & 0x1;
    n1 = 1;
    if(MPEG_2_flag) {
        profile_and_level_indication
            = getSplicedByte(bufData);

        temp = getSplicedByte(bufData);
        chroma_format = (temp & 0xc0) >> 6;
        frame_rate_extension_flag = (temp & 0x20) >> 5;
        /* next 5-bits are reserved */
        n1 += 2;
    }
    break;

case 3: /* Audio_Stream_Descriptor */
#ifdef debugS
    printf("Audio Stream Descriptor Information.. \n");
#endif
    temp = getSplicedByte(bufData);
    free_format_flag = temp >> 7;
    audio_ID = (temp & 0x40) >> 6;
    audio_Layer = (temp & 0x30) >> 4;
    n1 = 1;
    /* next 4-bits are reserved */
    break;

case 4: /* Hierarchy_Descriptor */
#ifdef debugS
    printf("Hierarchy Descriptor Information.. \n");
#endif
    temp = getSplicedByte(bufData);
    hierarchy_type = temp & 0xf;
    temp = getSplicedByte(bufData);
    hierarchy_layer_index = temp & 0x3f;
    temp = getSplicedByte(bufData);
    hierarchy_embedded_layer = temp & 0x3f;
    temp = getSplicedByte(bufData);
    hierarchy_priority = temp & 0x3f;
    n1 = 4;
    break;

case 5: /* Registration_Descriptor */
#ifdef debugS
    printf("Registration Descriptor Information.. \n");
#endif
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(bufData);
        temp1 = (temp1 << 8) | temp;
    }
    reg_format_identifier = temp1;
    n1 = descriptor_length - 4;
    for(i=0; i<n1; i++)
        reg_additional_id_info[i]
            = getSplicedByte(bufData);

```

```

        n1 = descriptor_length;
        break;

    case 6: /* Data_Stream_Alignment_Descriptor */
#ifdef debugS
        printf("Data Stream Alignment Descriptor Information.. \n");
#endif
        alignment_type = getSplicedByte(bufData);
        if((descType == 1) || (descType == 2)) { /* Video */
            videoStream_alignment_type = alignment_type;
        }
        else if((descType == 3) || (descType == 4)) { /* Audio */

            audioStream_alignment_type = alignment_type;
        }
        n1 = 1;
        break;

    case 7: /* Target_Background_Grid_Descriptor */
#ifdef debugS
        printf("Target Background Grid Descriptor Information.. \n");
#endif
        temp1 = 0;
        for(i=0; i<4; i++) {
            temp = getSplicedByte(bufData);
            temp1 = (temp1 << 8) | temp;
        }
        TBG_horizontal_size = (temp1 >> 18) & 0x3fff;
        TBG_vertical_size = (temp1 & 0x3fff0) >> 4;
        TBG_pel_aspect_ratio = temp1 & 0xf;
        n1 = 4;
        break;

    case 8: /* Video_Window_Descriptor */
#ifdef debugS
        printf("Video Window Descriptor Information.. \n");
#endif
        temp1 = 0;
        for(i=0; i<4; i++) {
            temp = getSplicedByte(bufData);
            temp1 = (temp1 << 8) | temp;
        }
        VW_horizontal_offset = (temp1 >> 18) & 0x3fff;
        VW_vertical_offset = (temp1 & 0x3fff0) >> 4;
        VW_window_priority = temp1 & 0xf;
        n1 = 4;
        break;

    case 9: /* CA_Descriptor */
#ifdef debugS
        printf("Conditional Access Descriptor Information.. \n");
#endif
        temp1 = 0;
        for(i=0; i<2; i++) {
            temp = getSplicedByte(bufData);
            temp1 = (temp1 << 8) | temp;
        }
        CA_system_ID = temp1;

```

```

    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = getSplicedByte(bufData);
        temp1 = (temp1 << 8) | temp;
    }
    CAT_PID = temp1 & 0x1fff;
    /* Now append descType with this CAT_PID for future use */
    /* as this CAT_PID may be valid for any descType, not */
    /* only for the CAS, but also for PMAP and other ESs */
    CAT_PID |= (descType << 16);

    n1 = descriptor_length - 4;
    for(i=0; i<n1; i++)
        CA_Desc_private_data_byte[i]
            = getSplicedByte(bufData);

    n1 = descriptor_length;
    break;

case 10: /* ISO_639_Language_Descriptor */
#ifdef debugS
    printf("ISO 639 Language Descriptor Information.. \n");
#endif
    n1 = descriptor_length - 1;
    n2 = n1 / 3;
    for(j=0; j<n2; j++) {
        temp1 = 0;
        for(i=0; i<3; i++) {
            temp = getSplicedByte(bufData);
            temp1 = (temp1 << 8) | temp;
        }
        ISO_639_Language_code[j] = temp1;
    }

    /* Is there any stuffing ? */
    n1 = n1 - (n2 * 3);
    for(i=0; i<n1; i++)
        temp = getSplicedByte(bufData);

    ISO_audio_type = getSplicedByte(bufData);
    n1 = descriptor_length;
    break;

case 11: /* System_Clock_Descriptor */
#ifdef debugS
    printf("System Clock Descriptor Information.. \n");
#endif
    temp = getSplicedByte(bufData);
    external_clock_reference_indicator = temp >> 7;
    clock_accuracy_integer = temp & 0x3f;
    temp = getSplicedByte(bufData);
    clock_accuracy_exponent = temp >> 5;
    /* next 5-bits are reserved */
    n1 = 2;
    break;

case 12: /* Multiplex_Buffer_Utilization_Descriptor */
#ifdef debugS
    printf("Multiplex Buffer Utilization Descriptor Information.. \n");

```

```

#endif
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = getSplicedByte(bufData);
        temp1 = (temp1 << 8) | temp;
    }
    mdv_valid_flag = temp1 >> 15;
    multiplex_delay_variation = temp1 & 0x7fff;
    temp = getSplicedByte(bufData);
    multiplex_strategy = temp >> 5;
    /* next 5-bits are reserved */
    n1 = 3;
    break;

    case 13: /* Copyright_Descriptor */
#ifdef debugS
    printf("Copyright Descriptor Information.. \n");
#endif
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(bufData);
        temp1 = (temp1 << 8) | temp;
    }
    copyright_identifier = temp1;
    n1 = descriptor_length - 4;
    for(i=0; i<n1; i++)
        additional_copyright_info[i]
            = getSplicedByte(bufData);
    n1 = descriptor_length;
    break;

    case 14: /* Maximum_Bitrate_Descriptor */
#ifdef debugS
    printf("Maximum Bitrate Descriptor Information.. \n");
#endif
    temp1 = 0;
    for(i=0; i<3; i++) {
        temp = getSplicedByte(bufData);
        temp1 = (temp1 << 8) | temp;
    }
    maximum_bitrate = temp1 >> 2;
    /* next 2-bits are reserved */
    n1 = 3;
    break;

    case 64: /* Private_Data_Indicator_Descriptor */
#ifdef debugS
    printf("Private Data Indicator Descriptor Information.. \n");
#endif
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(bufData);
        temp1 = (temp1 << 8) | temp;
    }
    private_data_indicator = temp1;
    n1 = 4;
    break;

```

```

    default: /* Undefined Descriptor */
#ifdef debugS
    printf("Descriptor is not defined .. \n");
#endif
    n1 = 0;
    break;
} /* Close the Switch */

if(n1 < descriptor_length) {
#ifdef debugS
    printf("Stuffing bytes may be present in the descriptor .. \n");
#endif
    for(i=n1; i<descriptor_length; i++)
        temp = getSplicedByte(bufData);
    }
}

#endif

void dsmcc_time_code(trace)
unsigned int trace;
{
    unsigned int i,temp,temp1;

    temp = System_buffer[bytes_out_System_buffer++];
    /* next 7-bits are reserved */
    dsmcc_infinite_time_flag = temp & 0x1;

    if(!dsmcc_infinite_time_flag) {
        /* next byte */
        temp = System_buffer[bytes_out_System_buffer++];
        /* 1st 4-bits are reserved */
        dsmcc_msb_PTS = (temp & 0x08) >> 3;
        dsmcc_lsb_PTS = (temp & 0x06) << 29;
        if(!(temp & 0x1)) {
            printf("Marker-bit (1) in dsmcc_PTS is wrong \n");
            exit(1);
        }

        /* 2 bytes */
        temp1 = 0;
        for(i=0; i<2; i++) {
            temp = System_buffer[bytes_out_System_buffer++];
            temp1 = (temp1 << 8) | temp;
        }
        if(!(temp1 & 0x1)) {
            printf("Marker-bit (2) in dsmcc_PTS is wrong \n");
            exit(1);
        }
        dsmcc_lsb_PTS |= ((temp1 & 0xfffe) << 14);

        /* 2 bytes */
        temp1 = 0;
        for(i=0; i<2; i++) {
            temp = System_buffer[bytes_out_System_buffer++];
            temp1 = (temp1 << 8) | temp;
        }
    }
}

```

```

    if(!(temp1 & 0x1)) {
        printf("Marker-bit (3) in dsmcc_PTS is wrong \n");
        exit(1);
    }
    dsmcc_lsb_PTS |= (temp1 >> 1);
    if(trace) printf("dsmcc_PTS = %0x %0x\n", dsmcc_msb_PTS, dsmcc_lsb_PTS);
}

}

void DSM_CC_Decoder(trace)
unsigned int trace;
{
    unsigned int i, temp, temp1;

    switch(dsmcc_command_id) {
        case 0: /* Control */
            /* Get the 1st byte */
            temp = System_buffer[bytes_out_System_buffer++];
            dsmcc_select_flag = (temp >> 7);
            dsmcc_retrieval_flag = (temp >> 6) & 0x1;
            dsmcc_storage_flag = (temp >> 5) & 0x1;
            /* next 12-bits are reserved */
            temp = System_buffer[bytes_out_System_buffer++];
            if(!(temp & 0x1)) {
                printf("Marker-bit (1) in DSM_CC_Control is wrong \n");
                exit(1);
            }

            if(dsmcc_select_flag) {
                temp1 = 0;
                for(i=0; i<2; i++) {
                    temp = System_buffer[bytes_out_System_buffer++];
                    temp1 = (temp1 << 8) | temp;
                }
                if(!(temp1 & 0x1)) {
                    printf("Marker-bit (2) in DSM_CC_Control is wrong \n");
                    exit(1);
                }
                dsmcc_bitstream_id = (temp1 & 0xffff) << 16;

                temp1 = 0;
                for(i=0; i<2; i++) {
                    temp = System_buffer[bytes_out_System_buffer++];
                    temp1 = (temp1 << 8) | temp;
                }
                if(!(temp1 & 0x1)) {
                    printf("Marker-bit (3) in DSM_CC_Control is wrong \n");
                    exit(1);
                }
                dsmcc_bitstream_id |= ((temp1 & 0xffff) << 1);

                temp = System_buffer[bytes_out_System_buffer++];
                if(!(temp & 0x1)) {
                    printf("Marker-bit (4) in DSM_CC_Control is wrong \n");
                    exit(1);
                }
                dsmcc_bitstream_id |= (temp >> 6);
                if(trace) printf("dsmcc_bitstream_id = %0x\n", dsmcc_bitstream_id);
            }
        }
    }
}

```

```

        dsmcc_select_mode = (temp >> 1) & 0x1f;
    }

    if(dsmcc_retrieval_flag) {
        temp = System_buffer[bytes_out_System_buffer++];
        dsmcc_jump_flag = temp >> 7;
        dsmcc_play_flag = (temp >> 6) & 0x1;
        dsmcc_pause_mode = (temp >> 5) & 0x1;
        dsmcc_resume_mode = (temp >> 4) & 0x1;
        dsmcc_stop_mode = (temp >> 3) & 0x1;
        /* next 10-bits are reserved */
        temp = System_buffer[bytes_out_System_buffer++];
        if(!(temp & 0x1)) {
            printf("Marker-bit (5) in DSM_CC_Control is wrong \n");
            exit(1);
        }

        if(dsmcc_jump_flag) {
            temp = System_buffer[bytes_out_System_buffer++];
            /* next 7-bits are reserved */
            dsmcc_direction_indicator = temp & 0x1;
            dsmcc_time_code(trace);
        }

        if(dsmcc_play_flag) {
            temp = System_buffer[bytes_out_System_buffer++];
            dsmcc_speed_mode = temp >> 7;
            dsmcc_direction_indicator = (temp >> 6) & 0x1;
            /* next 6-bits are reserved */
            dsmcc_time_code(trace);
        }
    }

    if(dsmcc_storage_flag) {
        temp = System_buffer[bytes_out_System_buffer++];
        /* next 6-bits are reserved */
        dsmcc_record_flag = (temp >> 1) & 0x1;
        dsmcc_stop_mode = temp & 0x1;
        if(dsmcc_record_flag)
            dsmcc_time_code(trace);
    }

    break;

case 1:      /* Acknowledge */
    /* Get the 1st byte */
    temp = System_buffer[bytes_out_System_buffer++];
    dsmcc_select_ack = (temp >> 7);
    dsmcc_retrieval_ack = (temp >> 6) & 0x1;
    dsmcc_storage_ack = (temp >> 5) & 0x1;
    dsmcc_error_ack = (temp >> 4) & 0x1;
    /* next 10-bits are reserved */
    temp = System_buffer[bytes_out_System_buffer++];
    if(!(temp & 0x2)) {
        printf("Marker-bit (1) in DSM_CC_Ack is wrong \n");
        exit(1);
    }
    dsmcc_cmd_status = temp & 0x1;

    if(dsmcc_cmd_status &&
        (dsmcc_retrieval_ack || dsmcc_storage_ack) ) {
        dsmcc_time_code(trace);
    }

```



```

    }
    break;

default: /* Acknowledge */
    printf("DSM_Command is undefined ...\n");
    exit(0);
}

}

```

#### A.4 commondm.h

```

/* CommonDmux.h : Global Variables used in commonDmux.c          *
 * **** *
 * Originator: Munsir A. Haque; Date: September 13th, 1994.      *
 * Hyundai Electronics America, San Jose, CA.                   *
 * e-mail: munsir@heava.com                                     *
 * **** *
 */

unsigned int descriptor_tag,descriptor_length,multiple_frame_rate_flag;
unsigned int frame_rate_code,MPEG_2_flag,constrained_parameter_flag;
unsigned int still_picture_flag,profile_and_level_indication;
unsigned int chroma_format,frame_rate_extension_flag;
unsigned int free_format_flag,audio_ID,audio_Layer;
unsigned int hierarchy_type,hierarchy_layer_index;
unsigned int hierarchy_embedded_layer,hierarchy_priority;
unsigned int reg_format_identifier,reg_additional_id_info[32];
unsigned int alignment_type,TBG_horizontal_size,TBG_vertical_size;
unsigned int TBG_pel_aspect_ratio,VW_horizontal_offset;
unsigned int VW_vertical_offset,VW_window_priority;
unsigned int CA_system_ID,CA_PID,CA_Desc_private_data_byte[32];
unsigned int ISO_639_Language_code[32],ISO_audio_type;
unsigned int external_clock_reference_indicator,clock_accuracy_integer;
unsigned int clock_accuracy_exponent,mdv_valid_flag;
unsigned int multiplex_delay_variation,multiplex_strategy;
unsigned int copyright_identifier,additional_copyright_info[32];
unsigned int maximum_bitrate,private_data_indicator;

unsigned int dsmcc_command_id,dsmcc_select_flag,dsmcc_retrieval_flag;
unsigned int dsmcc_storage_flag,dsmcc_bitstream_id,dsmcc_select_mode;
unsigned int dsmcc_jump_flag,dsmcc_play_flag,dsmcc_pause_mode;
unsigned int dsmcc_resume_mode,dsmcc_stop_mode,dsmcc_direction_indicator;
unsigned int dsmcc_speed_mode,dsmcc_direction_indicator,dsmcc_record_flag;
unsigned int dsmcc_select_ack,dsmcc_retrieval_ack,dsmcc_storage_ack;
unsigned int dsmcc_error_ack,dsmcc_cmd_status,dsmcc_infinite_time_flag;
unsigned int dsmcc_msb_PTS,dsmcc_lsb_PTS;
unsigned int videoStream_alignment_type,audioStream_alignment_type;

```

#### A.5 makefile

```
cc -o mpeg2Ts mpeg2Sys.c mpeg2BS.c mpeg2Tr.c mpeg2Pr.c commonDmux.c -lm -lc
```

## A.6 mpeg2bs.c

```

/* *****
* mpeg2BS.c: This program creates a Pseudo User-interface to handle *
* various input Bitstreams from the MPEG-2 Systems Bitstream *
* Exchange Ad-hoc Committe as well as the System Conformance *
* Committe. It defines the respective user-selectable PID values *
* and corresponding initialization. *
* *****
* Originator: Munsu A. Haque; Date: September 13th, 1994. *
* Hyundai Electronics America, San Jose, CA. *
* e-mail: munsu@heava.com *
* *****
*/

#include <stdio.h>
#include <math.h>
#include "mpeg2Sys.h"

#ifdef TS_STREAM
/* Input Transport bitstreams Sources */
/* GI: Video Pes_Packet_Length = 0 */
/* ALCATEL: Marker-bit (1) in DTS_next_au wrong */
/* DIVICOM: Video_PES_Packet_Length = 0 */
/* HHI: Video OK, Audio Less data stored */
/* TT1: Video_PES_Packet_Length = 0 */
/* TI_TRDC: PTS_DTS_flags Marker (Audio) wrong */
/* LEP_01: Video Problem */
/* LEP_02: Video_PES_Packet_Length = 0 */
/* SAMSUNG:Worked Perfectly */

#define GI 0 /* NOT GOOD */
#define ALCATEL 0 /* NOT GOOD */
#define HHI 0 /* NOT GOOD */
#define TI_TRDC 0 /* NOT GOOD */
#define LEP_01 0 /* NOT GOOD */
#define HHI_snr 1 /* GOOD */
#define HYUNDAI 0 /* GOOD */
#define LEP_02 0 /* GOOD */
#define SAMSUNG 0 /* GOOD */

/* Conformance Bitstream at Norway Meeting */
#define SA 0 /* CONFORM BS: Worked */
#define DIVICOM 0 /* CONFORM BS: Worked */
#define TT1 0 /* CONFORM BS: Did not work:PMAP ? mux_buff-descriptor */
#define TERACOM 0 /* CONFORM BS: */

void user_interface_for_main(trace)
unsigned int trace;
{

    int i,program_number,video_type,audio_type;

    printf("Which program_number you wish to select ?\n");
    fscanf(stdin,"%d",&program_number);
    printf("You selected program-%d for decoding\n",program_number);

```

```

program_select = program_number;

/* HYUNDAI */
if(HYUNDAI) {
    programMAP[0].PID = 0x050; /* nitPID */
    programMAP[1].PID = 0x070;
    programMAP[2].PID = 0x071;
    if(program_number == 1) {
        videoPIDselect = 0x100; /* MPEG-2 Video */
        audioPIDselect = 0x101; /* MPEG-1 Audio */
        nitPIDselect = programMAP[0].PID; /* NIT */
        pvt1PIDselect = 0x200; /* Private Stream_1 */
        pvt2PIDselect = 0x201; /* Private Stream_2 */
    }
    else if(program_number == 2) {
        videoPIDselect = 0x110; /* MPEG-2 Video */
        audioPIDselect = 0x111; /* MPEG-1 Audio */
        nitPIDselect = programMAP[0].PID; /* NIT */
    }
}

/* GI */
if(GI) {
    programMAP[1].PID = 0x032;
    programMAP[2].PID = 0x033;
    if(program_number == 1) {
        videoPIDselect = 0x100; /* MPEG-2 Video */
        audioPIDselect = 0x102; /* MPEG-2 Audio */
    }
    else if(program_number == 2) {
        videoPIDselect = 0x101; /* MPEG-2 Video */
        audioPIDselect = 0x1fff; /* No MPEG-2 Audio */
    }
}

/* ALCATEL */
if(ALCATEL) {
    programMAP[1].PID = 1024;
    programMAP[2].PID = 1026;
    if(program_number == 1) {
        videoPIDselect = 0x14; /* MPEG-2 Video */
        audioPIDselect = 0x32; /* MPEG-2 Audio */
    }
    else if(program_number == 2) {
        videoPIDselect = 0x1e; /* MPEG-2 Video */
        audioPIDselect = 0x3c; /* No MPEG-2 Audio */
    }
}

/* HHI */
if(HHI) {
    programMAP[1].PID = 0x10;
    if(program_number == 1) {
        videoPIDselect = 0x14; /* MPEG-2 Video */
        audioPIDselect = 0x34; /* MPEG-2 Audio */
    }
}

/* HHI_snr: 4 video & 4 audio in 2 Programs */

```

```

if(HHI_snr) {
    programMAP[1].PID = 0x110;
    programMAP[2].PID = 0x111;

    if(program_number == 1) {
        videoPIDselect = 0x114; /* MPEG-2 Video: 0x124 ignored */
        /* videoPIDselect = 0x124; /* MPEG-2 Video: 0x114 ignored */
        audioPIDselect = 0x134; /* MPEG-2 Audio: 0x144 ignored */
        /* audioPIDselect = 0x144; /* MPEG-2 Audio: 0x134 ignored */
        pvt2PIDselect = 0x154; /* Private Stream_2 */
        /* pvt2PIDselect = 0x164; /* Private Stream_2 */
    }
    else if(program_number == 2) {
        videoPIDselect = 0x119; /* MPEG-2 Video: 0x129 ignored */
        /* videoPIDselect = 0x129; /* MPEG-2 Video: 0x119 ignored */
        audioPIDselect = 0x139; /* MPEG-2 Audio: 0x149 ignored */
        /* audioPIDselect = 0x149; /* MPEG-2 Audio: 0x139 ignored */
        pvt2PIDselect = 0x159; /* Private Stream_2 */
        /* pvt2PIDselect = 0x169; /* Private Stream_2 */
    }
}

/* TT1 */
if(TT1) {
    programMAP[1].PID = 0x20;
    if(program_number == 1) {
        videoPIDselect = 0x30; /* MPEG-2 Video */
        audioPIDselect = 0x31; /* MPEG-2 Audio */
    }
}

/* TI_TRDC */
if(TI_TRDC) {
    programMAP[1].PID = 0x100;
    if(program_number == 1) {
        videoPIDselect = 0x901; /* MPEG-2 Video */
        audioPIDselect = 0x902; /* MPEG-2 Audio */
    }
}

/* LEP_01: PHILIPS */
if(LEP_01) {
    programMAP[1].PID = 0xaab;
    if(program_number == 1) {
        videoPIDselect = 0xabc; /* MPEG-2 Video */
        audioPIDselect = 0xbcd; /* MPEG-1 Audio */
    }
}

/* LEP_02: PHILIPS */
if(LEP_02) {
    programMAP[1].PID = 0x301;
    if(program_number == 1) {
        videoPIDselect = 0x201; /* MPEG-2 Video */
        audioPIDselect = 0x202; /* MPEG-1 Audio */
    }
}

/* SAMSUNG */

```

```

if(SAMSUNG) {
programMAP[1].PID = 0x880;
if(program_number == 1) {
    videoPIDselect = 0x800; /* MPEG-2 Video */
    audioPIDselect = 0x801; /* MPEG-2 Audio: 5.1 Channel */
}
}

/* SA: Scientific Atlanta */
if(SA) {
programMAP[0].PID = 34;
programMAP[1].PID = 68;
/* Conditional Access Information PID = 67 : Not yet implemented */
if(program_number == 1) {
    videoPIDselect = 133; /* MPEG-2 Video */
    audioPIDselect = 134; /* MPEG-2 Audio */
}
nitPIDselect = programMAP[0].PID; /* NIT */
}

/* DIVICOM */
if(DIVICOM) {
programMAP[1].PID = 0x10;
if(program_number == 1) {
    videoPIDselect = 0x11; /* MPEG-1 Video */
    audioPIDselect = 0x12; /* MPEG-1 Audio */
}
}

/* TT: Tandenberg TV */
if(TT1) {
programMAP[1].PID = 0x20;
if(program_number == 1) {
    videoPIDselect = 0x30; /* MPEG-2 Video */
    audioPIDselect = 0x31; /* MPEG-2 Audio */
}
}

/* TERACOM */
if(TERACOM) {
programMAP[0].PID = 55;
programMAP[1].PID = 77;
programMAP[2].PID = 77;
/* Conditional Access Information: CA_PID = 80 */
if(program_number == 1) {
    videoPIDselect = 56; /* MPEG-2 Video */
    audioPIDselect = 57; /* MPEG-2 Audio */
}
else if(program_number == 2) {
    videoPIDselect = 58; /* MPEG-2 Video */
    audioPIDselect = 59; /* No MPEG-2 Audio */
}
nitPIDselect = programMAP[0].PID; /* NIT */
}

if(trace) printf("selected videoPID=%0x\n",videoPIDselect);
if(trace) printf("selected audioPID=%0x\n",audioPIDselect);
if(trace) printf("selected pvt1PID=%0x\n",pvt1PIDselect);
if(trace) printf("selected pvt2PID=%0x\n",pvt2PIDselect);

```

```

programPIDselect = programMAP[program_number].PID;
if(trace) printf("selected programMAP_PID=%0x\n",programPIDselect);

}

#endif

```

## A.7 mpeg2pr.c

```

/*****
 * mpeg2Pr.c: MPEG-II Program Stream Bitstream Parser:      *
 * This part contains the program_stream Demultiplexing.    *
 *****/
/*****
 * MPEG Bitstream Parser:                                  *
 * SYSTEM:                                                  *
    iso_11172_end_code      00 00 01 b9
    pack_start_code         00 00 01 ba
    system_header_start     00 00 01 bb
    packet_start_prefix     00 00 01
    stream_id               bc : reserved stream
                           bd : private stream_1
                           be : padding stream
                           bf : private stream_2
                           c0 - df : audio stream
                           e0 - ef : video stream
                           f0 - ff : reserved stream
 *****/
 * Originator: Munsir A. Haque; Date: September 13th, 1994. *
 * Hyundai Electronics America, San Jose, CA.              *
 * e-mail: munsir@heava.com                                *
 *****/
*/
#include <stdio.h>
#include <math.h>

#include "mpeg2Sys.h"
#include "commonDmux.h"

/*****
 *****/
/***** PROGRAM STREAM *****/
/*****/

/* Pack Header */
void pack_header(trace)
int trace;
{
    unsigned int i,temp,temp1;

    /* Another byte */
    temp = Decoder_buffer[bytes_out_Decompressor++];
    if((temp & 0xc0) != 0x40) {
        printf("The first Marker-bits (01) in pack_header are wrong \n");
        exit(1);
    }
}

```

```

msb_SCRB = (temp & 0x20) >> 5;
temp1 = temp & 0x03;
temp1 |= ((temp & 0x18) >> 1);
system_clock_reference_base = temp1 << 28;
if(!(temp & 0x4)) {
    printf("Marker-bit (1) in system_clock_reference_base is wrong \n");
    exit(1);
}

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x4)) {
    printf("Marker-bit (2) in system_clock_reference_base is wrong \n");
    exit(1);
}
temp = (temp1 & 0xff8) >> 1;
temp |= (temp1 & 0x3); /* 15-bit */
system_clock_reference_base |= (temp << 13);

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x4)) {
    printf("Marker-bit (3) in system_clock_reference_base is wrong \n");
    exit(1);
}
system_clock_reference_base |= (temp1 >> 3);
if(trace)
    printf("system_clock_reference_base = %0x %0x\n",msb_SCRB,system_clock_reference_base);
system_clock_reference_extension = (temp & 0x3) << 7;

/* Another byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer++];
system_clock_reference_extension |= temp >> 1;
if(trace)
    printf("system_clock_reference_extension = %0x\n",system_clock_reference_extension);
if(!(temp & 0x1)) {
    printf("Marker-bit (4) in system_clock_reference_base is wrong \n");
    exit(1);
}

/* 3 bytes */
temp1 = 0;
for(i=0; i<3; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
program_mux_rate = (temp1 >> 2);

if(trace) printf("program_mux_rate = %d\n",program_mux_rate);

if((temp1 & 0x3) != 0x3) {

```

```

    printf("Two consecutive Marker-bits (11) in program_mux_rate is wrong \n");
    exit(1);
}

/* Another byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer++];
pack_stuffing_length = temp & 0x7;
for(i=0; i<pack_stuffing_length; i++) {
    temp1 = Decoder_buffer[bytes_out_Decoder_buffer++];
    if(temp1 != 0xff) {
        printf("Wrong Stuffing Bytes in Pack_Header ...\n");
        exit(1);
    }
}

}

/* System_Header */
void system_header(trace)
int trace;
{
    unsigned int i,temp,temp1,doflag;
    int datCount;

    /* next 3 bytes */
    rate_bound = 0;
    for(i=0; i<3; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        rate_bound = (rate_bound << 8) | temp;
    }
    if((rate_bound & 0x800000) != 0x800000) {
        printf("WARNING: Marker bit (1) in system_header is not matched \n");
        exit(1);
    }
    if((rate_bound & 0x1) != 1) {
        printf("WARNING: Marker bit (2) in system_header is not matched \n");
        exit(1);
    }
    rate_bound = (rate_bound & 0x7ffffe) >> 1;
    if(trace) printf("rate_bound=%d\n",rate_bound);

    /* next byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    audio_bound = temp >> 2;
    fixed_flag = (temp & 0x2) >> 1;
    CSPA_flag = temp & 0x1;
    if(trace) {
        printf("audio_bound=%d\n",audio_bound);
        printf("fixed_flag=%d\n",fixed_flag);
        printf("CSPA_flag=%d\n",CSPA_flag);
    }

    /* next byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    system_audio_lock_flag = (temp & 0x80) >> 7;
    system_video_lock_flag = (temp & 0x40) >> 6;
    if((temp & 0x20) != 0x20) {
        printf("WARNING: Marker bit (3) in system_header is not matched \n");
        exit(1);
    }
}

```



```

video_bound = temp & 0x1f;
if(trace) printf("video_bound=%d\n",video_bound);

/* next byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer++];
if(temp != 0xff) {
    printf("WARNING: Wrong reserved_byte in the system_header \n");
    exit(1);
}

datCount = 6;
/* next byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer];
if((temp & 0x80) == 0x80) {
    do {
        stream_id = Decoder_buffer[bytes_out_Decoder_buffer++];
        if(trace) printf("stream_id=0x%x\n",stream_id);

        /* 2 bytes */
        temp1 = 0;
        for(i=0; i<2; i++) {
            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            temp1 = (temp1 << 8) | temp;
        }
        if((temp1 & 0xc000) != 0xc000) {
            printf("Marker-11 in P_STD_buffer_bound (system_header) is wrong \n");
            exit(1);
        }
        P_STD_buffer_bound_scale = (temp1 >> 13) & 0x1;
        P_STD_buffer_size_bound = temp1 & 0x1fff;

        if(trace) printf("P_STD_buffer_size_bound=%d\n",P_STD_buffer_size_bound);
        datCount += 3;

        /* next byte */
        temp = Decoder_buffer[bytes_out_Decoder_buffer];
        if(datCount == system_header_length) temp = 0; /* safe rule */
    } while ((temp & 0x80) == 0x80);
}

if(datCount != system_header_length) {
    printf("datCount=%d and system_header_length=%d\n",datCount,system_header_length);
    printf("WARNING: system_header_length != data needed in System_Header block..\n");
}
if(datCount < system_header_length) {
    for(i=datCount; i<system_header_length; i++)
        bytes_out_Decoder_buffer++;
}
else {
    printf("ERROR in system_header..\n");
    exit(0);
}
}

/*****
*   PES_packet(): Header demuxing and then return   *
*****/
void PES_packet(trace,pktLength)

```

```

int trace,*pktLength;
{
    unsigned int i,temp,temp1,stuffing_byte,streamID;
    int n1,n2,n3,N;

    /* 1st byte */
    n1 = bytes_out_Decoder_buffer;
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    if((temp & 0xc0) != 0x80) {
        printf("WARNING: PES_packet: Marker - 10 is not matched \n");
        temp1 = temp & 0xc0;
        if(temp1 == 0xc0)
            printf("PES_packet: Marker - 11 is present\n");
    }
    PES_scrambling_control = (temp & 0x30) >> 4;
    PES_priority = (temp >> 3) & 0x1;
    data_alignment_indicator = (temp >> 2) & 0x1;
    copyright = (temp >> 1) & 0x1;
    original_or_copy = temp & 0x1;

    /* 2nd byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    PTS_DTS_flags = (temp >> 6);
    ESCR_flag = (temp >> 5) & 0x1;
    ES_rate_flag = (temp >> 4) & 0x1;
    DSM_trick_mode_flag = (temp >> 3) & 0x1;
    additional_copy_info_flag = (temp >> 2) & 0x1;
    PES_CRC_flag = (temp >> 1) & 0x1;
    PES_extension_flag = temp & 0x1;

    /* 3rd byte */
    PES_header_data_length = Decoder_buffer[bytes_out_Decoder_buffer++];
    n2 = bytes_out_Decoder_buffer;
    if(PTS_DTS_flags == 0x2) {
        /* 4th byte */
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        if((temp & 0xf0) != 0x20) {
            printf("WARNING: PTS_DTS_flags: Marker-bits (0010) is not matched ..\n");
            exit(0);
        }
        msb_PTS = (temp & 0x08) >> 3;
        PTS = (temp & 0x06) << 29;
        if(!(temp & 0x1)) {
            printf("Marker-bit (1) in PTS is wrong \n");
            exit(1);
        }
    }

    /* 5-6th bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("Marker-bit (2) in PTS is wrong \n");
        exit(1);
    }
    PTS |= ((temp1 & 0xfffe) << 14);

```

```

/* 7-8th bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("Marker-bit (3) in PTS is wrong \n");
    exit(1);
}
PTS |= (temp1 >> 1);
if(trace) printf("PTS = %0x %0x\n",msb_PTS,PTS);

}

if(PTS_DTS_flags == 0x3) {

/* 4th byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer++];
if((temp & 0xf0) != 0x30) {
    printf("WARNING: Marker - 0011 is not matched \n");
    temp1 = temp & 0xf0;
    if(temp1 == 0x20)
        printf("instead Marker - 0010 is present \n");
    else if(temp1 == 0x10)
        printf("instead Marker - 0001 is present \n");
    else if(temp1 == 0x00)
        printf("instead Marker - 0000 is present \n");
    exit(0);
}
msb_PTS = (temp & 0x08) >> 3;
PTS = (temp & 0x06) << 29;
if(!(temp & 0x1)) {
    printf("Marker-bit (4) in PTS is wrong \n");
    exit(1);
}

/* 5-6th bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("Marker-bit (5) in PTS is wrong \n");
    exit(1);
}
PTS |= ((temp1 & 0xfffe) << 14);

/* 7-8th bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("Marker-bit (6) in PTS is wrong \n");
    exit(1);
}

```

```

    }
    PTS |= (temp1 >> 1);
    if(trace) printf("PTS = %0x %0x\n",msb_PTS,PTS);

    /* 9th byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    if((temp & 0xf0) != 0x10) {
        printf("Error in the packet_layer \n");
        exit(1);
    }
    msb_DTS = (temp & 0x08) >> 3;
    DTS = (temp & 0x06) << 29;
    if(!(temp & 0x1)) {
        printf("Marker-bit (1) in DTS is wrong \n");
        exit(1);
    }

    /* 10-11th bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("Marker-bit (2) in decoding_time_stamp is wrong \n");
        exit(1);
    }
    DTS |= ((temp1 & 0xfffe) << 14);

    /* 12-13th bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("Marker-bit (3) in decoding_time_stamp is wrong \n");
        exit(1);
    }
    DTS |= (temp1 >> 1);
    if(trace) printf("DTS = %0x %0x\n",msb_DTS,DTS);
}

if(ESCR_flag) {
    /* Another byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    msb_ESCR = (temp & 0x20) >> 5;
    temp1 = temp & 0x03;
    temp1 |= ((temp & 0x18) >> 1);
    ESCR = temp1 << 28;
    if(!(temp & 0x4)) {
        printf("Marker-bit (1) in ESCR is wrong \n");
        exit(1);
    }

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];

```

```

    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x4)) {
    printf("Marker-bit (2) in ESCR is wrong \n");
    exit(1);
}
temp = (temp1 & 0xff8) >> 1;
temp |= (temp1 & 0x3); /* 15-bit */
ESCR |= (temp << 13);

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x4)) {
    printf("Marker-bit (3) in ESCR is wrong \n");
    exit(1);
}
ESCR |= (temp1 >> 3);
if(trace) printf("ESCR = %0x %0x\n",msb_ESCR,ESCR);
ESCR_extension = (temp & 0x3) << 7;

/* Another byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer++];
ESCR_extension |= temp >> 1;
if(trace) printf("ESCR_extension = %0x\n",ESCR_extension);
if(!(temp & 0x1)) {
    printf("Marker-bit (4) in ESCR is wrong \n");
    exit(1);
}
}

if(ES_rate_flag) {
    /* Another byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    ES_rate = (temp & 0x7f) << 15;
    if(!(temp & 0x80)) {
        printf("Marker-bit (1) in ES_rate is wrong \n");
        exit(1);
    }

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("Marker-bit (2) in ES_rate is wrong \n");
        exit(1);
    }
    ES_rate |= (temp1 >> 1);
    if(trace) printf("ES_rate = %0x\n",ES_rate);
}

if(DSM_trick_mode_flag) {
    /* Another byte */

```

```

temp = Decoder_buffer[bytes_out_Decoder_buffer++];
trick_mode_control = temp >> 5;
switch(trick_mode_control) {
    case 0:
    case 3:
        field_id = (temp >> 3) & 0x3;
        intra_slice_refresh = (temp >> 2) & 0x1;
        frequency_truncation = temp & 0x3;
        break;

    case 1:
    case 4:
        field_rep_cntrl = temp & 0x1f;
        break;

    case 2:
        field_id = (temp >> 3) & 0x3;
        break;

    default: /* cases 5 to 7 are reserved */
        break;
}

}

if(additional_copy_info_flag) {
    /* Another byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    additional_copy_info = temp & 0x7f;
    if(!(temp & 0x80)) {
        printf("Marker-bit (1) in additional_copy_info is wrong \n");
        exit(1);
    }
}

if(PES_CRC_flag) {
    /* 2 bytes */
    previous_PES_packet_CRC = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        previous_PES_packet_CRC = (previous_PES_packet_CRC << 8) | temp;
    }
}

if(PES_extension_flag) {
    /* Another byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    PES_private_data_flag = (temp & 0x80) >> 7;
    pack_header_field_flag = (temp & 0x40) >> 6;
    program_packet_sequence_counter_flag = (temp & 0x20) >> 5;
    P_STD_buffer_flag = (temp & 0x10) >> 4;
    PES_extension_flag_2 = temp & 0x1;
}
else {
    PES_private_data_flag = 0;
    pack_header_field_flag = 0;
    program_packet_sequence_counter_flag = 0;
    P_STD_buffer_flag = 0;
    PES_extension_flag_2 = 0;
}

```

```

    }

    if(PES_private_data_flag) {
        /* 16 bytes */
        for(i=0; i<16; i++)
            PES_private_data[i] = Decoder_buffer[bytes_out_Decoder_buffer++];
    }

    if(pack_header_field_flag) {
        /* One byte */
        pack_field_length = Decoder_buffer[bytes_out_Decoder_buffer++];
        n3 = bytes_out_Decoder_buffer;
        temp = 0;
        for(i=0; i<4; i++)
            temp = (temp << 8) | Decoder_buffer[bytes_out_Decoder_buffer++];
        if(temp != 0x1ba) {
            printf("It is not a pack_start_code ...\n");
            exit(0);
        }
        pack_header(trace);
        n3 = bytes_out_Decoder_buffer - n3;
        if(n3 < pack_field_length) {
            if(trace) printf("Now get the optional System_header\n");

            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            if(!temp) {
                printf("Unknown data is present ??? \n");
                exit(0);
            }

            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            if(!temp) {
                printf("Unknown data is present ??? \n");
                exit(0);
            }

            do {
                temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            } while(temp == 0); /* again 00 */

            if(temp == 1) { /*3 00 00 01 */
                streamID = Decoder_buffer[bytes_out_Decoder_buffer++];
                if(streamID != 0xbb) {
                    printf("Unknown data is present ??? \n");
                    exit(0);
                }
                system_header(trace);
            }
            else {
                printf("Unknown data is present ??? \n");
                exit(0);
            }
        } /* pack_field_length */
    } /* pack_header_field_flag */

    if(program_packet_sequence_counter_flag) {
        /* One byte */
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        program_packet_sequence_counter = temp & 0x7f;
    }

```

```

if(!(temp & 0x80)) {
    printf("Marker-bit (1) in program_packet_sequence_counter is wrong \n");
    exit(1);
}

/* One byte */
temp = Decoder_buffer[bytes_out_Decoder_buffer++];
MPEG1_MPEG2_identifier = (temp & 0x40) >> 6;
original_stuff_length = temp & 0x3f;
if(!(temp & 0x80)) {
    printf("Marker-bit (2) in program_packet_sequence_counter is wrong \n");
    exit(1);
}
}

if(P_STD_buffer_flag) {
    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if((temp1 & 0xc0) != 0x40) {
        printf("Marker- 01 in P_STD_buffer is wrong \n");
        exit(1);
    }
    P_STD_buffer_scale = (temp1 >> 13) & 0x1;
    P_STD_buffer_size = temp1 & 0x1fff;
}

if(PES_extension_flag_2) {
    /* One byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    PES_extension_field_length = temp & 0x7f;
    if(!(temp & 0x80)) {
        printf("Marker-bit (1) in PES_extension_field is wrong \n");
        exit(1);
    }
    /* reserved */
    bytes_out_Decoder_buffer += PES_extension_field_length;
}

/* Number of bytes used in PES_header_flag_data */
n2 = bytes_out_Decoder_buffer - n2;

/* Number of stuffing bytes in PES_header */
N = PES_header_data_length - n2;
for(i=0; i<N; i++) {
    stuffing_byte = Decoder_buffer[bytes_out_Decoder_buffer++];
    if(stuffing_byte != 0xff) {
        printf("Wrong Stuffing Bytes in PES_header ..\n");
        exit(1);
    }
}

/* Determine size of remainder of the packet containing data-bytes */
/* Number of bytes used in PES_packet */
n1 = bytes_out_Decoder_buffer - n1;
(*pktLength) -= n1;

```



```

    if(PES_packet_length > 0)
        PacketSize = PES_packet_length - n1;

    if(trace)
        printf("Number of bytes used by Packet_Header (n1) = %d\n",n1);
    if(trace)
        printf("Extract %d bytes of data from the packet.... \n",PacketSize);
}

/* program_stream_map */
void program_stream_map(trace)
int trace;
{
    unsigned int i,temp,temp1;
    int n1,n2,n3,descLength,not_ok,ePno=0;
    unsigned char *data_ptr,*crc_ptr;

    /* start of the PSM */
    data_ptr = &Decoder_buffer[bytes_out_Decoder_buffer - 6];

    /* next byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    current_next_indicator = (temp >> 7);
    /* next 2-bits are reserved */
    program_stream_map_vesrion = temp & 0x1f;

    /* next byte:7-bits reserved and 1-bit Marker */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    if(!(temp & 0x1)) {
        printf("WARNING: Marker bit (1) in program_stream_map is not matched\n");
        exit(1);
    }

    /* program_stream_info_length */
    program_stream_info_length = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        program_stream_info_length = (program_stream_info_length << 8) | temp;
    }
    if(trace)
        printf("program_stream_info_length=%d\n",program_stream_info_length);
    n1 = 4;
    if(program_stream_info_length) {
        descLength = 0;
        do {
            bytes_out_System_buffer = bytes_out_Decoder_buffer;
            System_buffer = Decoder_buffer;

            descriptor_ps(trace);
            descLength += (2 + descriptor_length);

            bytes_out_Decoder_buffer = bytes_out_System_buffer;
            Decoder_buffer = System_buffer;

        } while (descLength < program_stream_info_length);
    }
}

```

```

n1 += program_stream_info_length;

/* elementary_stream_map_length */
elementary_stream_map_length = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    elementary_stream_map_length = (elementary_stream_map_length << 8) | temp;
}
if(trace)
    printf("elementary_stream_map_length=%d\n",elementary_stream_map_length);

n1 += 2; /* total number of bytes used so far */
n2 = 0;
if(elementary_stream_map_length > 0) {
    do {
        /* next byte */
        stream_type = Decoder_buffer[bytes_out_Decoder_buffer++];
        if(trace) printf("stream_type=%0x h\n",stream_type);

        /* next byte */
        elementary_stream_id = Decoder_buffer[bytes_out_Decoder_buffer++];
        if(trace) printf("elementary_stream_id=%0x h\n",elementary_stream_id);

        /* elementary_stream_info_length */
        elementary_stream_info_length = 0;
        for(i=0; i<2; i++) {
            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            elementary_stream_info_length = (elementary_stream_info_length << 8) | temp;
        }
        if(trace) printf("elementary_stream_info_length=%d\n",elementary_stream_info_length);
        if(elementary_stream_info_length) {
            descLength = 0;
            do {
                bytes_out_System_buffer = bytes_out_Decoder_buffer;
                System_buffer = Decoder_buffer;

                descriptor_ps(trace);
                descLength += (2 + descriptor_length);

                bytes_out_Decoder_buffer = bytes_out_System_buffer;
                Decoder_buffer = System_buffer;

            } while (descLength < elementary_stream_info_length);
        }

        n2 += (elementary_stream_info_length + 4);
        n3 = elementary_stream_map_length - n2;

        } while (n3 > 0);
    }

n1 +=elementary_stream_map_length;

/* PSM_CRC_32: */
CRC_32 = 0;
for(i=0; i<4; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    CRC_32 = (CRC_32 << 8) | temp;
}

```

```

if(trace) printf("PSM_CRC_32 = %0xh\n",CRC_32);
n1 += 4;
if(n1 != program_stream_map_length) {
    printf("ERROR: Wrong number of bytes in program_stream_map .. |n");
    exit(1);
}

/* address of the last bytes of PSM including CRC-32 */
crc_ptr = &Decoder_buffer[bytes_out_Decoder_buffer];

/* Check PSM_CRC_32 */
not_ok = Check_CRC(data_ptr,crc_ptr);
if (not_ok)
    printf("PSM: CRC showed parity error !!\n");
else
    printf("PSM: CRC checked OK\n");
}

/* program_stream_directory_PES_packet */
void PS_directory_PES_packet(trace)
int trace;
{
    unsigned int i,cnt,temp,temp1;
    unsigned int klok1,clok2;
    unsigned int prev_directory_offset_msb,prev_directory_offset_lsb;
    unsigned int next_directory_offset_msb,next_directory_offset_lsb;

    /* number_of_access_units */
    number_of_access_units = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        number_of_access_units = (number_of_access_units << 8) | temp;
    }
    if(!(number_of_access_units & 0x1)) {
        printf("WARNING: Marker (1) in Directory_PES_Packet is not matched \n");
        exit(0);
    }
    if(trace) printf("number_of_access_units=%d\n",number_of_access_units);
    number_of_access_units = number_of_access_units >> 1;
    PSD_Directory.number_of_access_units = number_of_access_units;

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("WARNING: Marker (2) in Directory_PES_Packet is not matched \n");
        exit(0);
    }
    klok1 = (temp1 >> 3);
    prev_directory_offset_msb = klok1;
    PSD_Directory.prev_directory_offset_msb = prev_directory_offset_msb;

    prev_directory_offset_lsb = (temp1 & 0x6) << 29;

    /* 2 bytes */

```

```

temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("WARNING: Marker (3) in Directory_PES_Packet is not matched \n");
    exit(0);
}
clock1 = (temp1 & 0xfffe) << 14;
prev_directory_offset_lsb |= clock1;

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("WARNING: Marker (4) in Directory_PES_Packet is not matched \n");
    exit(0);
}
clock1 = (temp1 >> 1);
prev_directory_offset_lsb |= clock1;
PSD_Directory.prev_directory_offset_lsb = prev_directory_offset_lsb;

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("WARNING: Marker (5) in Directory_PES_Packet is not matched \n");
    exit(0);
}
clock1 = (temp1 >> 3);
next_directory_offset_msb = clock1;
PSD_Directory.next_directory_offset_msb = next_directory_offset_msb;

next_directory_offset_lsb = (temp1 & 0x6) << 29;

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("WARNING: Marker (6) in Directory_PES_Packet is not matched \n");
    exit(0);
}
clock1 = (temp1 & 0xfffe) << 14;
next_directory_offset_lsb |= clock1;

/* 2 bytes */
temp1 = 0;
for(i=0; i<2; i++) {
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];

```

```

    temp1 = (temp1 << 8) | temp;
}
if(!(temp1 & 0x1)) {
    printf("WARNING: Marker (7) in Directory_PES_Packet is not matched \n");
    exit(0);
}
clock1 = (temp1 >> 1);
next_directory_offset_lsb |= clock1;
PSD_Directory.next_directory_offset_lsb = next_directory_offset_lsb;

psd_AU_length = 18 * number_of_access_units;
for(cnt=0; cnt<number_of_access_units; cnt++) {
    PSD_Directory.PSD_AU_Info[cnt].packetStreamID = Decoder_buffer[bytes_out_Decoder_buffer++];

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("WARNING: Marker (1) in Directory_PES_Packet_Loop is not matched \n");
        exit(0);
    }
    PSD_Directory.PSD_AU_Info[cnt].PES_header_position_offset_sign = (temp1 & 0x8000) >> 15;
    clock1 = (temp1 & 0x7ff8) >> 3;
    PSD_Directory.PSD_AU_Info[cnt].PES_header_position_offset_msb = clock1;
    PSD_Directory.PSD_AU_Info[cnt].PES_header_position_offset_lsb = (temp1 & 0x6) << 29;

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("WARNING: Marker (2) in Directory_PES_Packet_Loop is not matched \n");
        exit(0);
    }
    clock1 = (temp1 & 0xfffe) << 14;
    PSD_Directory.PSD_AU_Info[cnt].PES_header_position_offset_lsb |= clock1;

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("WARNING: Marker (3) in Directory_PES_Packet_Loop is not matched \n");
        exit(0);
    }
    clock1 = (temp1 >> 1);
    PSD_Directory.PSD_AU_Info[cnt].PES_header_position_offset_lsb |= clock1;

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];

```

```

        temp1 = (temp1 << 8) | temp;
    }
    PSD_Directory.PSD_AU_Info[cnt].reference_offset = temp1;

    /* next byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    if((temp & 0x80) != 0x80) {
        printf("PSD:Marker-bit (1) in PTS is wrong \n");
        exit(0);
    }
    PSD_Directory.PSD_AU_Info[cnt].msb_PTS = (temp & 0x08) >> 3;
    PSD_Directory.PSD_AU_Info[cnt].lsb_PTS = (temp & 0x06) << 29;
    if(!(temp & 0x1)) {
        printf("PSD:Marker-bit (2) in PTS is wrong \n");
        exit(1);
    }

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("PSD:Marker-bit (3) in PTS is wrong \n");
        exit(1);
    }
    PSD_Directory.PSD_AU_Info[cnt].lsb_PTS |= ((temp1 & 0xfffe) << 14);

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("PSD: Marker-bit (4) in PTS is wrong \n");
        exit(1);
    }
    PSD_Directory.PSD_AU_Info[cnt].lsb_PTS |= (temp1 >> 1);
    if(trace) printf("PTS = %0x\n", PSD_Directory.PSD_AU_Info[cnt].msb_PTS, PSD_Directory.PSD_AU_Info[cnt].lsb_PTS);

    /* 2 bytes */
    temp1 = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        temp1 = (temp1 << 8) | temp;
    }
    if(!(temp1 & 0x1)) {
        printf("Marker-bit (1) in bytes_to_read is wrong \n");
        exit(1);
    }
    PSD_Directory.PSD_AU_Info[cnt].bytes_to_read = ((temp1 & 0xfffe) <<
7) | Decoder_buffer[bytes_out_Decoder_buffer++];
    if(trace) printf("bytes_to_read = %d\n", PSD_Directory.PSD_AU_Info[cnt].bytes_to_read);

    /* next byte */
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];

```

```

        if((temp & 0x80) != 0x80) {
            printf("WARNING: Marker-bit at the end of PSD_Packet is wrong \n");
            exit(0);
        }
        PSD_Directory.PSD_AU_Info[cnt].intra_coded_indicator = (temp >> 6) & 0x1;
        PSD_Directory.PSD_AU_Info[cnt].coding_parameters_indicator = (temp >> 4) & 0x3;
    } /* End of PSD_Loop */
}

/*****
PS_Demultiplexor() block is clocked once for each byte that comes
from the Decoder.
the state of the Demultiplexor to look for the next expected syntactic
element.
*****/
The states are as follows:
a) IDENTIFY_START_CODE: finds a start code,determines the activities
    associated with that state, eg., parsing the Pack_header,
    Packet_header, etc, and then sets any of the next two states
    accordingly.
b) AUDIO_PACKET: transfer bytes from the Decoder to the audio decoder file.
c) VIDEO_PACKET: transfer bytes from the Decoder to the video decoder file.
*****/
void PS_Demultiplexor(trace,pktLengthTS)
int trace,*pktLengthTS;
{
    unsigned int temp,temp1,streamID,scount,packet_length;
    static unsigned int p1=0,p2=0,p3=0,p4=0;
    int i,strmNo,pktLength,zero_count;

    if(trace) printf("PS_Demultiplexor Loop: P_state = %d \n",P_State);

    switch (P_State) {

        case IDENTIFY_START_CODE:
            scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
            zero_count = 0 - 1;
            if (scount < 10) break; /* maximum size (!) = 10 */

            if((temp = Decoder_buffer[bytes_out_Decoder_buffer++]) == 0) { /*1 00 */
                if((temp = Decoder_buffer[bytes_out_Decoder_buffer++]) == 0) { /*2 00 00 */

                    do {
                        zero_count += 1;
                        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
                    } while(temp == 0); /* again 00 */

                    if(temp == 1) { /*3 00 00 01 */
                        streamID = Decoder_buffer[bytes_out_Decoder_buffer++];
                        if(streamID <= 0xb8) { /* 00 to b8 */
                            printf("WARNING: Wrong start code in System stream \n");
                            printf("    These start codes belong to Video stream \n");
                            exit(1);
                        }
                    }

                    if(streamID == 0xbd)
                        PES_stream_type = PVT1;
                }
            }
        }
    }

```

```

if((streamID >= 0xc0) && (streamID <= 0xdf)) {
    strmNo = streamID - 0xc0;
    if(trace)
        printf("Packet Start: Audio Stream-%d Code(%d) \n",
            strmNo,p3++);
    streamID = 0xc0;
    PES_stream_type = AUDIO;
}
if((streamID >= 0xe0) && (streamID <= 0xef)) {
    strmNo = streamID - 0xe0;
    if(trace)
        printf("Packet Start: Video Stream-%d Code(%d) \n",
            strmNo,p4++);
    streamID = 0xe0;
    PES_stream_type = VIDEO;
}
switch(streamID) {

    case 0xb9: {
        iso_sequence_end = 1;
        printf("MPEG_program_end_code ! \n");
        P_State = IDENTIFY_START_CODE;
        break;
    }

    case 0xba: {
        if(trace)
            printf("Pack Start Code (%d) \n",p1++);
        P_State = PACK_HEADER;
        break;
    }

    case 0xbb: {
        if(trace)
            printf("system_header_start_code (%d) \n",p2++);
        /* system_header_length */
        system_header_length = 0;
        for(i=0; i<2; i++) {
            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            system_header_length = (system_header_length << 8) | temp;
        }
        if(trace)
            printf("system_header_length=%d\n",system_header_length);

        P_State = SYSTEM_HEADER;
        break;
    }

    case 0xbc: {
        if(trace)
            printf("Packet Start: program_stream_map: \n");
        /* program_stream_map_length */
        program_stream_map_length = 0;
        for(i=0; i<2; i++) {
            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            program_stream_map_length = (program_stream_map_length << 8) | temp;
        }
        if(trace)
            printf("program_stream_map_length=%d\n",program_stream_map_length);
    }
}

```



```

    P_State = PROGRAM_STREAM_MAP;
        break;
    }

case 0xbe: {
    if(trace) printf("Packet Start: padding stream \n");
    /* header_length */
    padding_packet_length = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        padding_packet_length = (padding_packet_length << 8) | temp;
    }
    if(trace)
        printf("padding_packet_length=%d\n",padding_packet_length);
    P_State = PADDING_PACKET;
    break;
}

case 0xbf: {
    if(trace) printf("Packet Start: private stream_2 \n");
    PES_stream_type = PVT2;
    /* header_length */
    pvtStream2_packet_length = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        pvtStream2_packet_length = (pvtStream2_packet_length << 8) | temp;
    }
    if(trace)
        printf("pvtStream2_packet_length=%d\n",pvtStream2_packet_length);
        (*pktLengthTS) -= (6 + zero_count);
    P_State = PRIVATE_2_PACKET;
        break;
}

case 0xbd:
case 0xc0:
case 0xe0: {
    /* packet_length */
    PES_packet_length = 0;
    for(i=0; i<2; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        PES_packet_length = (PES_packet_length << 8) | temp;
    }
    if(trace)
        printf("PES_packet_length=%d\n",PES_packet_length);

    if(PES_stream_type == VIDEO) {
        if(!PES_packet_length) {
            printf("Zero Video_PES_Packet_Length.... \n");
            specialVideoCase = TRUE;
        }
        else
            specialVideoCase = FALSE;
    }

    else if(!PES_packet_length) {
        printf("Zero PES_Packet_Length is not allowed.\n");
        exit(0);
    }
}

```

```

        (*pktLengthTS) -= (6 + zero_count);
        P_State = PES_PACKET_HEADER;
        break;
    }

    case 0xf2: {
        if(trace)
            printf("Packet Start: DSM_CC Decoder Data: \n");
        /* dsmcc_command_id */
        dsmcc_command_id = Decoder_buffer[bytes_out_Decoder_buffer++];
        if(trace)
            printf("dsmcc_command_id=%d\n", dsmcc_command_id);

        P_State = DSM_CC;
        break;
    }

    case 0xff: {
        if(trace)
            printf("Packet Start: program stream directory ....\n");
        /* ps_directory_payload_length */
        ps_directory_packet_length = 0;
        for(i=0; i<2; i++) {
            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            ps_directory_packet_length = (ps_directory_packet_length << 8) | temp;
        }
        if(trace)
            printf("ps_directory_packet_length=%d\n", ps_directory_packet_length);

        P_State = PS_DIRECTORY;
        break;
    }

    default: {
        printf("ECM,EMM,MHEG or reserved streams\n");
        printf(" Not handled now ... \n");
        exit();
        break;
    }

    } /* end of big switch */
} } /* 1,2,3 */

break; /* IDENTIFY_START_CODE */

case PACK_HEADER:
    /* Handles Pack-header */
    scout = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scout < 18) break; /* max #bytes = 18 */

    pack_header(trace);
    P_State = IDENTIFY_START_CODE;
    break;

case SYSTEM_HEADER:
    /* Handles System-header */
    scout = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scout < system_header_length) break;

```

```

    system_header(trace);
    P_State = IDENTIFY_START_CODE;
    break;

case PROGRAM_STREAM_MAP:
    /* Handles program-stream-map */
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < program_stream_map_length) break;

    program_stream_map(trace);
    P_State = IDENTIFY_START_CODE;
    break;

case PRIVATE_1_PACKET:
    /* Handles Private-1 bitstream */
#ifdef TS_STREAM
    pktLength = *pktLengthTS;
#endif
#ifdef PS_STREAM
    pktLength = PacketSize;
#endif
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < pktLength) break;

    if(PacketSize < pktLength) pktLength = PacketSize;

    if(trace) printf("PRIVATE-1 STREAM IS STORED NOW - %d bytes\n",pktLength);
    for(i=0; i<pktLength; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        putc(temp,OutfileS1);
    }
    fflush(OutfileS1);

#ifdef TS_STREAM
    if(PacketSize > 0) {
        PacketSize -= pktLength;
        if(PacketSize > 0)
            P_State = PRIVATE_1_PACKET;
        else
            P_State = IDENTIFY_START_CODE;
    }
    else
        P_State = IDENTIFY_START_CODE;
#endif
#ifdef PS_STREAM
    P_State = IDENTIFY_START_CODE;
#endif
    break;

case PADDING_PACKET:
    /* Handles System-header */
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < padding_packet_length) break;

    /* Packet Start: Padding Stream */
    bytes_out_Decoder_buffer += padding_packet_length;

    P_State = IDENTIFY_START_CODE;

```

```

    break;

    case PRIVATE_2_PACKET:
        /* Handles System-header */
#ifdef TS_STREAM
        pktLength = *pktLengthTS;
#endif
#ifdef PS_STREAM
        pktLength = pvtStream2_packet_length;
#endif
        scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
        if (scount < pktLength) break;

        if(pvtStream2_packet_length < pktLength)
            pktLength = pvtStream2_packet_length;

        /* Packet Start: store private stream_2 */
        if(trace) printf("Private-2 Stream is stored - %d bytes\n",pktLength);
        for(i=0; i<pktLength; i++) {
            temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            putc(temp,OutfileS2);
        }
        fflush(OutfileS2);

#ifdef TS_STREAM
        if(pvtStream2_packet_length > 0) {
            pvtStream2_packet_length -= pktLength;
            if(pvtStream2_packet_length > 0)
                P_State = PRIVATE_2_PACKET;
            else
                P_State = IDENTIFY_START_CODE;
        }
        else
            P_State = IDENTIFY_START_CODE;
#endif
#ifdef PS_STREAM
        P_State = IDENTIFY_START_CODE;
#endif
        break;

    case DSM_CC:
        /* Handles DSM_CC */
        scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
        if (scount < 12) break;

        /* DSM_CC Decoder */
        bytes_out_System_buffer = bytes_out_Decoder_buffer;
        System_buffer = Decoder_buffer;

        DSM_CC_Decoder(trace);

        bytes_out_Decoder_buffer = bytes_out_System_buffer;
        Decoder_buffer = System_buffer;

        P_State = IDENTIFY_START_CODE;
        break;

    case PES_PACKET_HEADER:

```

```

/* Handles PES_Packet-header */
scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
if(trace) printf("IS BYTE_SIZE_DECODER_BUFFER = %d > 260 ?\n",scount);
/* Max number of bytes in PES_packet Header */
if (scount < 260) break;

pktLength = *pktLengthTS;
PES_packet(trace, &pktLength);
*pktLengthTS = pktLength;

if(PES_stream_type == AUDIO)
    P_State = AUDIO_PACKET;
else if(PES_stream_type == VIDEO)
    P_State = VIDEO_PACKET;
else if(PES_stream_type == PVT1)
    P_State = PRIVATE_1_PACKET;
break;

case AUDIO_PACKET:
    /* Handles Audio bitstream */
#ifdef TS_STREAM
    pktLength = *pktLengthTS;
#endif
#ifdef PS_STREAM
    pktLength = PacketSize;
#endif
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < pktLength) break;

    if(PacketSize < pktLength)
        pktLength = PacketSize;

    if(trace) printf("AUDIO-DATA IS STORED NOW - %d bytes\n",pktLength);

    for(i=0; i<pktLength; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        putc(temp,OutfileA);
    }
    fflush(OutfileA);
#ifdef TS_STREAM
    if(PacketSize > 0) {
        PacketSize -= pktLength;
        if(PacketSize > 0)
            P_State = AUDIO_PACKET;
        else
            P_State = IDENTIFY_START_CODE;
    }
    else
        P_State = IDENTIFY_START_CODE;
#endif
#ifdef PS_STREAM
    P_State = IDENTIFY_START_CODE;
#endif
    break;

case VIDEO_PACKET:
    /* Handles Video bitstream */
#ifdef TS_STREAM
    pktLength = *pktLengthTS;

```

```

#endif
#ifdef PS_STREAM
    pktLength = PacketSize;
#endif
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < pktLength) break;

    if ((!specialVideoCase) && (PacketSize < pktLength))
        pktLength = PacketSize;

    if (trace) printf("Video-data is stored now - %d bytes\n", pktLength);
    for (i=0; i<pktLength; i++) {
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        putc(temp, OutfileV);
    }
    fflush(OutfileV);
#ifdef TS_STREAM
    if (specialVideoCase)
        P_State = VIDEO_PACKET;
    else if (PacketSize > 0) {
        PacketSize -= pktLength;
        if (PacketSize > 0)
            P_State = VIDEO_PACKET;
        else {
            P_State = IDENTIFY_START_CODE;
            videoStart = 0;
        }
    }
    else {
        P_State = IDENTIFY_START_CODE;
        videoStart = 0;
    }
#endif
#ifdef PS_STREAM
    P_State = IDENTIFY_START_CODE;
#endif
    break;

case PS_DIRECTORY:
    /* Handles System-header */
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < ps_directory_packet_length) break;

    /* Directory_PES_packet */
    PS_directory_PES_packet(trace);
    P_State = IDENTIFY_START_CODE;
    break;

default:
    P_State = IDENTIFY_START_CODE;
    break;

} /* end of P_State switch */

}

```

## A.8 mpeg2sys.c

```

/*****
 * mpeg2Sys.c: MPEG-II Composite Transport Stream Bitstream Parser: *
 * The main Program Section. The program-exit for the Transport Stream *
 * is not defined, so exit from the big "DO LOOP" is primitive here. *
 * SO the last few hundred bytes of the Video/Audio files will be *
 * incorrect in some cases. *
 *****/
 * Originator: Munsu A. Haque; Date: September 13th, 1994. *
 * Hyundai Electronics America, San Jose, CA. *
 * e-mail: munsu@heava.com *
 *****/
*/
#include <stdio.h>
#include <math.h>
#include <malloc.h>

#include "mpeg2Sys.h"

#define debugM 0

/* Just print a bunch of data for debugging */
void printData(buffer,start,count)
unsigned char * buffer;
int start,count;
{
    int i,j,k,m;
    unsigned char locbuf[128];

    j = 0;
    for(i=start; i<(start+count); i++) {
        locbuf[j++] = buffer[i];
    }

    m = 0;
    for(i=0; i<(count/4); i++) {
        for(j=0; j<4; j++) {
            k = locbuf[m++];
            printf(" %0x ",k);
        }
        printf("\n");
    }
}

/*****
 *****/
/***** Data Transfer from DSM *****/
/*****
 */
/* The DSM block puts a byte into buffer each time it is clocked. */
void DSM(trace)
int trace;
{
    int i,temp;
    if( (bytes_in_DSM_buffer - bytes_out_DSM_buffer) < 4096 )
        for(i=0; i<4096; i++) {

```

```

        temp = getc(Infile);
#ifdef TS_STREAM
        if(temp == EOF) {
            end_of_TS_loop = 1;
            printf("_____END_OF_FILE_____\n");
            break;
        }
#endif
        DSM_buffer[bytes_in_DSM_buffer++] = temp;
    }
    if (trace && debugM) {
        printf("DSM: bytes_in_DSM_buffer=%ld ", bytes_in_DSM_buffer);
        printf("and bytes_out_DSM_buffer=%ld\n", bytes_out_DSM_buffer);
    }
}

/*****
 * Readjust the DSM buffer data-pointer so that the array
 * starts from 0 location after the data-bytes are extracted
 * from the buffer.
 *****/
void adjust_DSM_buffer()
{
    unsigned short int i,j=0;

    for(i=bytes_out_DSM_buffer; i<bytes_in_DSM_buffer; i++)
        DSM_buffer[j++] = DSM_buffer[i];
    bytes_in_DSM_buffer = j;
    bytes_out_DSM_buffer = 0;
}

/*****
 *                               *
 *                               *
 *****/
void main(argc,argv)
int argc;
char **argv;
{
    char infile[128],outfile1[128],outfile2[128];
    char outfile3[128],outfile4[128];
    char *fname;
    int i,j,temp,dataCnt,trace = FALSE;

    if(argc < 1) {
#ifdef TS_STREAM
        printf("Command is: mpeg2Ts <input_filename> \n");
#endif
#ifdef PS_STREAM
        printf("Command is: mpeg2Ps <input_filename> \n");
#endif
        exit(0);
    }

    fname = argv[1];

    /* Open the Input MPEG Composite Bitstream File */
    sprintf(infile,"%s",fname);
    Infile = fopen(infile,"r");

```



```

/* Open the Output Audio and Video Bit_stream Files */
sprintf(outfile1,"%s.audio",fname);
OutfileA = fopen(outfile1,"w");
sprintf(outfile2,"%s.video",fname);
OutfileV = fopen(outfile2,"w");

/* Open the Output Stream_1 Files */
sprintf(outfile3,"%s.strm1",fname);
OutfileS1 = fopen(outfile3,"w");

/* Open the Output Stream_2 Files */
sprintf(outfile4,"%s.strm2",fname);
OutfileS2 = fopen(outfile4,"w");

DSM_buffer = (unsigned char *) malloc(DSM_SIZE);
specialVideoCase = FALSE;
bytes_in_DSM_buffer = 0;
bytes_out_DSM_buffer = 0;
bytes_in_System_buffer = 0;
bytes_out_System_buffer = 0;

#ifdef TS_STREAM
/* Start MPEG System Demultiplexor */
System_buffer = (unsigned char *) malloc(SBUFFER_SIZE);
videoBuffer = &Video_buffer[0];
audioBuffer = &Audio_buffer[0];
pvt1Buffer = &Private1_buffer[0];
pvt2Buffer = &Private2_buffer[0];
byteCtr = 0;
videoStart = 0;
audioStart = 0;
pvt1Start = 0;
pvt2Start = 0;
bytes_in_Video_buffer = 0;
bytes_out_Video_buffer = 0;
bytes_in_Audio_buffer = 0;
bytes_out_Audio_buffer = 0;
bytes_in_Private1_buffer = 0;
bytes_out_Private1_buffer = 0;
bytes_in_Private2_buffer = 0;
bytes_out_Private2_buffer = 0;
end_of_TS_loop = 0;

/* Select one program for storing into a file */
user_interface_for_main(trace);

A_State = IDENTIFY_START_CODE;
V_State = IDENTIFY_START_CODE;
PVT1_State = IDENTIFY_START_CODE;
PVT2_State = IDENTIFY_START_CODE;

do {
    DSM(trace);
    TS_Demultiplexor(trace);
    dataCnt = bytes_in_DSM_buffer - bytes_out_DSM_buffer;

    } while((!end_of_TS_loop) || dataCnt);

```

```

if(trace) printf("Here end_of_TS_loop is reached based on EOF during reading - no end_of_TS_sequence
flag..\n");

```

```

/* Needs code_improvement on how to handle the last few PES Packets ?? */
dataCnt = bytes_in_Video_buffer - bytes_out_Video_buffer;
if(trace) printf("Now dump remaining (%d) Video_data in the buffers..\n",dataCnt);
dataCnt = bytes_in_Audio_buffer - bytes_out_Audio_buffer;
if(trace) printf("Now dump remaining (%d) Audio_data in the buffers..\n",dataCnt);
dataCnt = bytes_in_Private1_buffer - bytes_out_Private1_buffer;
if(trace) printf("Now dump remaining (%d) Private1_data in the buffers..\n",dataCnt);
dataCnt = bytes_in_Private2_buffer - bytes_out_Private2_buffer;
if(trace) printf("Now dump remaining (%d) Private2_data in the buffers..\n",dataCnt);

```

```

/*
for(i=bytes_out_Video_buffer; i<bytes_in_Video_buffer; i++) {
    temp = Video_buffer[i];
    putc(temp,OutfileV);
}

for(i=bytes_out_Audio_buffer; i<bytes_in_Audio_buffer; i++) {
    temp = Audio_buffer[i];
    putc(temp,OutfileA);
}

for(i=bytes_out_Private1_buffer; i<bytes_in_Private1_buffer; i++) {
    temp = Private1_buffer[i];
    putc(temp,OutfileS1);
}

for(i=bytes_out_Private2_buffer; i<bytes_in_Private2_buffer; i++) {
    temp = Private2_buffer[i];
    putc(temp,OutfileS2);
}
*/

```

```

*/
#endif

```

```

#ifdef PS_STREAM
iso_sequence_end = 0;
P_State = IDENTIFY_START_CODE;

do {
    DSM(trace);

    bytes_in_Decoder_buffer = bytes_in_DSM_buffer;
    bytes_out_Decoder_buffer = bytes_out_DSM_buffer;
    Decoder_buffer = DSM_buffer;

    PS_Demultiplexor(trace,&dataCnt);

    bytes_in_DSM_buffer = bytes_in_Decoder_buffer;
    bytes_out_DSM_buffer = bytes_out_Decoder_buffer;

    adjust_DSM_buffer();

} while(!iso_sequence_end);

```

```

#endif

```

```

fclose(Infile);

```

```

fclose(OutfileV);
fclose(OutfileA);
fclose(OutfileS1);
fclose(OutfileS2);
}

```

#### A.10 mpeg2sys.h

```

/* mpeg2Sys.h: MPEG-II Transport & Program stream definition      *
 * Some of these global variables are kept as global, only to      *
 * verify the simulation runs, may not be needed ultimately.      *
 * *****                                                        *
 * Originator: Munsir A. Haque; Date: September 13th, 1994.      *
 * Hyundai Electronics America, San Jose, CA.                    *
 * e-mail: munsir@heava.com                                       *
 * *****                                                        *
 */
#define TS_STREAM          /* Transport Stream Definition */
/* #define PS_STREAM      /* Program Stream Definition */

#ifndef TS_STREAM

#define SYNC_BYTE          0x47

#define PAS                1
#define CAS                2
#define NIS                3
#define PMAP              4
#define PRIVATE            5
#define PSI_TS_PKT_DONE   6

#define PSI_DATA           10
#define DSM_CC_TS         11
#define VIDEO_STREAM      12
#define AUDIO_STREAM      13
#define PRIVATE_STREAM_1  14
#define PRIVATE_STREAM_2  15
#define TRASHED_STREAM    16
#endif

#define IDENTIFY_START_CODE 1
#define PACK_HEADER         2
#define SYSTEM_HEADER       3
#define PROGRAM_STREAM_MAP  4
#define PRIVATE_1_PACKET    5
#define PADDING_PACKET      6
#define PRIVATE_2_PACKET    7
#define PES_PACKET_HEADER   8
#define AUDIO_PACKET        9
#define VIDEO_PACKET       10
#define PS_DIRECTORY        11
#define DSM_CC              12

#define AUDIO              1
#define VIDEO              2

```

```

#define SYSTEM          3
#define PVT1            4
#define PVT2            5
#define PADD_STRM       6

#define DSM_SIZE        40000
#define SBUFFER_SIZE    20000

#ifdef TS_STREAM
#define ABUFFER_SIZE    40000
#define VBUFFER_SIZE    80000
#define PBUFFER_SIZE    20000
/* Size Limit of the Private Data Bytes in Private Table (PSI) */
#define PVT_TBL_SIZE    1024

#endif

#define TRUE            1
#define FALSE          0

int video_PES_packet_length, audio_PES_packet_length, padding_packet_length;
int pvt1_PES_packet_length, pvtStream2_packet_length, PacketSize;
int ps_directory_packet_length, PES_packet_length;
int program_stream_map_length, elementary_stream_info_length;
int program_stream_info_length, elementary_stream_map_length;

unsigned int iso_sequence_end, program_select, PES_stream_type;
unsigned int PES_scrambling_control, PES_priority, copyright;
unsigned int data_alignment_indicator, original_or_copy;
unsigned int PTS_DTS_flags, ESCR_flag, ES_rate_flag, trick_mode_control;
unsigned int DSM_trick_mode_flag, additional_copy_info_flag;
unsigned int PES_CRC_flag, PES_extension_flag, PES_header_data_length;
unsigned int frequency_truncation, field_rep_cntrl, additional_copy_info;
unsigned int previous_PES_packet_CRC, PES_private_data_flag;
unsigned int pack_header_field_flag, PES_extension_flag_2;
unsigned int system_header_length, P_STD_buffer_flag, PES_private_data[16];
unsigned int pack_field_length, program_packet_sequence_counter_flag;
unsigned int P_STD_buffer_flag, MPEG1_MPEG2_identifier;
unsigned int program_packet_sequence_counter, original_stuff_length;
unsigned int psd_AU_length, P_STD_buffer_scale, P_STD_buffer_size;
unsigned int PES_extension_field_length, intra_slice_refresh;
unsigned int ESCR_extension, ES_rate, field_id, audio_bound;
unsigned int system_clock_reference_base, system_clock_reference_extension;
unsigned int program_mux_rate, pack_stuffing_length, rate_bound;
unsigned int fixed_flag, CSPS_flag, video_bound, current_next_indicator;
unsigned int system_audio_lock_flag, system_video_lock_flag, stream_id;
unsigned int P_STD_buffer_bound_scale, P_STD_buffer_size_bound, CAT_PID;
unsigned int stream_type, elementary_stream_id, program_stream_map_versrion;
unsigned int number_of_access_units, packet_stream_Id, reference_offset;
unsigned int bytes_to_read, intra_coded_indicator, coding_parameters_indicators;
unsigned char msb_PTS, msb_DTS, msb_ESCR, msb_SCRB, end_of_TS_loop;
unsigned char P_State, specialVideoCase;
unsigned long int PTS, DTS, ESCR, CRC_32;
long int bytes_in_DSM_buffer, bytes_out_DSM_buffer;
long int bytes_in_Decoder_buffer, bytes_out_Decoder_buffer;
long int bytes_in_System_buffer, bytes_out_System_buffer;
unsigned char *DSM_buffer, *Decoder_buffer, *System_buffer;

#ifdef TS_STREAM

```

```

int programCount,programNO,pointer_field,section_length,pas_section_length;
int videoStart,audioStart,pvt1Start,pvt2Start,cas_section_length;
int videoPacketSize,audioPacketSize,pvt1PacketSize,pmap_section_length;
int private_section_length,ES_info_length,program_info_length;
int pas_section_loop_length,pmap_section_loop_length;
int cas_section_loop_length,pvt_section_loop_length;
int pmap_program_info_length,nit_section_length,nit_section_loop_length;

unsigned char A_State,V_State,PVT1_State,PVT2_State;
unsigned int adaptation_field_length,discontinuity_indicator;
unsigned int random_access_indicator,elementary_stream_priority_indicator;
unsigned int PCR_flag,OPCR_flag,splicing_point_flag,msb_PCRB,msb_OPCRB;
unsigned int transport_private_data_flag,adaptation_field_extension_flag;
unsigned int program_clock_reference_base,program_clock_reference_extension;
unsigned int original_program_clock_reference_base,splice_countdown;
unsigned int original_program_clock_reference_extension;
unsigned char adfield_private_data_byte[183],pvtsec_private_data_byte[1024];
unsigned char nitsec_private_data_byte[1024];
unsigned int transport_private_data_length,adaptation_field_extension_length;
unsigned int stuffing_byte,videoPIDselect,audioPIDselect,table_id;
unsigned int nitPIDselect,programPIDselect,pvt1PIDselect,pvt2PIDselect;
unsigned int section_syntax_indicator,transport_stream_id;
unsigned int PAS_version_number,PA_section_number,PA_last_section_number;
unsigned int CAS_version_number,CA_section_number,CA_last_section_number;
unsigned int PMAP_version_number,PMAP_section_number,PMAP_last_section_number;
unsigned int private_version_number,private_section_number;
unsigned int pas_data_ptr,pas_version_number,private_last_section_number;
unsigned int pas_section_number,pas_last_section_number,pas_crc_ptr;
unsigned int cas_data_ptr,cas_version_number,PCR_PID;
unsigned int cas_section_number,cas_last_section_number,cas_crc_ptr;
unsigned int pmap_data_ptr,pmap_version_number;
unsigned int pmap_section_number,pmap_last_section_number,pmap_crc_ptr;
unsigned int pvt_version_number,pvt_section_number,pvt_last_section_number;
unsigned int pvt_section_syntax_indicator,private_indicator;
unsigned int table_id_extension,pvt_current_next_indicator;
unsigned int pvt_data_ptr,pvt_crc_ptr,elementary_PID,dsmccPIDselect;
unsigned int nit_data_ptr,nit_indicator,nit_version_number,nit_section_number;
unsigned int nit_current_next_indicator,nit_section_number,nit_last_section_number;
unsigned int transport_error_indicator,payload_unit_start_indicator;
unsigned int transport_priority,transport_scrambling_control;
unsigned int adaptation_field_control,continuity_counter;
unsigned int ltw_flag,piecewise_rate_flag,seamless_splice_flag;
unsigned int ltw_valid_flag,ltw_offset,piecewise_rate,splice_type;
unsigned int DTS_next_au_msb,DTS_next_au_lsb;
unsigned int cas_current_next_indicator,pvt_section_syntax_indicator;
unsigned int nit_crc_ptr,input_data_ptr,crc_data_ptr;
unsigned int pvt_indicator,pvt_section_length;

int tByteCnt,v1Cnt,v2Cnt,a1Cnt,a2Cnt;
unsigned int video1_PID[16],video2_PID[16],audio1_PID[32],audio2_PID[32];
unsigned int cas_PID,pvt1_PID,pvt2_PID,dsm_CC_PID;
unsigned int PAT_State,CAT_State,NIT_State,PMAP_State,PVT_State;
unsigned int patData[32],catData[32],pmapData[32],nitData[32],pvtData[32];

long int byteCtr,bytes_in_Video_buffer,bytes_out_Video_buffer;
long int bytes_in_Audio_buffer,bytes_out_Audio_buffer;
long int bytes_in_Private1_buffer,bytes_out_Private1_buffer;
long int bytes_in_Private2_buffer,bytes_out_Private2_buffer;

```

```

unsigned char Video_buffer[VBUFFER_SIZE],Audio_buffer[ABUFFER_SIZE];
unsigned char Private1_buffer[PBUFFER_SIZE],Private2_buffer[PBUFFER_SIZE];
unsigned char *videoBuffer,*audioBuffer,*pvt1Buffer,*pvt2Buffer;

```

```

typedef struct {
    int programNumber;
    unsigned int PID;
    } programMap;
programMap    programMAP[64];

```

```

#endif

```

```

typedef struct {
    unsigned int packetStreamID;
    unsigned int PES_header_position_offset_sign;
    unsigned int PES_header_position_offset_msb;
    unsigned int PES_header_position_offset_lsb;
    unsigned int reference_offset;
    unsigned int msb_PTS;
    unsigned int lsb_PTS;
    unsigned int bytes_to_read;
    unsigned int intra_coded_indicator;
    unsigned int coding_parameters_indicator;
    } psd_au;

```

```

/* assumed maximum access_unit number = 16 */

```

```

typedef struct {
    unsigned int number_of_access_units;
    unsigned int prev_directory_offset_msb;
    unsigned int prev_directory_offset_lsb;
    unsigned int next_directory_offset_msb;
    unsigned int next_directory_offset_lsb;
    psd_au PSD_AU_Info[16];
    } psd_directory;

```

```

psd_directory PSD_Directory;

```

```

FILE *Infile,*OutfileA,*OutfileV,*OutfileS1,*OutfileS2;

```

## A.11 mpeg2tr.c

```

/*****
 * mpeg2Tr.c: MPEG-II Composite Transport Stream Bitstream Parser:  *
 *****/
WARNING: Current Codes handle only one Private Section and NIT
at the same time in the same or multiple consecutive TS_packets.
For multiple Private Tables, we need multiple copies of the Private
Sections as done for the NIT.
*****/
* Originator: Munsu A. Haque; Date: September 13th, 1994.      *
* Hyundai Electronics America, San Jose, CA.                  *
* e-mail: munsu@heava.com                                       *
 *****/
*/
#include <stdio.h>

```

```

#include <math.h>

#include "mpeg2Sys.h"
#include "commonDmux.h"

unsigned int past_nit_size, past_pvt_size;

/* #define CHANNEL_SWITCH    /* Channel Hopping case OFF now */
static int initSystem = 0;
#ifdef TS_STREAM

#define debugS                0

/*****
***** TRANSPORT STREAM *****/
/*****/

* Readjust the System buffer data-pointer so that the array
* starts from 0 location after the data-bytes are extracted
* from the buffer.
*****/
void adjust_System_buffer()
{
    unsigned int i,j=0;

    bytes_in_System_buffer = 0;
    bytes_out_System_buffer = 0;
}

/*****
* Readjust the Video buffer data-pointer so that the array
* starts from 0 location after the data-bytes are extracted
* from the buffer.
*****/
void adjust_Video_buffer()
{
    unsigned int i,j=0;
    if(debugS)
        printf("adjust_before:bytes_in_Video_buffer=%d\n",bytes_in_Video_buffer);
    if(debugS)
        printf(" :bytes_out_Video_buffer=%d\n",bytes_out_Video_buffer);
    for(i=bytes_out_Video_buffer; i<bytes_in_Video_buffer; i++)
        Video_buffer[j++] = Video_buffer[i];
    bytes_in_Video_buffer = j;
    bytes_out_Video_buffer = 0;
    if(debugS)
        printf("adjust_after:bytes_in_Video_buffer=%d\n",bytes_in_Video_buffer);
    if(debugS)
        printf(" :bytes_out_Video_buffer=%d\n",bytes_out_Video_buffer);
}

/*****
* Readjust the Audio buffer data-pointer so that the array
* starts from 0 location after the data-bytes are extracted
* from the buffer.
*****/
void adjust_Audio_buffer()
{
    unsigned int i,j=0;

```

```

for(i=bytes_out_Audio_buffer; i<bytes_in_Audio_buffer; i++)
    Audio_buffer[j++] = Audio_buffer[i];
bytes_in_Audio_buffer = j;
bytes_out_Audio_buffer = 0;
}

/*****
* Readjust the Private buffers data-pointer so that the array
* starts from 0 location after the data-bytes are extracted
* from the buffer.
*****/
void adjust_Private1_buffer()
{
    unsigned int i,j=0;

    for(i=bytes_out_Private1_buffer; i<bytes_in_Private1_buffer; i++)
        Private1_buffer[j++] = Private1_buffer[i];
    bytes_in_Private1_buffer = j;
    bytes_out_Private1_buffer = 0;
}

void adjust_Private2_buffer()
{
    unsigned int i,j=0;

    for(i=bytes_out_Private2_buffer; i<bytes_in_Private2_buffer; i++)
        Private2_buffer[j++] = Private2_buffer[i];
    bytes_in_Private2_buffer = j;
    bytes_out_Private2_buffer = 0;
}

/*****
*      adaptation_field      *
*****/
void adaptation_field(trace)
unsigned int trace;
{
    unsigned int i,temp;
    unsigned long int temp1;
    int N,n1,bCount;

    /* Get the 1st byte */
    adaptation_field_length = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    if(trace) printf("adaptation_field_length = %d\n",adaptation_field_length);

    /* Get the 1st byte */
    bCount = bytes_out_DSM_buffer;
    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    discontinuity_indicator = (temp >> 7);
    random_access_indicator = (temp >> 6) & 0x1;
    elementary_stream_priority_indicator = (temp >> 5) & 0x1;
    if(trace) printf("adaptation_flags=%0xh\n",(temp & 0x1f));
    PCR_flag = (temp >> 4) & 0x1;
    OPCR_flag = (temp >> 3) & 0x1;
    splicing_point_flag = (temp >> 2) & 0x1;
    transport_private_data_flag = (temp >> 1) & 0x1;

```



```

adaptation_field_extension_flag = temp & 0x1;

if(PCR_flag) {
    /* 4 bytes */
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
        temp1 = (temp1 << 8) | temp;
    }
    msb_PCRB = temp1 >> 31;
    program_clock_reference_base = temp1 << 1;

    /* Get another byte */
    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    program_clock_reference_base |= (temp >> 7);
    program_clock_reference_extension = (temp & 0x1) << 8;
    /* Get another byte */
    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    program_clock_reference_extension |= (temp >> 7);
    program_clock_reference_extension |= (temp & 0x1) << 8;

if(OPCR_flag) {
    /* 4 bytes */
    temp1 = 0;
    for(i=0; i<4; i++) {
        temp = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
        temp1 = (temp1 << 8) | temp;
    }
    msb_OPCRB = temp1 >> 31;
    original_program_clock_reference_base = temp1 << 1;

    /* Get another byte */
    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    original_program_clock_reference_base |= (temp >> 7);
    original_program_clock_reference_extension = (temp & 0x1) << 8;
    /* Get another byte */
    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    original_program_clock_reference_extension |= (temp >> 7);
    original_program_clock_reference_extension |= (temp & 0x1) << 8;

if(splicing_point_flag) {
    /* Get another byte */
    splice_countdown = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
}

if(transport_private_data_flag) {
    /* Get another byte */
    transport_private_data_length = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    for (i=0; i<transport_private_data_length; i++) {
        adfield_private_data_byte[i] = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    }
}

if(adaptation_field_extension_flag) {
    adaptation_field_extension_length = DSM_buffer[bytes_out_DSM_buffer++];

```

```

    byteCtr++;

    temp = DSM_buffer[bytes_out_DSM_buffer++];
    ltw_flag = (temp & 0x80) >> 7;
    piecewise_rate_flag = (temp & 0x40) >> 6;
    seamless_splice_flag = (temp & 0x20) >> 5;
    /* 5-bits are reserved */
    n1 = 1;

    if(ltw_flag) {
        temp = DSM_buffer[bytes_out_DSM_buffer++];
        ltw_valid_flag = (temp & 0x80) >> 7;
        ltw_offset = (temp & 0x7f) << 8;
        ltw_offset |= DSM_buffer[bytes_out_DSM_buffer++];
        n1 += 2;
    }

    if(piecewise_rate_flag) {
        temp = DSM_buffer[bytes_out_DSM_buffer++];
        piecewise_rate = (temp & 0x3f) << 16;
        piecewise_rate
            |= (DSM_buffer[bytes_out_DSM_buffer++] << 8);
        piecewise_rate
            |= DSM_buffer[bytes_out_DSM_buffer++];
        n1 += 3;
    }

    if(seamless_splice_flag) {
        temp = DSM_buffer[bytes_out_DSM_buffer++];
        splice_type = (temp & 0xf0) >> 4;
        DTS_next_au_msb = (temp & 0x08) >> 3;
        DTS_next_au_lsb = (temp & 0x06) << 30;
        if(!(temp & 0x1)) {
            printf("Error: Marker-bit (1) in DTS_next_au is wrong..\n");
            exit(0);
        }
        temp = (DSM_buffer[bytes_out_DSM_buffer++] << 8);
        temp |= DSM_buffer[bytes_out_DSM_buffer++];
        if(!(temp & 0x1)) {
            printf("Error: Marker-bit (2) in DTS_next_au is wrong..\n");
            exit(0);
        }
        DTS_next_au_lsb |= ((temp & 0xffff) << 14);

        temp = (DSM_buffer[bytes_out_DSM_buffer++] << 8);
        temp |= DSM_buffer[bytes_out_DSM_buffer++];
        if(!(temp & 0x1)) {
            printf("Error: Marker-bit (2) in DTS_next_au is wrong..\n");
            exit(0);
        }
        DTS_next_au_lsb |= ((temp & 0xffff) >> 1);

        n1 += 5;
    }

    n1 = adaptation_field_extension_length - n1;
    /* Reserved Bytes */
    bytes_out_DSM_buffer += n1;
    byteCtr += adaptation_field_extension_length;

```

```

    }

    bCount = bytes_out_DSM_buffer - bCount;
    N = adaptation_field_length - bCount;
    if(trace) printf("Number of stuffing bytes in adaptation_field = %d\n",N);
    for (i=0;i<N;i++){
        stuffing_byte = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
        if((stuffing_byte != 0xff) && (debugS))
            printf("Warning: Wrong stuffing byte type \n");
    }
}

unsigned int detect_PID(trace,PID,pidPsiType)
unsigned int trace,PID,*pidPsiType;
{
    unsigned int tType;

    /* PIDs for PAS - 0 and CAS - 1 */
    /* PIDs for NIT (program_number=0) and PMAP are user-selected */
    if(PID == 0) {
        tType = PSI_DATA;
        *pidPsiType = PAS;
    }
    else if(PID == 1) {
        tType = PSI_DATA;
        *pidPsiType = CAS;
    }
    else if (PID == programPIDselect) {
        tType = PSI_DATA;
        *pidPsiType = PMAP;
    }
    else if (PID == nitPIDselect) {
        tType = PSI_DATA;
        *pidPsiType = NIS;
    }
    else if (PID == cas_PID) {
        tType = PSI_DATA;
        *pidPsiType = PRIVATE;
    }
    else if (PID == dsmccPIDselect) {
        tType = DSM_CC;
        *pidPsiType = DSM_CC;
    }
    else if (PID == videoPIDselect) {
        tType = VIDEO_STREAM;
        *pidPsiType = VIDEO_STREAM;
    }
    else if (PID == audioPIDselect) {
        tType = AUDIO_STREAM;
        *pidPsiType = AUDIO_STREAM;
    }
    else if (PID == pvt1PIDselect) {
        tType = PRIVATE_STREAM_1;
        *pidPsiType = PRIVATE_STREAM_1;
    }
    else if (PID == pvt2PIDselect) {
        tType = PRIVATE_STREAM_2;
        *pidPsiType = PRIVATE_STREAM_2;
    }
}

```

```

    }
    else if( PID == 0x1fff) {
        tType = TRASHED_STREAM;
        *pidPsiType = TRASHED_STREAM;
    }
    else {
        if(trace) printf("Note: Streams with unknown PIDs (%0x) are present ..\n",PID);
        tType = TRASHED_STREAM;
        *pidPsiType = TRASHED_STREAM;
    }
    return(tType);
}

```

```
unsigned int detect_TableID(table_id)
```

```

unsigned int table_id;
{
    unsigned int psiType;

    if( table_id == 0 )
        psiType = PAS;
    else if( table_id == 1 )
        psiType = CAS;
    else if( table_id == 2 )
        psiType = PMAP;
    else if ( (table_id >= 0x40) && (table_id < 0xff) )
        psiType = PRIVATE;
    else if( table_id == 0xff )
        psiType = PSI_TS_PKT_DONE;
    else {
        printf("Wrong table_id=%0x h is found ...\n",table_id);
        psiType = PRIVATE; /* Should be reserved ??? */
    }
    return(psiType);
}

```

```
unsigned int getSplicedByte(dataBuffer)
```

```

unsigned int *dataBuffer;
{
    unsigned int i,temp,dByte;

    temp = dataBuffer[0];
    if(!temp) dByte = System_buffer[bytes_out_System_buffer++];
    else {
        dByte = dataBuffer[1];
        temp--;
        dataBuffer[0] = temp;
        for(i=1; i<=temp; i++)
            dataBuffer[i] = dataBuffer[i+1];
    }
    return(dByte);
}

```

```
void doSplicing(dataBuffer)
```

```

unsigned int *dataBuffer;
{
    unsigned int i,pastCount;

    pastCount = dataBuffer[0];
    dataBuffer[0] = pastCount + tByteCnt;
}

```

```

for(i=1; i<=tByteCnt; i++)
    dataBuffer[pastCount+i] = System_buffer[bytes_out_System_buffer++];  tByteCnt = 0;
}

```

```

unsigned int TS_program_association_section(trace)
unsigned int trace;
{
    unsigned int i,j,temp,not_ok,program_number,loop_number;
    int tcount,bCount,tspCount,presCount,error=1;

    programCount = 0;
    do {
        switch(PAT_State) {

            case 0:
                presCount = patData[0] + tByteCnt;
                if(presCount < 7) {
                    doSplicing(patData); /* do Splicing */
                    break;
                }
                /* start of the PAS */
                pas_data_ptr = input_data_ptr;

                /* Get the next byte */
                temp = getSplicedByte(patData);
                section_syntax_indicator = temp & 0x80;
                if(!section_syntax_indicator) {
                    printf("PAT: section_syntax_indicator should be set to 1..\n");
                    exit();
                }
                if(temp & 0x40) {
                    printf("PAT: it should be set to 0..\n");
                    exit();
                }
                pas_section_length = (temp & 0x0f) << 8;
                /* Get the next byte */
                pas_section_length |= getSplicedByte(patData);
                if(trace) printf("pas_section_length = %d\n",pas_section_length);
                /* Get the next 2 bytes */
                temp = 0;
                for(i=0; i<2; i++)
                    temp = (temp << 8) | getSplicedByte(patData);
                transport_stream_id = temp;
                if(trace) printf("transport_stream_id = %0x\n",transport_stream_id);

                /* Get the next byte */
                temp = getSplicedByte(patData);
                pas_version_number = (temp >> 1) & 0x1f;
                if(trace) printf("pas_version_number = %d\n",pas_version_number);
                current_next_indicator = temp & 0x1;
                if(trace && current_next_indicator)
                    printf("The TS_program_association_table is currently applicable \n");

                /* Get the next byte */
                pas_section_number = getSplicedByte(patData);
                if(trace) printf("pas_section_number = %d\n",pas_section_number);
                /* Get the next byte */
                pas_last_section_number = getSplicedByte(patData);

```

```

    if(trace)
        printf("pas_last_section_number = %d\n",pas_last_section_number);

    pas_section_length -= 5;    /* 5 bytes consumed so far */
    tByteCnt = presCount - 7;

    /* Total number of bytes for the prNO Loop for this Section */
    /* 4 bytes for CRC_32 */
    pas_section_loop_length = pas_section_length - 4;
    if(pas_section_loop_length)
        PAT_State = 1;
    else
        PAT_State = 2;
    break;

case 1:
    presCount = patData[0] + tByteCnt;
    if (presCount < pas_section_loop_length) {
        doSplicing(patData);    /* do Splicing */
        break;
    }
    loop_number = pas_section_loop_length >> 2;
    for(j=0; j<loop_number; j++) {
        /* Get the next 2 bytes */
        program_number = 0;
        for(i=0; i<2; i++)
            program_number = (program_number << 8) | getSplicedByte(patData);
        programMAP[programCount].programNumber = program_number;
        if(trace) printf("program_number = %d\n",program_number);

        /* Get the next 2 bytes */
        temp = 0;
        for(i=0; i<2; i++)
            temp = (temp << 8) | getSplicedByte(patData);
        temp &= 0x1fff;
        programMAP[programCount++].PID = temp;
        if(trace && (!program_number))
            printf("network_PID = %0x\n",temp);
        else if(trace)
            printf("program_map_PID[%d] = %0x\n",program_number,temp);
    }
    pas_section_loop_length -= 4;
    tByteCnt = presCount - (loop_number << 2);
    PAT_State = 2;
    break;

case 2:
    presCount = patData[0] + tByteCnt;
    if (presCount < 4) {
        doSplicing(patData);    /* do Splicing */
        break;
    }

    /* CRC_32: */
    CRC_32 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(patData);
        CRC_32 = (CRC_32 << 8) | temp;
    }

```

```

    if(trace) printf("PAT: CRC_32 = %0xh\n",CRC_32);

    /* address of the last bytes of PAS including CRC-32 */
    pas_crc_ptr = &System_buffer[bytes_out_System_buffer];

    /* Check CRC_32 of PAS */
    not_ok = Check_CRC(pas_data_ptr,pas_crc_ptr);
    if (not_ok)
        printf("PAS: CRC showed parity error !!\n");
    else
        printf("PAS: CRC checked OK\n");

    pas_section_loop_length = 0;
    tByteCnt = presCount - 4;
    PAT_State = 0;
    if(pas_section_number == pas_last_section_number)
        return(0); /* END OF PAT-TABLE */
    else
        return(2); /* Start next PAS - Section */
} /* end of switch */

} while (PAT_State && (tByteCnt > 0));
return(error);
}

unsigned int TS_CA_section(trace)
unsigned int trace;
{
    unsigned int i,j,n1,temp,not_ok,ePno=0,error=1;
    unsigned int presCount,descLength,descType;
    do {
        switch(CAT_State) {

        case 0:
            presCount = catData[0] + tByteCnt;
            if(presCount < 7) {
                doSplicing(catData); /* do Splicing */
                break;
            }

            cas_data_ptr = input_data_ptr;

            /* Get the next byte */
            temp = getSplicedByte(catData);
            section_syntax_indicator = temp & 0x80;
            if(!section_syntax_indicator) {
                printf("TS_CAS: section_syntax_indicator should be set to 1 ..\n");
                exit();
            }
            if(temp & 0x40) {
                printf("TS_CAS: it should be set to 0 ..\n");
                exit();
            }

            cas_section_length = (temp & 0x0f) << 8;
            /* Get the next byte */
            cas_section_length |= getSplicedByte(catData);
            if(trace) printf("CAS_section_length = %0x\n",cas_section_length);

```

```

/* Get the next 2 bytes and 2-bits as Reserved */
temp = 0;
for(i=0; i<2; i++)
    temp = (temp << 8) | getSplicedByte(catData);

/* Get the next byte */
temp = getSplicedByte(catData);
cas_version_number = (temp >> 1) & 0x1f;
if(trace) printf("cas_version_number = %d\n",cas_version_number);
cas_current_next_indicator = temp & 0x1;
if(trace && current_next_indicator)
    printf("The TS_CA_section is currently applicable \n");

/* Get the next byte */
cas_section_number = getSplicedByte(catData);

/* Get the next byte */
cas_last_section_number = getSplicedByte(catData);

/* number of bytes consumed so far */
cas_section_length -= 5; /* 5 bytes consumed so far */
tByteCnt = presCount - 7;

/* 4bytes for CRC_32 */
cas_section_loop_length = (cas_section_length) - 4;
if(cas_section_loop_length)
    CAT_State = 1;
else
    CAT_State = 2;
break;

case 1:
    presCount = catData[0] + tByteCnt;
    if (presCount < cas_section_loop_length) {
        doSplicing(catData); /* do Splicing */
        break;
    }

    if(cas_section_loop_length) {
        descLength = 0;
        do {
            descType = 257; /* descType = 257 for CAS */
            descriptor_ts(descType,catData);
            descLength += (2 + descriptor_length);
        } while (descLength < cas_section_loop_length);
    }

    /* Get the CAT_PID from the CA-Descriptor */
    if(CAT_PID)
        cas_PID = CAT_PID & 0x1fff;

    cas_section_length -= cas_section_loop_length;
    tByteCnt = presCount - cas_section_loop_length;
    CAT_State = 2;
    break;

case 2:
    presCount = catData[0] + tByteCnt;
    if (presCount < 4) {

```



```

        doSplicing(catData); /* do Splicing */
        break;
    }

    /* CRC_32: */
    CRC_32 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(catData);
        CRC_32 = (CRC_32 << 8) | temp;
    }
    if(trace) printf("CAT: CRC_32 = %0xh\n",CRC_32);

    /* address of the last bytes of PSM including CRC-32 */
    cas_crc_ptr = &System_buffer[bytes_out_System_buffer];

    /* Check CRC_32 of CAT */
    not_ok = Check_CRC(cas_data_ptr,cas_crc_ptr);
    if (not_ok)
        printf("CAS: CRC showed parity error !!\n");
    else
        printf("CAS: CRC checked OK\n");

    cas_section_loop_length = 0;
    tByteCnt = presCount - 4;
    CAT_State = 0;
    if(cas_section_number == cas_last_section_number)
        return(0); /* END OF CAT-TABLE */
    else
        return(2); /* Start next CAS - Section */
    } /* End of Switch */
    } while (CAT_State && (tByteCnt > 0));
    return(error);
}

unsigned int TS_program_map_section(trace)
unsigned int trace;
{
    unsigned int i,n1,n2,temp,descLength,not_ok,ES_PID,ePno=0;
    unsigned int program_number,p1_p2=0,error=1;
    unsigned int presCount,descType;

    do {
        switch(PMAP_State) {

        case 0:
            presCount = pmapData[0] + tByteCnt;
            if(presCount < 11) {
                doSplicing(pmapData); /* do Splicing */
                break;
            }

            pmap_data_ptr = input_data_ptr ;

            /* Get the next byte */
            temp = getSplicedByte(pmapData);
            section_syntax_indicator = temp & 0x80;
            if(!section_syntax_indicator) {
                printf("TS_PMAP: section_syntax_indicator should be set to 1..\n");

```

```

        exit();
    }
    if(temp & 0x40) {
        printf("TS_PMAP: it should be set to 0..\n");
        exit();
    }
    pmap_section_length = (temp & 0x0f) << 8;
    /* Get the next byte */
    pmap_section_length |= getSplicedByte(pmapData);
    if(trace)
        printf("pmap_section_length = %0x h\n",pmap_section_length);

    /* Get the next 2 bytes */
    program_number = 0;
    for(i=0; i<2; i++)
        program_number = (program_number << 8) | getSplicedByte(pmapData);
    if(trace) printf("program_number = %0x\n",program_number);

    /* Get the next byte */
    temp = getSplicedByte(pmapData);
    pmap_version_number = (temp >> 1) & 0x1f;
    if(trace) printf("pmap_version_number = %d\n",pmap_version_number);
    current_next_indicator = temp & 0x1;
    if(trace && current_next_indicator)
        printf("The TS_program_map_section is currently applicable \n");

    /* Get the next byte */
    pmap_section_number = getSplicedByte(pmapData);

    /* Get the next byte */
    pmap_last_section_number = getSplicedByte(pmapData);

    /* Get the next 2 bytes */
    temp = 0;
    for(i=0; i<2; i++)
        temp = (temp << 8) | getSplicedByte(pmapData);
    PCR_PID = temp & 0x1fff;
    if(trace)
        printf("PID of program that contains PCR info. = %0x\n",PCR_PID);

    /* program_info_length */
    temp = 0;
    for(i=0; i<2; i++)
        temp = (temp << 8) | getSplicedByte(pmapData);
    pmap_program_info_length = temp & 0xffff;
    if(trace)
        printf("program_info_length=%d\n",program_info_length);

    /* number of bytes consumed so far */
    pmap_section_length -= 9; /* 9 bytes consumed so far */
    tByteCnt = presCount - 11;
    if(pmap_program_info_length)
        PMAP_State = 1;
    else {
        /* 4bytes for CRC_32 */
        pmap_section_loop_length = (pmap_section_length) - 4;
        if(pmap_section_loop_length)
            PMAP_State = 2;
        else

```

```

        PMAP_State = 4;
    }
    break;

case 1:
    presCount = pmapData[0] + tByteCnt;
    if (presCount < (pmap_program_info_length) ) {
        doSplicing(pmapData); /* do Splicing */
        break;
    }
    if(pmap_program_info_length) {
        descLength = 0;
        do {
            descType = 258; /* descType = 258 for TS_PMAP_PInfo */
            descriptor_ts(descType,pmapData);
            descLength += (2 + descriptor_length);
        } while (descLength < pmap_program_info_length);
    }
    if(trace) printf("descLength (%d) <= program_info_length (%d)\n",descLength,program_info_length);

    pmap_section_length -= (pmap_program_info_length);
    tByteCnt = presCount - (pmap_program_info_length);

    /* 4bytes for CRC_32 */
    pmap_section_loop_length = (pmap_section_length) - 4;
    if(pmap_section_loop_length)
        PMAP_State = 2;
    else
        PMAP_State = 4;
    break;

case 2:
    presCount = pmapData[0] + tByteCnt;
    if (presCount < 5) {
        doSplicing(pmapData); /* do Splicing */
        break;
    }

    /* next byte */
    stream_type = getSplicedByte(pmapData);

    /* next 2 bytes */
    ES_PID = 0;
    for(i=0; i<2; i++)
        ES_PID = (ES_PID << 8) | getSplicedByte(pmapData);
    if(trace)
        printf("elementary_PID[%d] = %0x\n",ePno++,ES_PID);

    /* ES_info_length */
    temp = 0;
    for(i=0; i<2; i++)
        temp = (temp << 8) | getSplicedByte(pmapData);
    ES_info_length = temp & 0xff;
    if(trace) printf("ES_info_length=%d\n",ES_info_length);

    /* The following switch is used for keeping track of PCR_PID only */
    switch(stream_type) {

        case 1:

```

```

        video1_PID[(v1Cnt)++] = ES_PID;
        break;
    case 2:
        video2_PID[(v2Cnt)++] = ES_PID;
        break;

    case 3:
        audio1_PID[(a1Cnt)++] = ES_PID;
        break;
    case 4:
        audio2_PID[(a2Cnt)++] = ES_PID;
        break;

    case 6:
        if(!p1_p2) {
            pvt1_PID = ES_PID;
            p1_p2 = 1;
        }
        else
            pvt2_PID = ES_PID;
        break;

    case 8:
        dsm_CC_PID = ES_PID;
        break;

    default: /* Others are not used for now, for the PCR_PID case */
        break;
}

pmap_section_loop_length -= 5;
tByteCnt = presCount - 5;

if(!(pmap_section_loop_length))
    PMAP_State = 4;
else if(ES_info_length)
    PMAP_State = 3;
else
    PMAP_State = 2;
break;

case 3:
    presCount = pmapData[0] + tByteCnt;
    if (presCount < (ES_info_length) ) {
        doSplicing(pmapData); /* do Splicing */
        break;
    }

    descLength = 0;
    do {
        descType = stream_type; /* descType = stream_type */
        descriptor_ts(descType,pmapData);
        descLength += (2 + descriptor_length);
    } while (descLength < ES_info_length);

    pmap_section_loop_length -= (ES_info_length);
    tByteCnt = presCount - (ES_info_length);
    if(pmap_section_loop_length)
        PMAP_State = 2;

```

```

    else
        PMAP_State = 4;
    break;

case 4:
    presCount = pmapData[0] + tByteCnt;
    if (presCount < 4) {
        doSplicing(pmapData); /* do Splicing */
        break;
    }

    /* CRC_32: */
    CRC_32 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(pmapData);
        CRC_32 = (CRC_32 << 8) | temp;
    }
    if(trace) printf("PMAP: CRC_32 = %0xh\n",CRC_32);

    /* address of the last bytes of PSM including CRC-32 */
    pmap_crc_ptr = &System_buffer[bytes_out_System_buffer];

    /* Check CRC_32 of PMAP */
    not_ok = Check_CRC(pmap_data_ptr,pmap_crc_ptr);
    if (not_ok)
        printf("PMAP: CRC showed parity error !!\n");
    else
        printf("PMAP: CRC checked OK\n");

    pmap_section_loop_length = 0;
    tByteCnt = presCount - 4;
    PMAP_State = 0;
    if(pmap_section_number == pmap_last_section_number)
        return(0); /* END OF PMAP-TABLE */
    else
        return(2); /* Start next PMAP - Section */

} /* End of Switch */

} while (PMAP_State && (tByteCnt > 0));

return(error);
}

```

```

unsigned int TS_NIT_section(trace)
unsigned int trace;
{
    unsigned int presCount,descLength,descType;
    unsigned int i,j,temp,error=1;
    int n1,not_ok;

    do {
        switch(NIT_State) {

        case 0:
            /* start of the Private-section */
            presCount = nitData[0] + tByteCnt;
            if(presCount < 2) {

```

```

        doSplicing(nitData); /* do Splicing */
        break;
    }

    nit_data_ptr = input_data_ptr ;

    /* Get the next byte */
    temp = getSplicedByte(nitData);
    section_syntax_indicator = temp & 0x80;
    if(trace)
        printf("section_syntax_indicator = %0x h\n",section_syntax_indicator);
    nit_indicator = (temp & 0x40) >> 6;
    nit_section_length = (temp & 0x0f) << 8;
    /* Get the next byte */
    nit_section_length |= getSplicedByte(nitData);
    if(trace)
        printf("nit_section_length = %0x h\n",nit_section_length);
    tByteCnt = presCount - 2;

    /* 4-bytes are for CRC_32 */
    nit_section_loop_length = nit_section_length - 4;

    if(!section_syntax_indicator)
        NIT_State = 2;
    else
        NIT_State = 1;
    break;

case 1:
    presCount = nitData[0] + tByteCnt;
    if(presCount < 5) {
        doSplicing(nitData); /* do Splicing */
        break;
    }

    /* Get the next 2 bytes */
    temp = 0;
    for(i=0; i<2; i++)
        temp = (temp << 8) | getSplicedByte(nitData);
    table_id_extension = temp;

    /* Get the next byte: 1st 2 bits are reserved */
    temp = getSplicedByte(nitData);
    nit_version_number = (temp >> 1) & 0x1f;
    if(trace) printf("private_version_number = %d\n",nit_version_number);
    nit_current_next_indicator = temp & 0x1;
    if(trace && nit_current_next_indicator)
        printf("The TS_private_section is currently applicable \n");

    /* Get the next byte */
    nit_section_number = getSplicedByte(nitData);

    /* Get the next byte */
    nit_last_section_number = getSplicedByte(nitData);

    /* number of bytes consumed so far */
    nit_section_loop_length -= 5; /* 5 bytes consumed so far */
    tByteCnt = presCount - 5;

```

```

    if(nit_section_loop_length)
        NIT_State = 2;
    else
        NIT_State = 3;
    break;

case 2:
    presCount = nitData[0] + tByteCnt;
    if (presCount < nit_section_loop_length) {
        doSplicing(nitData); /* do Splicing */
        break;
    }

    /* Store the Private Data Bytes */
    temp = past_nit_size + nit_section_loop_length;
    if(temp > PVT_TBL_SIZE) exit(0);
    for(i=0; i<nit_section_loop_length; i++)
        nitsec_private_data_byte[past_nit_size++] = getSplicedByte(nitData);

    NIT_State = 3;
    break;

case 3:
    presCount = nitData[0] + tByteCnt;
    if (presCount < 4) {
        doSplicing(nitData); /* do Splicing */
        break;
    }

    /* CRC_32: */
    CRC_32 = 0;
    for(i=0; i<4; i++) {
        temp = getSplicedByte(nitData);
        CRC_32 = (CRC_32 << 8) | temp;
    }
    if(trace) printf("Private: CRC_32 = %0xh\n",CRC_32);

    /* address of the last bytes of PRIVATE Section including CRC-32 */
    nit_crc_ptr = &System_buffer[bytes_out_System_buffer];

    /* Check CRC_32 of Private Section */
    not_ok = Check_CRC(nit_data_ptr,nit_crc_ptr);
    if (not_ok)
        printf("Private: CRC showed parity error !!\n");
    else
        printf("Private: CRC checked OK\n");

    nit_section_loop_length = 0;
    tByteCnt = presCount - 4;
    NIT_State = 0;
    if(nit_section_number == nit_last_section_number)
        return(0); /* END OF Private-TABLE */
    else
        return(2); /* Start next Private - Section */

} /* End of Switch */

} while (NIT_State && (tByteCnt > 0));

```

```

    return(error);
}

unsigned int TS_private_section(trace)
unsigned int trace;
{
    unsigned int presCount,descLength,descType;
    unsigned int i,j,temp,error=1;
    int n1,not_ok;

    do {
        switch(PVT_State) {

        case 0:
            /* start of the Private-section */
            presCount = pvtData[0] + tByteCnt;
            if(presCount < 2) {
                doSplicing(pvtData); /* do Splicing */
                break;
            }

            pvt_data_ptr = input_data_ptr ;

            /* Get the next byte */
            temp = getSplicedByte(pvtData);
            section_syntax_indicator = temp & 0x80;
            if(trace)
                printf("section_syntax_indicator = %0x h\n",section_syntax_indicator);
            pvt_indicator = (temp & 0x40) >> 6;
            pvt_section_length = (temp & 0x0f) << 8;
            /* Get the next byte */
            pvt_section_length |= getSplicedByte(pvtData);
            if(trace)
                printf("pvt_section_length = %0x h\n",pvt_section_length);
            tByteCnt = presCount - 2;

            /* 4-bytes are for CRC_32 */
            pvt_section_loop_length = pvt_section_length - 4;

            if(!section_syntax_indicator)
                PVT_State = 2;
            else
                PVT_State = 1;
            break;

        case 1:
            presCount = pvtData[0] + tByteCnt;
            if(presCount < 5) {
                doSplicing(pvtData); /* do Splicing */
                break;
            }

            /* Get the next 2 bytes */
            temp = 0;
            for(i=0; i<2; i++)
                temp = (temp << 8) | getSplicedByte(pvtData);
            table_id_extension = temp;

```



```

/* Get the next byte: 1st 2 bits are reserved */
temp = getSplicedByte(pvtData);
pvt_version_number = (temp >> 1) & 0x1f;
if(trace) printf("private_version_number = %d\n",pvt_version_number);
pvt_current_next_indicator = temp & 0x1;
if(trace && pvt_current_next_indicator)
    printf("The TS_private_section is currently applicable \n");

/* Get the next byte */
pvt_section_number = getSplicedByte(pvtData);

/* Get the next byte */
pvt_last_section_number = getSplicedByte(pvtData);

/* number of bytes consumed so far */
pvt_section_loop_length -= 5; /* 5 bytes consumed so far */
tByteCnt = presCount - 5;

if(pvt_section_loop_length)
    PVT_State = 2;
else
    PVT_State = 3;
break;

case 2:
    presCount = pvtData[0] + tByteCnt;
    if (presCount < pvt_section_loop_length) {
        doSplicing(pvtData); /* do Splicing */
        break;
    }

/* Store the Private Data Bytes */
temp = past_pvtt_size + pvt_section_loop_length;
if(temp > PVT_TBL_SIZE) exit(0);
for(i=0; i<pvt_section_loop_length; i++)
    pvtsec_private_data_byte[past_pvtt_size++] = getSplicedByte(pvtData);

PVT_State = 3;
break;

case 3:
    presCount = pvtData[0] + tByteCnt;
    if (presCount < 4) {
        doSplicing(pvtData); /* do Splicing */
        break;
    }

/* CRC_32: */
CRC_32 = 0;
for(i=0; i<4; i++) {
    temp = getSplicedByte(pvtData);
    CRC_32 = (CRC_32 << 8) | temp;
}
if(trace) printf("Private: CRC_32 = %0xh\n",CRC_32);

/* address of the last bytes of PRIVATE Section including CRC-32 */
pvt_crc_ptr = &System_buffer[bytes_out_System_buffer];

/* Check CRC_32 of Private Section */

```

```

    not_ok = Check_CRC(pvt_data_ptr,pvt_crc_ptr);
    if (not_ok)
        printf("Private: CRC showed parity error !!\n");
    else
        printf("Private: CRC checked OK\n");

    pvt_section_loop_length = 0;
    tByteCnt = presCount - 4;
    PVT_State = 0;
    if(pvt_section_number == pvt_last_section_number)
        return(0); /* END OF Private-TABLE */
    else
        return(2); /* Start next Private - Section */

} /* End of Switch */

} while (tByteCnt > 0);

return(error);
}

unsigned int DSM_CC_Header(trace)
unsigned int trace;
{
    unsigned int temp,streamID;

    if((temp = System_buffer[bytes_out_System_buffer++]) == 0) { /*1 00 */
        if((temp = System_buffer[bytes_out_System_buffer++]) == 0) { /*2 00 00 */
            do {
                temp = System_buffer[bytes_out_System_buffer++];
            } while(temp == 0); /* again 00 */

            if(temp==1) { /*3 00 00 01 */
                streamID = System_buffer[bytes_out_System_buffer++];
                if(streamID != 0xf2) {
                    printf("WARNING: Wrong start_code (%x) in DSM_CC stream\n",streamID);
                    exit(1);
                }
            }
            if(trace)
                printf("Packet Start: DSM_CC Decoder Data: \n");
            /* dsmcc_command_id */
            dsmcc_command_id = System_buffer[bytes_out_System_buffer++];
            if(trace)
                printf("dsmcc_command_id=%d\n",dsmcc_command_id);
        } /* 3 */
    } /* 2 */
} /* 1 */

return(0);
}

unsigned int check_for_start_code(trace,strm_type)
unsigned int trace,strm_type;
{
    int i,j,scount,location,error=1;
    unsigned char temp;
    int bytes_out = bytes_out_Decoder_buffer;

    location = bytes_out;

```

```

do {
    scount = bytes_in_Decoder_buffer - bytes_out_Decoder_buffer;
    if (scount < 10) {
        bytes_out_Decoder_buffer = bytes_out;
        return(error); /* maximum size (!) = 10 */
    }
    temp = Decoder_buffer[bytes_out_Decoder_buffer++];
    if (!temp) { /*1 00 */
        temp = Decoder_buffer[bytes_out_Decoder_buffer++];
        if (!temp) { /*2 00 00 */

            do {
                temp = Decoder_buffer[bytes_out_Decoder_buffer++];
            } while(temp == 0); /* again 00 */

            if(temp == 1) { /* 00 00 01 */
                temp = Decoder_buffer[bytes_out_Decoder_buffer++];
                if( (strm_type == VIDEO) &&
                    (temp >= 0xe0) && (temp <= 0xef) ) error = 0;
                else if( (strm_type == AUDIO) &&
                    (temp >= 0xc0) && (temp <= 0xdf) ) error = 0;
                else if( (strm_type == PVT1) && (temp == 0xbd) )
                    error = 0;
                else if( (strm_type == PVT2) && (temp == 0xbf) )
                    error = 0;

                /* Found start-codes 00 00 01 */
                if(!error) {
                    printf("Start-codes are found for the video/audio/PVT streams(0x) ..\n",temp);
                    if(trace) printf("-----byteCtr=%d\n",byteCtr);
                    location = bytes_out_Decoder_buffer - location;
                    location -= 4;
                    if(debugS) printf("location=%d\n",location);
                }
            }
        }
    }
} while(error);

bytes_out_Decoder_buffer = bytes_out + location;

if(specialVideoCase && (strm_type == VIDEO)) {
    for(i=0; i<location; i++) {
        temp = Decoder_buffer[bytes_out + i];
        putc(temp,OutfileV);
    }
    fflush(OutfileV);
}

if(PacketSize) {
    if(strm_type == VIDEO) {
        for(i=0; i<location; i++) {
            temp = Decoder_buffer[bytes_out + i];
            putc(temp,OutfileV);
        }
        fflush(OutfileV);
    }
    else if(strm_type == AUDIO) {
        for(i=0; i<location; i++) {

```

```

        temp = Decoder_buffer[bytes_out + i];
        putc(temp, OutfileA);
    }
    fflush(OutfileA);
}
else if(strm_type == PVT1) {
    for(i=0; i<location; i++) {
        temp = Decoder_buffer[bytes_out + i];
        putc(temp, OutfileS1);
    }
    fflush(OutfileS1);
}
else if(strm_type == PVT2) {
    for(i=0; i<location; i++) {
        temp = Decoder_buffer[bytes_out + i];
        putc(temp, OutfileS2);
    }
    fflush(OutfileS2);
}
}
return(error);
}

/*****
*   TS_packet_Header - Table 2.3: Transport Header
*****/
unsigned int TS_packet_Header(trace)
unsigned int trace;
{
    unsigned int temp, PID;

    /* Get the first Sync_Byte: assumed byte-aligned */
    do {
        temp = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    } while (temp != SYNC_BYTE);

    if(trace) printf("1st Sync_Byte-location = %d\n",
        (bytes_out_DSM_buffer-1));
    if(trace) printf("byteCtr = %d\n", byteCtr);

    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    transport_error_indicator = (temp >> 7);
    if(transport_error_indicator) {
        printf("ERROR: discard the transport packet ..\n");
        /* error recovery is not in place */
        PID = 0x1fff;
        tByteCnt -= 2;
        return(PID);
    }
    payload_unit_start_indicator = (temp >> 6) & 0x1;
    transport_priority = (temp >> 5) & 0x1;

    /* Get the 2nd byte */
    PID = ((temp & 0x1f) << 8) | DSM_buffer[bytes_out_DSM_buffer++];
    if(trace) printf("New PID=%0x h\n", PID);
    byteCtr++;
    if(PID == 0x1fff) {

```

```

        if(trace) printf("Null Packet.....\n");
        tByteCnt -= 3;
        return(PID);
    }

    /* Get the 3rd byte */
    temp = DSM_buffer[bytes_out_DSM_buffer++];
    byteCtr++;
    transport_scrambling_control = (temp >> 6) & 0x3;
    adaptation_field_control = (temp >> 4) & 0x3;
    if(trace) printf("adaptation_field_control = %d\n",adaptation_field_control);
    continuity_counter = temp & 0xf;

    tByteCnt -= 4;

    /* Adaptation_field demuxing */
    if(adaptation_field_control < 1) {
        printf("adaptation_field_control is 0.\n");
        PID = 0x1fff;
        return(PID);
    }
    else if(adaptation_field_control > 1) {
        adaptation_field(trace);
        tByteCnt -= (adaptation_field_length + 1);
    }

    return(PID);
}

/*****
TS_Demultiplexor() block is clocked once for each byte that comes
from the DSM. The demultiplexor waits until there are enough bytes in the
DSM_buffer (a small buffer that stores data for the Demultiplexor)
for it to decode a syntax element, such as a start code or packet header.
When there are enough, it does the work to decode that element, then sets
the state of the Demultiplexor to look for the next expected syntactic
element.
*****/
void TS_Demultiplexor(trace)
unsigned int trace;
{
    unsigned int temp,PID,error=1,newTSpkt;
    unsigned int TSS_type,psiType,pidPsiType,pastState;
    int i,count,tspDataLeft;
    static int tpasDone=1,tcasDone=1,tpmapDone=1,tnisDone=1,tpvtDone=1;

    count = bytes_in_DSM_buffer - bytes_out_DSM_buffer;
    if (count < 188) return;

    /* Transport Layer Packet-Header */
    tByteCnt = 188;
    PID = TS_packet_Header(trace);
    if(trace) printf("PID = %0x and tsPacket_Payload_size = %d\n",PID,tByteCnt);

    TSS_type = detect_PID(trace,PID,&pidPsiType);
    if(pidPsiType == TRASHED_STREAM) {
        bytes_out_DSM_buffer += tByteCnt;
        byteCtr += tByteCnt;

```

```

    }
else if((adaptation_field_control==1) || (adaptation_field_control==3)) {
    /* Now start the Demuxing of PSI & PES packets */
    switch(TSS_type) {
    case PSI_DATA: /* PSI Data only */
        if(payload_unit_start_indicator) {
            /* Get the 1st byte: thrown out bytes-counter */
            pointer_field = DSM_buffer[bytes_out_DSM_buffer++];
            if(trace) printf("Pointer-field = %d\n",pointer_field);
            bytes_out_DSM_buffer += pointer_field;
            byteCtr += (pointer_field + 1);
            tByteCnt -= (pointer_field + 1);
        }

        newTSpkt = 1;
        /* Store into a temporary System_buffer */
        for(i=0; i<tByteCnt; i++) {
            System_buffer[bytes_in_System_buffer++]
                = DSM_buffer[bytes_out_DSM_buffer++];
            byteCtr++;
        }

        while (tByteCnt > 0) {
            if(
                (!newTSpkt) ||
                (tpasDone && (pidPsiType == PAS)) ||
                (tcasDone && (pidPsiType == CAS)) ||
                (tpmapDone && (pidPsiType == PMAP)) ||
                (tnisDone && (pidPsiType == NIS)) ||
                (tpvtDone && (pidPsiType == PRIVATE)) ) {
                input_data_ptr = &System_buffer[bytes_out_System_buffer];
                /* Get the 2nd byte */
                table_id = System_buffer[bytes_out_System_buffer++];
                if(trace) printf("table_id = %d\n",table_id);
                tByteCnt -= 1;

                psiType = detect_TableID(table_id);
                newTSpkt = 0;
                if(psiType == PAS) {
                    PAT_State = 0;
                    patData[0] = 0;
                }
                else if(psiType == CAS) {
                    CAT_State = 0;
                    catData[0] = 0;
                }
                else if(psiType == PMAP) {
                    PMAP_State = 0;
                    pmapData[0] = 0;
                }
                else if(psiType == PRIVATE) {
                    if(pidPsiType == NIS) {
                        NIT_State = 0;
                        nitData[0] = 0;
                        psiType = NIS;
                        past_nit_size = 0;
                    }
                    else {
                        PVT_State = 0;
                        pvtData[0] = 0;
                    }
                }
            }
        }
    }
}

```

```

        past_pvtt_size = 0;
    }
}
else
    psiType = pidPsiType;

switch(psiType) {
case PAS: /* PID=0 & table_id=0 for PAS */
    if(trace) printf("PAS_Start:\n");

    error = TS_program_association_section(trace);
    tpasDone = !(error & 0x1);
#ifdef CHANNEL_SWITCH
    /* program Maps */
    for(i=0; i<programCount; i++) {
        programNO = programMAP[i].programNumber;
        if(!programNO)
            nitPIDselect = programMAP[i].PID;
        else if(programNO == program_select) {
            programPIDselect = programMAP[i].PID;
            break;
        }
    }
    if(trace) printf("selected programPID=%0x\n",programPIDselect);
#endif
    if(!tByteCnt) adjust_System_buffer();
    break;

case CAS: /* PID=1 & table_id=1 for CAS */
    if(trace) printf("CAS_Start:\n");

    error = TS_CA_section(trace);
    tcasDone = !(error & 0x1);

    if(!tByteCnt) adjust_System_buffer();
    break;

case PMAP: /* table_id=2 for Program_Maps */
    if(trace) printf("PMAP_Start:\n");
    error = TS_program_map_section(trace);
    tpmappedDone = !(error & 0x1);

#ifdef CHANNEL_SWITCH
    /* Select the video/audio channel for decoding */
    if(!initSystem) {
        user_interface_for_service(trace);
        if((videoPIDselect == 0x1fff)
            && (audioPIDselect == 0x1fff)) {
            printf("The program_map is not available yet - try later...\n");
            initSystem = 0;
        }
        else
            initSystem = 1;
    }
#endif
    if(!tByteCnt) adjust_System_buffer();
    break;

```

```

case NIS: /* table_id=0x40 to 0xfe for NIT: Private Table */
    if(trace) printf("NIT_Start: byteCtr=%d\n",byteCtr);

    error = TS_NIT_section(trace);
    tnisDone = !(error & 0x1);

    if(!tByteCnt) adjust_System_buffer();
    break;

case PRIVATE: /* table_id=0x40 to 0xfe for Private Table */
    if(trace) printf("Private_PSI_Start:\n");

    error = TS_private_section(trace);
    tpvtDone = !(error & 0x1);

    if(!tByteCnt) adjust_System_buffer();
    break;

case PSI_TS_PKT_DONE: /* No more PSI data in TS_packet */
    if(trace) printf("PSI_TS_Packet_Done:\n");
    tByteCnt = 0;
    adjust_System_buffer();
    break;
} /* end of PSI switch */
} /* end of while loop */
break; /* end of PSI data */

case DSM_CC_TS: /* DSM CC Decoder */
    if(trace) printf("DSM_CC_Decoder: byteCtr=%d\n",byteCtr);
    for(i=0; i<tByteCnt; i++) {
        System_buffer[bytes_in_System_buffer++]
            = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    }

    error = DSM_CC_Header(trace);

    /* Decode the DSM_CC Payload */
    DSM_CC_Decoder(trace);
    break;

case VIDEO_STREAM: /* Selected Video Program_Stream */
    if(trace) printf("VideoStream_Start: tByteCnt=%d\n",tByteCnt);
    if(trace) printf("byteCtr=%d\n",byteCtr);

    if(debugS) printData(DSM_buffer,bytes_out_DSM_buffer,100);

    for(i=0; i<tByteCnt; i++) {
        Video_buffer[bytes_in_Video_buffer++]
            = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    }

    bytes_in_Decoder_buffer = bytes_in_Video_buffer;
    bytes_out_Decoder_buffer = bytes_out_Video_buffer;
    PacketSize = videoPacketSize;
    P_State = V_State;
    Decoder_buffer = videoBuffer;
    PES_stream_type = VIDEO;

```



```

PES_packet_length = video_PES_packet_length;

    if(payload_unit_start_indicator)
        error = check_for_start_code(trace,VIDEO);
    if(!error) {
        specialVideoCase = FALSE;
        P_State = IDENTIFY_START_CODE;
    }
    if((!videoStart) && (!specialVideoCase)) {
        if(!error) videoStart = 1;
        else videoStart = 0;
    }

    if(trace) {
        printf("Video:Before PrDemux: bytes_in_Decoder_buffer = %d\n",bytes_in_Decoder_buffer);
        printf(" bytes_out_Decoder_buffer = %d\n",bytes_out_Decoder_buffer);
    }

    if(debugS) printData(Decoder_buffer,bytes_out_Decoder_buffer,100);

    if(videoStart) {
        do {
            pastState = P_State;
            tspDataLeft = tByteCnt;
            PS_Demultiplexor(trace,&tspDataLeft);
            tByteCnt = tspDataLeft;
        } while(P_State != pastState);
    }

    V_State = P_State;
    video_PES_packet_length = PES_packet_length;
    videoPacketSize = PacketSize;
    bytes_in_Video_buffer = bytes_in_Decoder_buffer;
    bytes_out_Video_buffer = bytes_out_Decoder_buffer;

    adjust_Video_buffer();
    break;

case AUDIO_STREAM: /* Selected Audio Program_Stream */
    if(trace) printf("AudioStream_Start: tByteCnt=%d\n",tByteCnt);
    if(trace) printf(" byteCtr=%d\n",byteCtr);

    if(debugS) printData(DSM_buffer,bytes_out_DSM_buffer,100);

    for(i=0; i<tByteCnt; i++) {
        Audio_buffer[bytes_in_Audio_buffer++]
            = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    }

    bytes_in_Decoder_buffer = bytes_in_Audio_buffer;
    bytes_out_Decoder_buffer = bytes_out_Audio_buffer;
    P_State = A_State;
    PacketSize = audioPacketSize;
    Decoder_buffer = audioBuffer;
    PES_stream_type = AUDIO;
    PES_packet_length = audio_PES_packet_length;

    if(!audioStart) {

```

```

    if(payload_unit_start_indicator)
        error = check_for_start_code(trace,AUDIO);
    if(!error) {
        P_State = IDENTIFY_START_CODE;
        audioStart = 1;
    }
    else audioStart = 0;
}

if(trace) {
    printf("Audio:Before PrDemux: bytes_in_Decoder_buffer = %d\n",bytes_in_Decoder_buffer);
    printf(" bytes_out_Decoder_buffer = %d\n",bytes_out_Decoder_buffer);
}

if(debugS) printData(Decoder_buffer,bytes_out_Decoder_buffer,100);

if(audioStart) {
    do {
        pastState = P_State;
        tspDataLeft = tByteCnt;
        PS_Demultiplexor(trace,&tspDataLeft);
        tByteCnt = tspDataLeft;
    } while(P_State != pastState);
}

A_State = P_State;
audio_PES_packet_length = PES_packet_length;
audioPacketSize = PacketSize;
bytes_in_Audio_buffer = bytes_in_Decoder_buffer;
bytes_out_Audio_buffer = bytes_out_Decoder_buffer;

adjust_Audio_buffer();
break;

case PRIVATE_STREAM_1: /* Selected Private1 Program_Stream */
    if(trace) printf("Pvt1Stream_Start: byteCtr=%d\n",byteCtr);

    if(debugS) printData(DSM_buffer,bytes_out_DSM_buffer,100);

    for(i=0; i<tByteCnt; i++) {
        Private1_buffer[bytes_in_Private1_buffer++]
            = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    }

    bytes_in_Decoder_buffer = bytes_in_Private1_buffer;
    bytes_out_Decoder_buffer = bytes_out_Private1_buffer;
    P_State = PVT1_State;
    PacketSize = pvt1PacketSize;
    Decoder_buffer = pvt1Buffer;
    PES_stream_type = PVT1;
    PES_packet_length = pvt1_PES_packet_length;

    if(!pvt1Start) {
        if(payload_unit_start_indicator)
            error = check_for_start_code(trace,PVT1);
        if(!error) {
            P_State = IDENTIFY_START_CODE;
            pvt1Start = 1;

```

```

    }
    else pvt1Start = 0;
    }

    if(trace) {
        printf("PVT1:Before PrDemux: bytes_in_Decoder_buffer = %d\n",bytes_in_Decoder_buffer);
        printf(" bytes_out_Decoder_buffer = %d\n",bytes_out_Decoder_buffer);
    }

    if(debugS) printData(Decoder_buffer,bytes_out_Decoder_buffer,100);

    if(pvt1Start) {
        do {
            pastState = P_State;
            tspDataLeft = tByteCnt;
            PS_Demultiplexor(trace,&tspDataLeft);
            tByteCnt = tspDataLeft;
        } while(P_State != pastState);
    }

    PVT1_State = P_State;
    pvt1_PES_packet_length = PES_packet_length;
    pvt1PacketSize = PacketSize;
    bytes_in_Private1_buffer = bytes_in_Decoder_buffer;
    bytes_out_Private1_buffer = bytes_out_Decoder_buffer;

    adjust_Private1_buffer();
    break;

case PRIVATE_STREAM_2: /* Selected Private2 Program_Stream */
    if(trace) printf("Pvt2Stream_Start: byteCtr=%d\n",byteCtr);

    if(debugS) printData(DSM_buffer,bytes_out_DSM_buffer,100);

    for(i=0; i<tByteCnt; i++) {
        Private2_buffer[bytes_in_Private2_buffer++]
            = DSM_buffer[bytes_out_DSM_buffer++];
        byteCtr++;
    }

    bytes_in_Decoder_buffer = bytes_in_Private2_buffer;
    bytes_out_Decoder_buffer = bytes_out_Private2_buffer;
    P_State = PVT2_State;
    Decoder_buffer = pvt2Buffer;

    if(!pvt2Start) {
        if(payload_unit_start_indicator)
            error = check_for_start_code(trace,PVT2);
        if(!error) {
            P_State = IDENTIFY_START_CODE;
            pvt2Start = 1;
        }
        else pvt2Start = 0;
    }

    if(trace) {
        printf("PVT2:Before PrDemux: bytes_in_Decoder_buffer = %d\n",bytes_in_Decoder_buffer);
        printf(" bytes_out_Decoder_buffer = %d\n",bytes_out_Decoder_buffer);
    }

```

```

if(debugS) printData(Decoder_buffer,bytes_out_Decoder_buffer,100);

if(pvt2Start) {
    do {
        pastState = P_State;
        tspDataLeft = tByteCnt;
        PS_Demultiplexor(trace,&tspDataLeft);
        tByteCnt = tspDataLeft;
    } while(P_State != pastState);
}

PVT2_State = P_State;
bytes_in_Private2_buffer = bytes_in_Decoder_buffer;
bytes_out_Private2_buffer = bytes_out_Decoder_buffer;

adjust_Private2_buffer();
break;

case TRASHED_STREAM:
    bytes_out_DSM_buffer += tByteCnt;
    byteCtr += tByteCnt;
    break;

} /* Big Switch for PSI & Others */

} /* Adaptation_field = 1 or 3 */
else {
    bytes_out_DSM_buffer += tByteCnt;
    byteCtr += tByteCnt;
}

adjust_DSM_buffer();
}

#endif

```

## Annex B

Video - code listings

### B.1 Introduction

This Annex contains the C source code listings for the CD 13818-5 Video codec.

**Table B.1 List of video encoder files**

filename	description
config.h	global configuration file
conform.c	compliance checks of user-specified parameters
fdctref.c	floating point reference DCT
global.h	example makefile for GNU gcc Unix and MS-DOS compilers
idct.c	Fast IDCT routine
makefile	example makefile (GNU gcc, Sun cc, et al)
motion.c	block matching routines
mpeg2enc.c	initialization and argument parsing
mpeg2enc.h	global constants and structures
predict.c	motion compensated prediction stages
putbits.c	low-level I/O
puthdr.c	header generation routines
putmpg.c	syntax element generation

putpic.c	picture generation routine
putseq.c	sequence generation routine
putvlc.c	encode variable length codes
quantize.c	scalar quantization of transform coefficients
ratectl.c	Test Model 5 rate control operations
readpic.c	read picture from file
stats.c	print bitstream statistics
transfrm.c	call DCT, zig-zag scanning, and quantization
vlc.h	variable length tables
writpic.c	write reconstructed picture to file

**Table B.2 List of video decoder files**

filename	description
config.h	global configuration file
display.c	display routines for X-windows
getbits.c	low level I/O operations for parsing bitstream
getblk.c	parse and decode 8x8 block of transform coefficients
gethdr.c	parse header routines
getpic.c	decode picture
getvlc.c	parse variable length codes
getvlc.h	variable length code tables
global.h	global constants
idct.c	IDCT algorithm
idctf.c	fast IDCT algorithm
idctref.c	floating point reference IDCT
makefile	example makefile (GNU gcc, Sun cc, et al)
motion.c	motion compensated prediction routines
mpeg2dec.c	initialization and argument parsing
mpeg2dec.h	global constants, structures, and configuration flags
recon.c	low-level block reconstruction routines
spatscal.c	spatial scalability
store.c	write reconstructed picture to file

## Decoder

config.h

/\* config.h, configuration defines

\*/

/\* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims  
 \* any and all warranties, whether express, implied, or statutory, 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.

\*

\* The MPEG Software Simulation Group 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.

\*

\*/

/\* define NON\_ANSI\_COMPILER for compilers without function prototyping \*/

/\* #define NON\_ANSI\_COMPILER \*/

#ifdef NON\_ANSI\_COMPILER

#define \_ANSI\_ARGS\_(x) ()

#else

#define \_ANSI\_ARGS\_(x) x

#endif

conform.c

/\* conform.c, conformance checks

\*/

/\* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims  
 \* any and all warranties, whether express, implied, or statutory, 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.
*
* The MPEG Software Simulation Group 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.
*
*/

```

```

#include <stdio.h>
#include <stdlib.h>

```

```

#include "config.h"
#include "global.h"

```

```

/* check for (level independent) parameter limits */
void range_checks()
{
    int i;

    /* range and value checks */

    if (horizontal_size<1 || horizontal_size>16383)
        error("horizontal_size must be between 1 and 16383");
    if (mpeg1 && horizontal_size>4095)
        error("horizontal_size must be less than 4096 (MPEG-1)");
    if ((horizontal_size&4095)==0)
        error("horizontal_size must not be a multiple of 4096");
    if (chroma_format!=CHROMA444 && horizontal_size%2 != 0)
        error("horizontal_size must be a even (4:2:0 / 4:2:2)");

    if (vertical_size<1 || vertical_size>16383)
        error("vertical_size must be between 1 and 16383");
    if (mpeg1 && vertical_size>4095)
        error("vertical size must be less than 4096 (MPEG-1)");
    if ((vertical_size&4095)==0)
        error("vertical_size must not be a multiple of 4096");
    if (chroma_format==CHROMA420 && vertical_size%2 != 0)
        error("vertical_size must be a even (4:2:0)");
    if(fieldpic)
    {
        if (vertical_size%2 != 0)
            error("vertical_size must be a even (field pictures)");
        if (chroma_format==CHROMA420 && vertical_size%4 != 0)
            error("vertical_size must be a multiple of 4 (4:2:0 field pictures)");
    }

    if (mpeg1)
    {
        if (aspectratio<1 || aspectratio>14)
            error("pel_aspect_ratio must be between 1 and 14 (MPEG-1)");
    }
    else

```

```

{
    if (aspectratio<1 || aspectratio>4)
        error("aspect_ratio_information must be 1, 2, 3 or 4");
}

if (frame_rate_code<1 || frame_rate_code>8)
    error("frame_rate code must be between 1 and 8");

if (bit_rate<=0.0)
    error("bit_rate must be positive");
if (bit_rate > ((1<<30)-1)*400.0)
    error("bit_rate must be less than 429 Gbit/s");
if (mpeg1 && bit_rate > ((1<<18)-1)*400.0)
    error("bit_rate must be less than 104 Mbit/s (MPEG-1)");

if (vbv_buffer_size<1 || vbv_buffer_size>0x3ffff)
    error("vbv_buffer_size must be in range 1..(2^18-1)");
if (mpeg1 && vbv_buffer_size>=1024)
    error("vbv_buffer_size must be less than 1024 (MPEG-1)");

if (chroma_format<CHROMA420 || chroma_format>CHROMA444)
    error("chroma_format must be in range 1...3");

if (video_format<0 || video_format>4)
    error("video_format must be in range 0...4");

if (color_primaries<1 || color_primaries>7 || color_primaries==3)
    error("color_primaries must be in range 1...2 or 4...7");

if (transfer_characteristics<1 || transfer_characteristics>7
    || transfer_characteristics==3)
    error("transfer_characteristics must be in range 1...2 or 4...7");

if (matrix_coefficients<1 || matrix_coefficients>7 || matrix_coefficients==3)
    error("matrix_coefficients must be in range 1...2 or 4...7");

if (display_horizontal_size<0 || display_horizontal_size>16383)
    error("display_horizontal_size must be in range 0...16383");
if (display_vertical_size<0 || display_vertical_size>16383)
    error("display_vertical_size must be in range 0...16383");

if (dc_prec<0 || dc_prec>3)
    error("intra_dc_precision must be in range 0...3");

for (i=0; i<M; i++)
{
    if (motion_data[i].forw_hor_f_code<1 || motion_data[i].forw_hor_f_code>9)
        error("f_code must be between 1 and 9");
    if (motion_data[i].forw_vert_f_code<1 || motion_data[i].forw_vert_f_code>9)
        error("f_code must be between 1 and 9");
    if (mpeg1 && motion_data[i].forw_hor_f_code>7)
        error("f_code must be le less than 8");
    if (mpeg1 && motion_data[i].forw_vert_f_code>7)
        error("f_code must be le less than 8");
    if (motion_data[i].sxf<=0)
        error("search window must be positive"); /* doesn't belong here */
    if (motion_data[i].syf<=0)
        error("search window must be positive");
    if (i!=0)

```



```

    {
        if (motion_data[i].back_hor_f_code<1 || motion_data[i].back_hor_f_code>9)
            error("f_code must be between 1 and 9");
        if (motion_data[i].back_vert_f_code<1 || motion_data[i].back_vert_f_code>9)
            error("f_code must be between 1 and 9");
        if (mpeg1 && motion_data[i].back_hor_f_code>7)
            error("f_code must be le less than 8");
        if (mpeg1 && motion_data[i].back_vert_f_code>7)
            error("f_code must be le less than 8");
        if (motion_data[i].sxb<=0)
            error("search window must be positive");
        if (motion_data[i].syb<=0)
            error("search window must be positive");
    }
}
}

```

/\* identifies valid profile / level combinations \*/

static char profile\_level\_defined[5][4] =

```

{
    /* HL H-14 ML LL */
    {1, 1, 1, 0}, /* HP */
    {0, 1, 0, 0}, /* Spat */
    {0, 0, 1, 1}, /* SNR */
    {1, 1, 1, 1}, /* MP */
    {0, 0, 1, 0} /* SP */
};

```

static struct level\_limits {

```

    int hor_f_code;
    int vert_f_code;
    int hor_size;
    int vert_size;
    int sample_rate;
    int bit_rate; /* Mbit/s */
    int vbv_buffer_size; /* 16384 bit steps */
} maxval_tab[4] =
{
    {9, 5, 1920, 1152, 62668800, 80, 597}, /* HL */
    {9, 5, 1440, 1152, 47001600, 60, 448}, /* H-14 */
    {8, 5, 720, 576, 10368000, 15, 112}, /* ML */
    {7, 4, 352, 288, 3041280, 4, 29} /* LL */
};

```

```

#define SP 5
#define MP 4
#define SNR 3
#define SPAT 2
#define HP 1

```

```

#define LL 10
#define ML 8
#define H14 6
#define HL 4

```

void profile\_and\_level\_checks()

```

{
    int i;
    struct level_limits *maxval;

```

```

if (profile<0 || profile>15)
    error("profile must be between 0 and 15");

if (level<0 || level>15)
    error("level must be between 0 and 15");

if (profile>=8)
{
    if (!quiet)
        fprintf(stderr,"Warning: profile uses a reserved value, conformance checks skipped\n");
    return;
}

if (profile<HP || profile>SP)
    error("undefined Profile");

if (profile==SNR || profile==SPAT)
    error("This encoder currently generates no scalable bitstreams");

if (level<HL || level>LL || level&1)
    error("undefined Level");

maxval = &maxval_tab[(level-4) >> 1];

/* check profile@level combination */
if(!profile_level_defined[profile-1][(level-4) >> 1])
    error("undefined profile@level combination");

/* profile (syntax) constraints */

if (profile==SP && M!=1)
    error("Simple Profile does not allow B pictures");

if (profile!=HP && chroma_format!=CHROMA420)
    error("chroma format must be 4:2:0 in specified Profile");

if (profile==HP && chroma_format==CHROMA444)
    error("chroma format must be 4:2:0 or 4:2:2 in High Profile");

if (profile>=MP) /* SP, MP: constrained repeat_first_field */
{
    if (frame_rate_code<=2 && repeatfirst)
        error("repeat_first_first must be zero");
    if (frame_rate_code<=6 && prog_seq && repeatfirst)
        error("repeat_first_first must be zero");
}

if (profile!=HP && dc_prec==3)
    error("11 bit DC precision only allowed in High Profile");

/* level (parameter value) constraints */

/* Table 8-8 */
if (frame_rate_code>5 && level>=ML)
    error("Picture rate greater than permitted in specified Level");

```

```

for (i=0; i<M; i++)
{
    if (motion_data[i].forw_hor_f_code > maxval->hor_f_code)
        error("forward horizontal f_code greater than permitted in specified Level");

    if (motion_data[i].forw_vert_f_code > maxval->vert_f_code)
        error("forward vertical f_code greater than permitted in specified Level");

    if (i!=0)
    {
        if (motion_data[i].back_hor_f_code > maxval->hor_f_code)
            error("backward horizontal f_code greater than permitted in specified Level");

        if (motion_data[i].back_vert_f_code > maxval->vert_f_code)
            error("backward vertical f_code greater than permitted in specified Level");
    }
}

/* Table 8-10 */
if (horizontal_size > maxval->hor_size)
    error("Horizontal size is greater than permitted in specified Level");

if (vertical_size > maxval->vert_size)
    error("Horizontal size is greater than permitted in specified Level");

/* Table 8-11 */
if (horizontal_size*vertical_size*frame_rate > maxval->sample_rate)
    error("Sample rate is greater than permitted in specified Level");

/* Table 8-12 */
if (bit_rate > 1.0e6 * maxval->bit_rate)
    error("Bit rate is greater than permitted in specified Level");

/* Table 8-13 */
if (vbv_buffer_size > maxval->vbv_buffer_size)
    error("vbv_buffer_size exceeds High Level limit");
}

```

fdctref.c

```

/* fdctref.c, forward discrete cosine transform, double precision */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.

```

```

*
* The MPEG Software Simulation Group 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.
*
*/

```

```
#include <math.h>
```

```
#include "config.h"
```

```

#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif

```

```

/* global declarations */
void init_fdct _ANSI_ARGS_((void));
void fdct _ANSI_ARGS_((short *block));

```

```

/* private data */
static double c[8][8]; /* transform coefficients */

```

```

void init_fdct()
{
    int i, j;
    double s;

    for (i=0; i<8; i++)
    {
        s = (i==0) ? sqrt(0.125) : 0.5;

        for (j=0; j<8; j++)
            c[i][j] = s * cos((PI/8.0)*i*(j+0.5));
    }
}

```

```

void fdct(block)
short *block;
{
    int i, j, k;
    double s;
    double tmp[64];

    for (i=0; i<8; i++)
        for (j=0; j<8; j++)
        {
            s = 0.0;

            for (k=0; k<8; k++)
                s += c[j][k] * block[8*i+k];

```

```

    tmp[8*i+j] = s;
}

for (j=0; j<8; j++)
    for (i=0; i<8; i++)
    {
        s = 0.0;

        for (k=0; k<8; k++)
            s += c[i][k] * tmp[8*k+j];

        block[8*i+j] = (int)floor(s+0.499999);
        /*
         * reason for adding 0.499999 instead of 0.5:
         * s is quite often x.5 (at least for i and/or j = 0 or 4)
         * and setting the rounding threshold exactly to 0.5 leads to an
         * extremely high arithmetic implementation dependency of the result;
         * s being between x.5 and x.500001 (which is now incorrectly rounded
         * downwards instead of upwards) is assumed to occur less often
         * (if at all)
         */
    }
}

global.h

/* global.h, global variables, function prototypes */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include "mpeg2enc.h"

```

```

/* choose between declaration (GLOBAL undefined)
 * and definition (GLOBAL defined)
 * GLOBAL is defined in exactly one file (mpeg2enc.c)
 */

#ifndef GLOBAL
#define EXTERN extern
#else
#define EXTERN
#endif

/* prototypes of global functions */

/* conform.c */
void range_checks _ANSI_ARGS_((void));
void profile_and_level_checks _ANSI_ARGS_(());

/* fdctref.c */
void init_fdct _ANSI_ARGS_((void));
void fdct _ANSI_ARGS_((short *block));

/* idct.c */
void idct _ANSI_ARGS_((short *block));
void init_idct _ANSI_ARGS_((void));

/* motion.c */
void motion_estimation _ANSI_ARGS_((unsigned char *oldorg, unsigned char *neworg,
unsigned char *oldref, unsigned char *newref, unsigned char *cur,
unsigned char *curref, int sx, int sy, int sxb, int syb,
struct mbinf *mbi, int secondfield, int ipflag));

/* mpeg2enc.c */
void error _ANSI_ARGS_((char *text));

/* predict.c */
void predict _ANSI_ARGS_((unsigned char *reff[], unsigned char *refb[],
unsigned char *cur[3], int secondfield, struct mbinf *mbi));

/* putbits.c */
void initbits _ANSI_ARGS_((void));
void putbits _ANSI_ARGS_((int val, int n));
void alignbits _ANSI_ARGS_((void));
int bitcount _ANSI_ARGS_((void));

/* puthdr.c */
void putseqhdr _ANSI_ARGS_((void));
void putseqext _ANSI_ARGS_((void));
void putseqdispext _ANSI_ARGS_((void));
void putuserdata _ANSI_ARGS_((char *userdata));
void putgophdr _ANSI_ARGS_((int frame, int closed_gop));
void putpichdr _ANSI_ARGS_((void));
void putpictcodext _ANSI_ARGS_((void));
void putseqend _ANSI_ARGS_((void));

/* putmpg.c */
void putintrablk _ANSI_ARGS_((short *blk, int cc));
void putnonintrablk _ANSI_ARGS_((short *blk));
void putmv _ANSI_ARGS_((int dm, int f_code));

```

```

/* putpic.c */
void putpict _ANSI_ARGS_((unsigned char *frame));

/* putseq.c */
void putseq _ANSI_ARGS_((void));

/* putvlc.c */
void putDClum _ANSI_ARGS_((int val));
void putDCchrom _ANSI_ARGS_((int val));
void putACfirst _ANSI_ARGS_((int run, int val));
void putAC _ANSI_ARGS_((int run, int signed_level, int vlcformat));
void putaddrinc _ANSI_ARGS_((int addrinc));
void putmbtype _ANSI_ARGS_((int pict_type, int mb_type));
void putmotioncode _ANSI_ARGS_((int motion_code));
void putdmv _ANSI_ARGS_((int dmv));
void putcbp _ANSI_ARGS_((int cbp));

/* quantize.c */
int quant_intra _ANSI_ARGS_((short *src, short *dst, int dc_prec,
    unsigned char *quant_mat, int mquant));
int quant_non_intra _ANSI_ARGS_((short *src, short *dst,
    unsigned char *quant_mat, int mquant));
void iquant_intra _ANSI_ARGS_((short *src, short *dst, int dc_prec,
    unsigned char *quant_mat, int mquant));
void iquant_non_intra _ANSI_ARGS_((short *src, short *dst,
    unsigned char *quant_mat, int mquant));

/* ratectl.c */
void rc_init_seq _ANSI_ARGS_((void));
void rc_init_GOP _ANSI_ARGS_((int np, int nb));
void rc_init_pict _ANSI_ARGS_((unsigned char *frame));
void rc_update_pict _ANSI_ARGS_((void));
int rc_start_mb _ANSI_ARGS_((void));
int rc_calc_mquant _ANSI_ARGS_((int j));
void vbv_end_of_picture _ANSI_ARGS_((void));
void calc_vbv_delay _ANSI_ARGS_((void));

/* readpic.c */
void readframe _ANSI_ARGS_((char *fname, unsigned char *frame[]));

/* stats.c */
void calcSNR _ANSI_ARGS_((unsigned char *org[3], unsigned char *rec[3]));
void stats _ANSI_ARGS_((void));

/* transfrm.c */
void transform _ANSI_ARGS_((unsigned char *pred[], unsigned char *cur[],
    struct mbinf *mbi, short blocks[][64]));
void itransform _ANSI_ARGS_((unsigned char *pred[], unsigned char *cur[],
    struct mbinf *mbi, short blocks[][64]));
void dct_type_estimation _ANSI_ARGS_((unsigned char *pred, unsigned char *cur,
    struct mbinf *mbi));

/* writepic.c */
void writeframe _ANSI_ARGS_((char *fname, unsigned char *frame[]));

/* global variables */

```

```

EXTERN char version[]
#ifdef GLOBAL
    ="mpeg2encode V1.1a, 94/07/04"
#endif
;

EXTERN char author[]
#ifdef GLOBAL
    ="(C) 1994, MPEG Software Simulation Group"
#endif
;

/* zig-zag scan */
EXTERN unsigned char zig_zag_scan[64]
#ifdef GLOBAL
    =
    {
        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
    }
#endif
;

/* alternate scan */
EXTERN unsigned char alternate_scan[64]
#ifdef GLOBAL
    =
    {
        0,8,16,24,1,9,2,10,17,25,32,40,48,56,57,49,
        41,33,26,18,3,11,4,12,19,27,34,42,50,58,35,43,
        51,59,20,28,5,13,6,14,21,29,36,44,52,60,37,45,
        53,61,22,30,7,15,23,31,38,46,54,62,39,47,55,63
    }
#endif
;

/* default intra quantization matrix */
EXTERN unsigned char default_intra_quantizer_matrix[64]
#ifdef GLOBAL
    =
    {
        8, 16, 19, 22, 26, 27, 29, 34,
        16, 16, 22, 24, 27, 29, 34, 37,
        19, 22, 26, 27, 29, 34, 34, 38,
        22, 22, 26, 27, 29, 34, 37, 40,
        22, 26, 27, 29, 32, 35, 40, 48,
        26, 27, 29, 32, 35, 40, 48, 58,
        26, 27, 29, 34, 38, 46, 56, 69,
        27, 29, 35, 38, 46, 56, 69, 83
    }
#endif
;

/* non-linear quantization coefficient table */
EXTERN unsigned char non_linear_mquant_table[32]
#ifdef GLOBAL
    =

```



```

{
    0, 1, 2, 3, 4, 5, 6, 7,
    8,10,12,14,16,18,20,22,
    24,28,32,36,40,44,48,52,
    56,64,72,80,88,96,104,112
}
#endif
;

/* non-linear mquant table for mapping from scale to code
 * since reconstruction levels are not bijective with the index map,
 * it is up to the designer to determine most of the quantization levels
 */

EXTERN unsigned char map_non_linear_mquant[113]
#ifdef GLOBAL
=
{
    0,1,2,3,4,5,6,7,8,8,9,9,10,10,11,11,12,12,13,13,14,14,15,15,16,16,
    16,17,17,17,18,18,18,18,19,19,19,19,20,20,20,20,21,21,21,21,22,22,
    22,22,23,23,23,23,24,24,24,24,24,24,25,25,25,25,25,25,26,26,
    26,26,26,26,26,27,27,27,27,27,27,27,28,28,28,28,28,28,29,
    29,29,29,29,29,29,29,29,30,30,30,30,30,30,31,31,31,31,31
}
#endif
;

/* picture data arrays */

/* reconstructed frames */
EXTERN unsigned char *newrefframe[3], *oldrefframe[3], *auxframe[3];
/* original frames */
EXTERN unsigned char *neworgframe[3], *oldorgframe[3], *auxorgframe[3];
/* prediction of current frame */
EXTERN unsigned char *predframe[3];
/* 8*8 block data */
EXTERN short (*blocks)[64];
/* intra / non_intra quantization matrices */
EXTERN unsigned char intra_q[64], inter_q[64];
EXTERN unsigned char chrom_intra_q[64], chrom_inter_q[64];
/* prediction values for DCT coefficient (0,0) */
EXTERN int dc_dct_pred[3];
/* macroblock side information array */
EXTERN struct mbinf *mbinfo;
/* motion estimation parameters */
EXTERN struct motion_data *motion_data;
/* clipping (=saturation) table */
EXTERN unsigned char *clp;

/* name strings */
EXTERN char id_string[256], tplorg[256], tplref[256];
EXTERN char iqname[256], niqname[256];
EXTERN char statname[256];
EXTERN char errortext[256];

EXTERN FILE *outfile, *statfile; /* file descriptors */
EXTERN int inputtype; /* format of input frames */

EXTERN int quiet; /* suppress warnings */

```

```
/* coding model parameters */
```

```
EXTERN int N; /* number of frames in Group of Pictures */
EXTERN int M; /* distance between I/P frames */
EXTERN int P; /* intra slice refresh interval */
EXTERN int nframes; /* total number of frames to encode */
EXTERN int frame0, tc0; /* number and timecode of first frame */
EXTERN int mpeg1; /* ISO/IEC IS 11172-2 sequence */
EXTERN int fieldpic; /* use field pictures */
```

```
/* sequence specific data (sequence header) */
```

```
EXTERN int horizontal_size, vertical_size; /* frame size (pels) */
EXTERN int width, height; /* encoded frame size (pels) multiples of 16 or 32 */
EXTERN int chrom_width, chrom_height, block_count;
EXTERN int mb_width, mb_height; /* frame size (macroblocks) */
EXTERN int width2, height2, mb_height2, chrom_width2; /* picture size */
EXTERN int aspectratio; /* aspect ratio information (pel or display) */
EXTERN int frame_rate_code; /* coded value of frame rate */
EXTERN double frame_rate; /* frames per second */
EXTERN double bit_rate; /* bits per second */
EXTERN int vbv_buffer_size; /* size of VBV buffer (* 16 kbit) */
EXTERN int constrparms; /* constrained parameters flag (MPEG-1 only) */
EXTERN int load_iquant, load_niquant; /* use non-default quant. matrices */
EXTERN int load_ciquant, load_cniquant;
```

```
/* sequence specific data (sequence extension) */
```

```
EXTERN int profile, level; /* syntax / parameter constraints */
EXTERN int prog_seq; /* progressive sequence */
EXTERN int chroma_format;
EXTERN int low_delay; /* no B pictures, skipped pictures */
```

```
/* sequence specific data (sequence display extension) */
```

```
EXTERN int video_format; /* component, PAL, NTSC, SECAM or MAC */
EXTERN int color_primaries; /* source primary chromaticity coordinates */
EXTERN int transfer_characteristics; /* opto-electronic transfer char. (gamma) */
EXTERN int matrix_coefficients; /* Eg, Eb, Er / Y, Cb, Cr matrix coefficients */
EXTERN int display_horizontal_size, display_vertical_size; /* display size */
```

```
/* picture specific data (picture header) */
```

```
EXTERN int temp_ref; /* temporal reference */
EXTERN int pict_type; /* picture coding type (I, P or B) */
EXTERN int vbv_delay; /* video buffering verifier delay (1/90000 seconds) */
```

```
/* picture specific data (picture coding extension) */
```

```
EXTERN int forw_hor_f_code, forw_vert_f_code;
EXTERN int back_hor_f_code, back_vert_f_code; /* motion vector ranges */
EXTERN int dc_prec; /* DC coefficient precision for intra coded blocks */
EXTERN int pict_struct; /* picture structure (frame, top / bottom field) */
```

```

EXTERN int topfirst; /* display top field first */
/* use only frame prediction and frame DCT (I,P,B,current) */
EXTERN int frame_pred_dct_tab[3], frame_pred_dct;
EXTERN int conceal_tab[3]; /* use concealment motion vectors (I,P,B) */
EXTERN int qscale_tab[3], q_scale_type; /* linear/non-linear quantization table */
EXTERN int intravlc_tab[3], intravlc; /* intra vlc format (I,P,B,current) */
EXTERN int altscan_tab[3], altscan; /* alternate scan (I,P,B,current) */
EXTERN int repeatfirst; /* repeat first field after second field */
EXTERN int prog_frame; /* progressive frame */

```

idct.c

```

/* idct.c, inverse fast discrete cosine transform */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

/*****
 * inverse two dimensional DCT, Chen-Wang algorithm */
 * (cf. IEEE ASSP-32, pp. 803-816, Aug. 1984) */
 * 32-bit integer arithmetic (8 bit coefficients) */
 * 11 mults, 29 adds per DCT */
 * sE, 18.8.91 */
 *****/
/* coefficients extended to 12 bit for IEEE1180-1990 */
/* compliance sE, 2.1.94 */
 *****/

/* this code assumes >> to be a two's-complement arithmetic */
/* right shift: (-2)>>1 == -1 , (-3)>>1 == -2 */

#include "config.h"

```

```

#define W1 2841 /* 2048*sqrt(2)*cos(1*pi/16) */
#define W2 2676 /* 2048*sqrt(2)*cos(2*pi/16) */
#define W3 2408 /* 2048*sqrt(2)*cos(3*pi/16) */
#define W5 1609 /* 2048*sqrt(2)*cos(5*pi/16) */
#define W6 1108 /* 2048*sqrt(2)*cos(6*pi/16) */
#define W7 565 /* 2048*sqrt(2)*cos(7*pi/16) */

/* global declarations */
void init_idct_ANSI_ARGS_((void));
void idct_ANSI_ARGS_((short *block));

/* private data */
static short iclip[1024]; /* clipping table */
static short *iclp;

/* private prototypes */
static void idctrow_ANSI_ARGS_((short *blk));
static void idctcol_ANSI_ARGS_((short *blk));

/* row (horizontal) IDCT
*
* 
$$dst[k] = \sum_{l=0}^7 c[l] * src[l] * \cos\left(\frac{\pi}{8} * (k + \frac{1}{2} - l) * 1\right)$$

*
* where: c[0] = 128
* c[1..7] = 128*sqrt(2)
*/

static void idctrow(blk)
short *blk;
{
    int x0, x1, x2, x3, x4, x5, x6, x7, x8;

    /* shortcut */
    if (!(x1 = blk[4]<<11) | (x2 = blk[6]) | (x3 = blk[2]) |
        (x4 = blk[1]) | (x5 = blk[7]) | (x6 = blk[5]) | (x7 = blk[3]))
    {
        blk[0]=blk[1]=blk[2]=blk[3]=blk[4]=blk[5]=blk[6]=blk[7]=blk[0]<<3;
        return;
    }

    x0 = (blk[0]<<11) + 128; /* for proper rounding in the fourth stage */

    /* first stage */
    x8 = W7*(x4+x5);
    x4 = x8 + (W1-W7)*x4;
    x5 = x8 - (W1+W7)*x5;
    x8 = W3*(x6+x7);
    x6 = x8 - (W3-W5)*x6;
    x7 = x8 - (W3+W5)*x7;

    /* second stage */
    x8 = x0 + x1;
    x0 -= x1;
    x1 = W6*(x3+x2);
    x2 = x1 - (W2+W6)*x2;
    x3 = x1 + (W2-W6)*x3;
    x1 = x4 + x6;

```

```

x4 -= x6;
x6 = x5 + x7;
x5 -= x7;

/* third stage */
x7 = x8 + x3;
x8 -= x3;
x3 = x0 + x2;
x0 -= x2;
x2 = (181*(x4+x5)+128)>>8;
x4 = (181*(x4-x5)+128)>>8;

/* fourth stage */
blk[0] = (x7+x1)>>8;
blk[1] = (x3+x2)>>8;
blk[2] = (x0+x4)>>8;
blk[3] = (x8+x6)>>8;
blk[4] = (x8-x6)>>8;
blk[5] = (x0-x4)>>8;
blk[6] = (x3-x2)>>8;
blk[7] = (x7-x1)>>8;
}

/* column (vertical) IDCT
*
*          7          pi      1
* dst[8*k] = sum c[l] * src[8*l] * cos( -- * ( k + - ) * l )
*          l=0          8      2
*
* where: c[0] = 1/1024
*          c[1..7] = (1/1024)*sqrt(2)
*/
static void idctcol(blk)
short *blk;
{
    int x0, x1, x2, x3, x4, x5, x6, x7, x8;

    /* shortcut */
    if (!(x1 = (blk[8*4]<<8)) | (x2 = blk[8*6]) | (x3 = blk[8*2]) |
        (x4 = blk[8*1]) | (x5 = blk[8*7]) | (x6 = blk[8*5]) | (x7 = blk[8*3])))
    {
        blk[8*0]=blk[8*1]=blk[8*2]=blk[8*3]=blk[8*4]=blk[8*5]=blk[8*6]=blk[8*7]=
            iclp[(blk[8*0]+32)>>6];
        return;
    }

    x0 = (blk[8*0]<<8) + 8192;

    /* first stage */
    x8 = W7*(x4+x5) + 4;
    x4 = (x8+(W1-W7)*x4)>>3;
    x5 = (x8-(W1+W7)*x5)>>3;
    x8 = W3*(x6+x7) + 4;
    x6 = (x8-(W3-W5)*x6)>>3;
    x7 = (x8-(W3+W5)*x7)>>3;

    /* second stage */
    x8 = x0 + x1;
    x0 -= x1;

```

```

x1 = W6*(x3+x2) + 4;
x2 = (x1-(W2+W6)*x2)>>3;
x3 = (x1+(W2-W6)*x3)>>3;
x1 = x4 + x6;
x4 -= x6;
x6 = x5 + x7;
x5 -= x7;

```

```

/* third stage */
x7 = x8 + x3;
x8 -= x3;
x3 = x0 + x2;
x0 -= x2;
x2 = (181*(x4+x5)+128)>>8;
x4 = (181*(x4-x5)+128)>>8;

```

```

/* fourth stage */
blk[8*0] = iclp[(x7+x1)>>14];
blk[8*1] = iclp[(x3+x2)>>14];
blk[8*2] = iclp[(x0+x4)>>14];
blk[8*3] = iclp[(x8+x6)>>14];
blk[8*4] = iclp[(x8-x6)>>14];
blk[8*5] = iclp[(x0-x4)>>14];
blk[8*6] = iclp[(x3-x2)>>14];
blk[8*7] = iclp[(x7-x1)>>14];
}

```

```

/* two dimensional inverse discrete cosine transform */

```

```

void idct(block)
short *block;
{
    int i;

    for (i=0; i<8; i++)
        idctrow(block+8*i);

    for (i=0; i<8; i++)
        idctcol(block+i);
}

```

```

void init_idct()
{
    int i;

    iclp = iclip+512;
    for (i= -512; i<512; i++)
        iclp[i] = (i<-256) ? -256 : ((i>255) ? 255 : i);
}

```

```

makefile

```

```

# Makefile for mpeg2encode

```

```

# Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
# any and all warranties, whether express, implied, or statutory, 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.
#
# The MPEG Software Simulation Group 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.
#
#
# select one of the following CC CFLAGS settings

#
# GNU gcc
#
CC = gcc
CFLAGS = -O2

#
# SPARCworks acc
#
#CC = acc
#CFLAGS = -fast

#
# SUN cc
#
#CC = cc
#CFLAGS = -O3 -DNON_ANSI_COMPILER

# the following are user contributed configurations

# DEC Alpha cc
#CC = cc
#CFLAGS = -O -Olimit 1000

# HP715
#CC = cc
#CFLAGS = -Aa -D_HPUX_SOURCE

# SGI Irix 5.2
#CC = cc
#CFLAGS = -O2 -sopt

OBJ = mpeg2enc.o conform.o putseq.o putpic.o puthdr.o putmpg.o putvlc.o putbits.o motion.o predict.o
readpic.o writepic.o transfrm.o fdctref.o idct.o quantize.o ratectl.o stats.o

```

all: mpeg2encode

pc: mpeg2enc.exe

clean:

rm -f \*.o \*% core mpeg2encode

mpeg2enc.exe: mpeg2encode

coff2exe mpeg2enc

mpeg2encode: \$(OBJ)

\$(CC) \$(CFLAGS) -o mpeg2encode \$(OBJ) -lm

conform.o : conform.c config.h global.h mpeg2enc.h

fdctref.o : fdctref.c config.h

idct.o : idct.c config.h

motion.o : motion.c config.h global.h mpeg2enc.h

mpeg2enc.o : mpeg2enc.c config.h global.h mpeg2enc.h

predict.o : predict.c config.h global.h mpeg2enc.h

putbits.o : putbits.c config.h

puthdr.o : puthdr.c config.h global.h mpeg2enc.h

putmpg.o : putmpg.c config.h global.h mpeg2enc.h

putpic.o : putpic.c config.h global.h mpeg2enc.h

putseq.o : putseq.c config.h global.h mpeg2enc.h

putvlc.o : putvlc.c config.h global.h mpeg2enc.h vlc.h

quantize.o : quantize.c config.h global.h mpeg2enc.h

ratectl.o : ratectl.c config.h global.h mpeg2enc.h

readpic.o : readpic.c config.h global.h mpeg2enc.h

stats.o : stats.c config.h global.h mpeg2enc.h

transfrm.o : transfrm.c config.h global.h mpeg2enc.h

writepic.o : writepic.c config.h global.h mpeg2enc.h

motion.c

/\* motion.c, motion estimation

\*/

/\* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims  
 \* any and all warranties, whether express, implied, or statutory, 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.

\*

\* The MPEG Software Simulation Group 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.
*
*/

```

```

#include <stdio.h>
#include "config.h"
#include "global.h"

```

```

/* private prototypes */

```

```

static void frame_ME_ANSI_ARGS_((unsigned char *oldorg, unsigned char *neworg,
    unsigned char *oldref, unsigned char *newref, unsigned char *cur,
    int i, int j, int sxf, int syf, int sxb, int syb, struct mbinf *mbi));

```

```

static void field_ME_ANSI_ARGS_((unsigned char *oldorg, unsigned char *neworg,
    unsigned char *oldref, unsigned char *newref, unsigned char *cur,
    unsigned char *curref, int i, int j, int sxf, int syf, int sxb, int syb,
    struct mbinf *mbi, int secondfield, int ipflag));

```

```

static void frame_estimate_ANSI_ARGS_((unsigned char *org,
    unsigned char *ref, unsigned char *mb,
    int i, int j,
    int sx, int sy, int *iminp, int *jminp, int *imintp, int *jmintp,
    int *iminbp, int *jminbp, int *dframep, int *dfieldp,
    int *tselp, int *bselp, int imins[2][2], int jmins[2][2]));

```

```

static void field_estimate_ANSI_ARGS_((unsigned char *toporg,
    unsigned char *topref, unsigned char *botorg, unsigned char *botref,
    unsigned char *mb, int i, int j, int sx, int sy, int ipflag,
    int *iminp, int *jminp, int *imin8up, int *jmin8up, int *imin8lp,
    int *jmin8lp, int *dfieldp, int *d8p, int *selp, int *sel8up, int *sel8lp,
    int *iminsp, int *jminsp, int *dsp));

```

```

static void dpframe_estimate_ANSI_ARGS_((unsigned char *ref,
    unsigned char *mb, int i, int j, int iminf[2][2], int jminf[2][2],
    int *iminp, int *jminp, int *imindmvp, int *jmindmvp,
    int *dmcp, int *vmcp));

```

```

static void dpfield_estimate_ANSI_ARGS_((unsigned char *topref,
    unsigned char *botref, unsigned char *mb,
    int i, int j, int imins, int jmins, int *imindmvp, int *jmindmvp,
    int *dmcp, int *vmcp));

```

```

static int fullsearch_ANSI_ARGS_((unsigned char *org, unsigned char *ref,
    unsigned char *blk,
    int lx, int i0, int j0, int sx, int sy, int h, int xmax, int ymax,
    int *iminp, int *jminp));

```

```

static int dist1_ANSI_ARGS_((unsigned char *blk1, unsigned char *blk2,
    int lx, int hx, int hy, int h, int distlim));

```

```

static int dist2_ANSI_ARGS_((unsigned char *blk1, unsigned char *blk2,
    int lx, int hx, int hy, int h));

```

```

static int bdist1_ANSI_ARGS_((unsigned char *pf, unsigned char *pb,
    unsigned char *p2, int lx, int hxf, int hyf, int hxb, int hyb, int h));

```

```

static int bdist2 _ANSI_ARGS_((unsigned char *pf, unsigned char *pb,
    unsigned char *p2, int lx, int hxf, int hyf, int hxb, int hyb, int h));

static int variance _ANSI_ARGS_((unsigned char *p, int lx));

/*
 * motion estimation for progressive and interlaced frame pictures
 *
 * oldorg: source frame for forward prediction (used for P and B frames)
 * neworg: source frame for backward prediction (B frames only)
 * oldref: reconstructed frame for forward prediction (P and B frames)
 * newref: reconstructed frame for backward prediction (B frames only)
 * cur: current frame (the one for which the prediction is formed)
 * sxf,syf: forward search window (frame coordinates)
 * sxb,syb: backward search window (frame coordinates)
 * mbi: pointer to macroblock info structure
 *
 * results:
 * mbi->
 *   mb_type: 0, MB_INTRA, MB_FORWARD, MB_BACKWARD, MB_FORWARD|MB_BACKWARD
 *   MV[][][]: motion vectors (frame format)
 *   mv_field_sel: top/bottom field (for field prediction)
 *   motion_type: MC_FRAME, MC_FIELD
 *
 * uses global vars: pict_type, frame_pred_dct
 */
void motion_estimation(oldorg,neworg,oldref,newref,cur,curref,
    sxf,syf,sxb,syb,mbi,secondfield,ipflag)
unsigned char *oldorg,*neworg,*oldref,*newref,*cur,*curref;
int sxf,syf,sxb,syb;
struct mbinfo *mbi;
int secondfield,ipflag;
{
    int i,j;

    /* loop through all macroblocks of the picture */
    for (j=0; j<height2; j+=16)
    {
        for (i=0; i<width; i+=16)
        {
            if (pict_struct==FRAME_PICTURE)
                frame_ME(oldorg,neworg,oldref,newref,cur,i,j,sxf,syf,sxb,syb,mbi);
            else
                field_ME(oldorg,neworg,oldref,newref,cur,curref,i,j,sxf,syf,sxb,syb,
                    mbi,secondfield,ipflag);
            mbi++;
        }
    }

    if (!quiet)
    {
        putc('.',stderr);
        fflush(stderr);
    }
}
if (!quiet)
    putc('\n',stderr);
}

```

```

static void frame_ME(oldorg,neworg,oldref,newref,cur,i,j,sxf,syf,sxb,syb,mbi)
unsigned char *oldorg,*neworg,*oldref,*newref,*cur;
int i,j,sxf,syf,sxb,syb;
struct mbinfo *mbi;
{
    int imin,jmin,iminr,jminr,iminb,jminb;
    int imint,jmint,iminb,jminb;
    int imintf,jmintf,iminbf,jminbf;
    int imintr,jmintr,iminbr,jminbr;
    int var,v0;
    int dmc,dmcf,dmcr,dmci,vmc,vmcf,vmcr,vmci;
    int dmcfield,dmcfeldf,dmcfeldr,dmcfeldi;
    int tsel,bsel,tself,bself,tselfr,bselfr;
    unsigned char *mb;
    int imins[2][2],jmins[2][2];
    int imindp,jmindp,imindmv,jmindmv,dmc_dp,vmc_dp;

    mb = cur + i + width*j;

    var = variance(mb,width);

    if (pict_type==I_TYPE)
        mbi->mb_type = MB_INTRA;
    else if (pict_type==P_TYPE)
    {
        if (frame_pred_dct)
        {
            dmc = fullsearch(oldorg,oldref,mb,
                            width,i,j,sxf,syf,16,width,height,&imin,&jmin);
            vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
                        width,imin&1,jmin&1,16);
            mbi->motion_type = MC_FRAME;
        }
        else
        {
            frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
                           &imin,&jmin,&imint,&jmint,&iminb,&jminb,
                           &dmc,&dmcfeld,&tsel,&bsel,imins,jmins);

            if (M==1)
                dpframe_estimate(oldref,mb,i,j>>1,imins,jmins,
                                &imindp,&jmindp,&imindmv,&jmindmv,&dmc_dp,&vmc_dp);

            /* select between dual prime, frame and field prediction */
            if (M==1 && dmc_dp<dmc && dmc_dp<dmcfeld)
            {
                mbi->motion_type = MC_DMV;
                dmc = dmc_dp;
                vmc = vmc_dp;
            }
            else if (dmc<=dmcfeld)
            {
                mbi->motion_type = MC_FRAME;
                vmc = dist2(oldref+(imin>>1)+width*(jmin>>1),mb,
                            width,imin&1,jmin&1,16);
            }
            else
            {
                mbi->motion_type = MC_FIELD;
            }
        }
    }
}

```

```

    dmc = dmcfield;
    vmc = dist2(oldref+(tsel?width:0)+(imint>>1)+(width<<1)*(jmint>>1),
               mb,width<<1,imint&1,jmint&1,8);
    vmc+= dist2(oldref+(bsel?width:0)+(iminb>>1)+(width<<1)*(jminb>>1),
               mb+width,width<<1,iminb&1,jminb&1,8);
}
}

/* select between intra or non-intra coding:
 *
 * selection is based on intra block variance (var) vs.
 * prediction error variance (vmc)
 *
 * blocks with small prediction error are always coded non-intra
 * even if variance is smaller (is this reasonable?)
 */
if (vmc>var && vmc>=9*256)
    mbi->mb_type = MB_INTRA;
else
{
    /* select between MC / No-MC
     *
     * use No-MC if var(No-MC) <= 1.25*var(MC)
     * (i.e slightly biased towards No-MC)
     *
     * blocks with small prediction error are always coded as No-MC
     * (requires no motion vectors, allows skipping)
     */
    v0 = dist2(oldref+i+width*j,mb,width,0,0,16);
    if (4*v0>5*vmc && v0>=9*256)
    {
        /* use MC */
        var = vmc;
        mbi->mb_type = MB_FORWARD;
        if (mbi->motion_type==MC_FRAME)
        {
            mbi->MV[0][0][0] = imin - (i<<1);
            mbi->MV[0][0][1] = jmin - (j<<1);
        }
        else if (mbi->motion_type==MC_DMV)
        {
            /* these are FRAME vectors */
            /* same parity vector */
            mbi->MV[0][0][0] = imindp - (i<<1);
            mbi->MV[0][0][1] = (jmindp<<1) - (j<<1);

            /* opposite parity vector */
            mbi->dmvector[0] = imindmv;
            mbi->dmvector[1] = jmindmv;
        }
    }
    else
    {
        /* these are FRAME vectors */
        mbi->MV[0][0][0] = imint - (i<<1);
        mbi->MV[0][0][1] = (jmint<<1) - (j<<1);
        mbi->MV[1][0][0] = iminb - (i<<1);
        mbi->MV[1][0][1] = (jminb<<1) - (j<<1);
        mbi->mv_field_sel[0][0] = tsel;
        mbi->mv_field_sel[1][0] = bsel;
    }
}

```

```

    }
  }
else
{
  /* No-MC */
  var = v0;
  mbi->mb_type = 0;
  mbi->motion_type = MC_FRAME;
  mbi->MV[0][0][0] = 0;
  mbi->MV[0][0][1] = 0;
}
}
}
else /* if (pict_type==B_TYPE) */
{
  if (frame_pred_dct)
  {
    /* forward */
    dmc = fullsearch(oldorg,oldref,mb,
                    width,i,j,sxf,syf,16,width,height,&iminf,&jminf);
    vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
              width,iminf&1,jminf&1,16);

    /* backward */
    dmc = fullsearch(neworg,newref,mb,
                    width,i,j,sxb,syb,16,width,height,&iminr,&jminr);
    vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
              width,iminr&1,jminr&1,16);

    /* interpolated (bidirectional) */
    vmci = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
                 newref+(iminr>>1)+width*(jminr>>1),
                 mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

    /* decisions */

    /* select between forward/backward/interpolated prediction:
     * use the one with smallest mean squared prediction error
     */
    if (vmc<=vmc && vmc<=vmci)
    {
      vmc = vmc;
      mbi->mb_type = MB_FORWARD;
    }
    else if (vmc<=vmci)
    {
      vmc = vmc;
      mbi->mb_type = MB_BACKWARD;
    }
    else
    {
      vmc = vmci;
      mbi->mb_type = MB_FORWARD|MB_BACKWARD;
    }

    mbi->motion_type = MC_FRAME;
  }
  else
  {

```

```

/* forward prediction */
frame_estimate(oldorg,oldref,mb,i,j,sxf,syf,
    &iminf,&jminf,&imintf,&jmintf,&iminbf,&jminbf,
    &dmcf,&dmcfieldf,&tself,&bself,imins,jmins);

/* backward prediction */
frame_estimate(neworg,newref,mb,i,j,sxb,syb,
    &iminr,&jminr,&imintr,&jmintr,&iminbr,&jminbr,
    &dmcr,&dmcfieldr,&tself,&bself,imins,jmins);

/* calculate interpolated distance */
/* frame */
dmci = bdist1(oldref+(iminf>>1)+width*(jminf>>1),
    newref+(iminr>>1)+width*(jminr>>1),
    mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);

/* top field */
dmcfieldi = bdist1(
    oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
    newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
    mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);

/* bottom field */
dmcfieldi+= bdist1(
    oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
    newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
    mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);

/* select prediction type of minimum distance from the
 * six candidates (field/frame * forward/backward/interpolated)
 */
if (dmci<dmcfieldi && dmci<dmcf && dmci<dmcfieldf
    && dmci<dmcr && dmci<dmcfieldr)
{
    /* frame, interpolated */
    mbi->mb_type = MB_FORWARD|MB_BACKWARD;
    mbi->motion_type = MC_FRAME;
    vmc = bdist2(oldref+(iminf>>1)+width*(jminf>>1),
        newref+(iminr>>1)+width*(jminr>>1),
        mb,width,iminf&1,jminf&1,iminr&1,jminr&1,16);
}
else if (dmcfieldi<dmcf && dmcfieldi<dmcfieldf
    && dmcfieldi<dmcr && dmcfieldi<dmcfieldr)
{
    /* field, interpolated */
    mbi->mb_type = MB_FORWARD|MB_BACKWARD;
    mbi->motion_type = MC_FIELD;
    vmc = bdist2(oldref+(imintf>>1)+(tself?width:0)+(width<<1)*(jmintf>>1),
        newref+(imintr>>1)+(tselr?width:0)+(width<<1)*(jmintr>>1),
        mb,width<<1,imintf&1,jmintf&1,imintr&1,jmintr&1,8);
    vmc+= bdist2(oldref+(iminbf>>1)+(bself?width:0)+(width<<1)*(jminbf>>1),
        newref+(iminbr>>1)+(bselr?width:0)+(width<<1)*(jminbr>>1),
        mb+width,width<<1,iminbf&1,jminbf&1,iminbr&1,jminbr&1,8);
}
else if (dmcf<dmcfieldf && dmcf<dmcr && dmcf<dmcfieldr)
{
    /* frame, forward */
    mbi->mb_type = MB_FORWARD;
    mbi->motion_type = MC_FRAME;

```

```

        vmc = dist2(oldref+(iminf>>1)+width*(jminf>>1),mb,
                    width,iminf&1,jminf&1,16);
    }
    else if (dmcfieldf<dmcrc && dmcfieldf<dmcfieldr)
    {
        /* field, forward */
        mbi->mb_type = MB_FORWARD;
        mbi->motion_type = MC_FIELD;
        vmc = dist2(oldref+(tsel?width:0)+(imintf>>1)+(width<<1)*(jmintf>>1),
                    mb,width<<1,imintf&1,jmintf&1,8);
        vmc+= dist2(oldref+(bsel?width:0)+(iminbf>>1)+(width<<1)*(jminbf>>1),
                    mb+width,width<<1,iminbf&1,jminbf&1,8);
    }
    else if (dmcrc<dmcfieldr)
    {
        /* frame, backward */
        mbi->mb_type = MB_BACKWARD;
        mbi->motion_type = MC_FRAME;
        vmc = dist2(newref+(iminr>>1)+width*(jminr>>1),mb,
                    width,iminr&1,jminr&1,16);
    }
    else
    {
        /* field, backward */
        mbi->mb_type = MB_BACKWARD;
        mbi->motion_type = MC_FIELD;
        vmc = dist2(newref+(tsel?width:0)+(imintr>>1)+(width<<1)*(jmintr>>1),
                    mb,width<<1,imintr&1,jmintr&1,8);
        vmc+= dist2(newref+(bsel?width:0)+(iminbr>>1)+(width<<1)*(jminbr>>1),
                    mb+width,width<<1,iminbr&1,jminbr&1,8);
    }
}

/* select between intra or non-intra coding:
*
* selection is based on intra block variance (var) vs.
* prediction error variance (vmc)
*
* blocks with small prediction error are always coded non-intra
* even if variance is smaller (is this reasonable?)
*/
if (vmc>var && vmc>=9*256)
    mbi->mb_type = MB_INTRA;
else
{
    var = vmc;
    if (mbi->motion_type==MC_FRAME)
    {
        /* forward */
        mbi->MV[0][0][0] = iminf - (i<<1);
        mbi->MV[0][0][1] = jminf - (j<<1);
        /* backward */
        mbi->MV[0][1][0] = iminr - (i<<1);
        mbi->MV[0][1][1] = jminr - (j<<1);
    }
    else
    {
        /* these are FRAME vectors */
        /* forward */

```

```

    mbi->MV[0][0][0] = imintf - (i<<1);
    mbi->MV[0][0][1] = (jintf<<1) - (j<<1);
    mbi->MV[1][0][0] = iminbf - (i<<1);
    mbi->MV[1][0][1] = (jminbf<<1) - (j<<1);
    mbi->mv_field_sel[0][0] = tself;
    mbi->mv_field_sel[1][0] = bself;
    /* backward */
    mbi->MV[0][1][0] = imintr - (i<<1);
    mbi->MV[0][1][1] = (jmintr<<1) - (j<<1);
    mbi->MV[1][1][0] = iminbr - (i<<1);
    mbi->MV[1][1][1] = (jminbr<<1) - (j<<1);
    mbi->mv_field_sel[0][1] = tselr;
    mbi->mv_field_sel[1][1] = bselr;
}
}
}

mbi->var = var;
}

/*
 * motion estimation for field pictures
 *
 * oldorg: original frame for forward prediction (P and B frames)
 * neworg: original frame for backward prediction (B frames only)
 * oldref: reconstructed frame for forward prediction (P and B frames)
 * newref: reconstructed frame for backward prediction (B frames only)
 * cur: current original frame (the one for which the prediction is formed)
 * curref: current reconstructed frame (to predict second field from first)
 * sxf,syf: forward search window (frame coordinates)
 * sxb,syb: backward search window (frame coordinates)
 * mbi: pointer to macroblock info structure
 * secondfield: indicates second field of a frame (in P fields this means
 * that reference field of opposite parity is in curref instead
 * of oldref)
 * ipflag: indicates a P type field which is the second field of a frame
 * in which the first field is I type (this restricts predictions
 * to be based only on the opposite parity (=I) field)
 *
 * results:
 * mbi->
 * mb_type: 0, MB_INTRA, MB_FORWARD, MB_BACKWARD, MB_FORWARD|MB_BACKWARD
 * MV[][][]: motion vectors (field format)
 * mv_field_sel: top/bottom field
 * motion_type: MC_FIELD, MC_16X8
 *
 * uses global vars: pict_type, pict_struct
 */
static void field_ME(oldorg,neworg,oldref,newref,cur,curref,i,j,
    sxf,syf,sxb,syb,mbi,secondfield,ipflag)
unsigned char *oldorg,*neworg,*oldref,*newref,*cur,*curref;
int i,j,sxf,syf,sxb,syb;
struct mbinfo *mbi;
int secondfield,ipflag;
{
    int w2;
    unsigned char *mb,*toporg,*topref,*botorg,*botref;
    int var,vmc,v0,dmc,dmcfldi,dmc8i;
    int imin,jmin,imin8u,jmin8u,imin8l,jmin8l,dmcfld,dmc8,sel,sel8u,sel8l;

```



```

int iminf,jminf,imin8uf,jmin8uf,imin8lf,jmin8lf,dmcfieldf,dmc8f,self,sel8uf,sel8lf;
int iminr,jminr,imin8ur,jmin8ur,imin8lr,jmin8lr,dmcfieldr,dmc8r,selr,sel8ur,sel8lr;
int imins,jmins,ds,imindmv,jmindmv,vmc_dp,dmc_dp;

```

```

w2 = width<<1;

```

```

mb = cur + i + w2*j;
if (pict_struct==BOTTOM_FIELD)
    mb += width;

```

```

var = variance(mb,w2);

```

```

if (pict_type==I_TYPE)
    mbi->mb_type = MB_INTRA;
else if (pict_type==P_TYPE)
{
    toporg = oldorg;
    topref = oldref;
    botorg = oldorg + width;
    botref = oldref + width;

```

```

    if (secondfield)
    {
        /* opposite parity field is in same frame */
        if (pict_struct==TOP_FIELD)
        {
            /* current is top field */
            botorg = cur + width;
            botref = curref + width;
        }
        else
        {
            /* current is bottom field */
            toporg = cur;
            topref = curref;
        }
    }
}

```

```

field_estimate(toporg,topref,botorg,botref,mb,i,j,sxf,syf,ipflag,
               &imin,&jmin,&imin8u,&jmin8u,&imin8l,&jmin8l,
               &dmcfield,&dmc8,&sel,&sel8u,&sel8l,&imins,&jmins,&ds);

```

```

if (M==1 && !ipflag) /* generic condition which permits Dual Prime */
    dpfield_estimate(topref,botref,mb,i,j,imins,jmins,&imindmv,&jmindmv,
                    &dmc_dp,&vmc_dp);

```

```

/* select between dual prime, field and 16x8 prediction */
if (M==1 && !ipflag && dmc_dp<dmc8 && dmc_dp<dmcfield)
{
    /* Dual Prime prediction */
    mbi->motion_type = MC_DMV;
    dmc = dmc_dp; /* L1 metric */
    vmc = vmc_dp; /* we already calculated L2 error for Dual */

```

```

}
else if (dmc8<dmcfield)
{
    /* 16x8 prediction */
    mbi->motion_type = MC_16X8;

```

```

/* upper half block */
vmc = dist2((sel8u?botref:topref) + (imin8u>>1) + w2*(jmin8u>>1),
            mb,w2,imin8u&1,jmin8u&1,8);
/* lower half block */
vmc+= dist2((sel8l?botref:topref) + (imin8l>>1) + w2*(jmin8l>>1),
            mb+8*w2,w2,imin8l&1,jmin8l&1,8);
}
else
{
/* field prediction */
mbi->motion_type = MC_FIELD;
vmc = dist2((sel?botref:topref) + (imin>>1) + w2*(jmin>>1),
            mb,w2,imin&1,jmin&1,16);
}

/* select between intra and non-intra coding */
if (vmc>var && vmc>=9*256)
    mbi->mb_type = MB_INTRA;
else
{
/* zero MV field prediction from same parity ref. field
 * (not allowed if ipflag is set)
 */
if (!ipflag)
    v0 = dist2(((pict_struct==BOTTOM_FIELD)?botref:topref) + i + w2*j,
                mb,w2,0,0,16);
if (ipflag || (4*v0>5*vmc && v0>=9*256))
{
    var = vmc;
    mbi->mb_type = MB_FORWARD;
    if (mbi->motion_type==MC_FIELD)
    {
        mbi->MV[0][0][0] = imin - (i<<1);
        mbi->MV[0][0][1] = jmin - (j<<1);
        mbi->mv_field_sel[0][0] = sel;
    }
    else if (mbi->motion_type==MC_DMV)
    {
        /* same parity vector */
        mbi->MV[0][0][0] = imins - (i<<1);
        mbi->MV[0][0][1] = jmins - (j<<1);

        /* opposite parity vector */
        mbi->dmvector[0] = imindmv;
        mbi->dmvector[1] = jmindmv;
    }
}
else
{
    mbi->MV[0][0][0] = imin8u - (i<<1);
    mbi->MV[0][0][1] = jmin8u - (j<<1);
    mbi->MV[1][0][0] = imin8l - (i<<1);
    mbi->MV[1][0][1] = jmin8l - ((j+8)<<1);
    mbi->mv_field_sel[0][0] = sel8u;
    mbi->mv_field_sel[1][0] = sel8l;
}
}
else
{
/* No MC */

```

```

    var = v0;
    mbi->mb_type = 0;
    mbi->motion_type = MC_FIELD;
    mbi->MV[0][0][0] = 0;
    mbi->MV[0][0][1] = 0;
    mbi->mv_field_sel[0][0] = (pict_struct==BOTTOM_FIELD);
}
}
}
else /* if (pict_type==B_TYPE) */
{
    /* forward prediction */
    field_estimate(oldorg,oldref,oldorg+width,oldref+width,mb,
        i,j,sxf,syf,0,
        &iminf,&jminf,&imin8uf,&jmin8uf,&imin8lf,&jmin8lf,
        &dmcfieldf,&dmc8f,&self,&sel8uf,&sel8lf,&imins,&jmins,&ds);

    /* backward prediction */
    field_estimate(neworg,newref,neworg+width,newref+width,mb,
        i,j,sxf,syf,0,
        &iminr,&jminr,&imin8ur,&jmin8ur,&imin8lr,&jmin8lr,
        &dmcfieldr,&dmc8r,&selr,&sel8ur,&sel8lr,&imins,&jmins,&ds);

    /* calculate distances for bidirectional prediction */
    /* field */
    dmcfieldi = bdist1(oldref + (self?width:0) + (iminf>>1) + w2*(jminf>>1),
        newref + (selr?width:0) + (iminr>>1) + w2*(jminr>>1),
        mb,w2,iminf&1,jminf&1,iminr&1,jminr&1,16);

    /* 16x8 upper half block */
    dmc8i = bdist1(oldref + (sel8uf?width:0) + (imin8uf>>1) + w2*(jmin8uf>>1),
        newref + (sel8ur?width:0) + (imin8ur>>1) + w2*(jmin8ur>>1),
        mb,w2,imin8uf&1,jmin8uf&1,imin8ur&1,jmin8ur&1,8);

    /* 16x8 lower half block */
    dmc8i+= bdist1(oldref + (sel8lf?width:0) + (imin8lf>>1) + w2*(jmin8lf>>1),
        newref + (sel8lr?width:0) + (imin8lr>>1) + w2*(jmin8lr>>1),
        mb+8*w2,w2,imin8lf&1,jmin8lf&1,imin8lr&1,jmin8lr&1,8);

    /* select prediction type of minimum distance */
    if (dmcfieldi<dmc8i && dmcfieldi<dmcfieldf && dmcfieldi<dmc8f
        && dmcfieldi<dmcfieldr && dmcfieldi<dmc8r)
    {
        /* field, interpolated */
        mbi->mb_type = MB_FORWARD|MB_BACKWARD;
        mbi->motion_type = MC_FIELD;
        vmc = bdist2(oldref + (self?width:0) + (iminf>>1) + w2*(jminf>>1),
            newref + (selr?width:0) + (iminr>>1) + w2*(jminr>>1),
            mb,w2,iminf&1,jminf&1,iminr&1,jminr&1,16);
    }
    else if (dmc8i<dmcfieldf && dmc8i<dmc8f
        && dmc8i<dmcfieldr && dmc8i<dmc8r)
    {
        /* 16x8, interpolated */
        mbi->mb_type = MB_FORWARD|MB_BACKWARD;
        mbi->motion_type = MC_16X8;

        /* upper half block */
        vmc = bdist2(oldref + (sel8uf?width:0) + (imin8uf>>1) + w2*(jmin8uf>>1),

```

```

        newref + (sel8ur?width:0) + (imin8ur>>1) + w2*(jmin8ur>>1),
        mb,w2,imin8uf&1,jmin8uf&1,imin8ur&1,jmin8ur&1,8);

/* lower half block */
vmc+= bdist2(oldref + (sel8lf?width:0) + (imin8lf>>1) + w2*(jmin8lf>>1),
        newref + (sel8lr?width:0) + (imin8lr>>1) + w2*(jmin8lr>>1),
        mb+8*w2,w2,imin8lf&1,jmin8lf&1,imin8lr&1,jmin8lr&1,8);
}
else if (dmcfieldf<dmc8f && dmcfieldf<dmcfieldr && dmcfieldf<dmc8r)
{
    /* field, forward */
    mbi->mb_type = MB_FORWARD;
    mbi->motion_type = MC_FIELD;
    vmc = dist2(oldref + (self?width:0) + (iminf>>1) + w2*(jminf>>1),
        mb,w2,iminf&1,jminf&1,16);
}
else if (dmc8f<dmcfieldr && dmc8f<dmc8r)
{
    /* 16x8, forward */
    mbi->mb_type = MB_FORWARD;
    mbi->motion_type = MC_16X8;

    /* upper half block */
    vmc = dist2(oldref + (sel8uf?width:0) + (imin8uf>>1) + w2*(jmin8uf>>1),
        mb,w2,imin8uf&1,jmin8uf&1,8);

    /* lower half block */
    vmc+= dist2(oldref + (sel8lf?width:0) + (imin8lf>>1) + w2*(jmin8lf>>1),
        mb+8*w2,w2,imin8lf&1,jmin8lf&1,8);
}
else if (dmcfieldr<dmc8r)
{
    /* field, backward */
    mbi->mb_type = MB_BACKWARD;
    mbi->motion_type = MC_FIELD;
    vmc = dist2(newref + (selr?width:0) + (iminr>>1) + w2*(jminr>>1),
        mb,w2,iminr&1,jminr&1,16);
}
else
{
    /* 16x8, backward */
    mbi->mb_type = MB_BACKWARD;
    mbi->motion_type = MC_16X8;

    /* upper half block */
    vmc = dist2(newref + (sel8ur?width:0) + (imin8ur>>1) + w2*(jmin8ur>>1),
        mb,w2,imin8ur&1,jmin8ur&1,8);

    /* lower half block */
    vmc+= dist2(newref + (sel8lr?width:0) + (imin8lr>>1) + w2*(jmin8lr>>1),
        mb+8*w2,w2,imin8lr&1,jmin8lr&1,8);
}

/* select between intra and non-intra coding */
if (vmc>var && vmc>=9*256)
    mbi->mb_type = MB_INTRA;
else
{
    var = vmc;
}

```

```

    if (mbi->motion_type==MC_FIELD)
    {
        /* forward */
        mbi->MV[0][0][0] = iminf - (i<<1);
        mbi->MV[0][0][1] = jminf - (j<<1);
        mbi->mv_field_sel[0][0] = self;
        /* backward */
        mbi->MV[0][1][0] = iminr - (i<<1);
        mbi->MV[0][1][1] = jminr - (j<<1);
        mbi->mv_field_sel[0][1] = selr;
    }
    else /* MC_16X8 */
    {
        /* forward */
        mbi->MV[0][0][0] = imin8uf - (i<<1);
        mbi->MV[0][0][1] = jmin8uf - (j<<1);
        mbi->mv_field_sel[0][0] = sel8uf;
        mbi->MV[1][0][0] = imin8lf - (i<<1);
        mbi->MV[1][0][1] = jmin8lf - ((j+8)<<1);
        mbi->mv_field_sel[1][0] = sel8lf;
        /* backward */
        mbi->MV[0][1][0] = imin8ur - (i<<1);
        mbi->MV[0][1][1] = jmin8ur - (j<<1);
        mbi->mv_field_sel[0][1] = sel8ur;
        mbi->MV[1][1][0] = imin8lr - (i<<1);
        mbi->MV[1][1][1] = jmin8lr - ((j+8)<<1);
        mbi->mv_field_sel[1][1] = sel8lr;
    }
}
}

mbi->var = var;
}

/*
 * frame picture motion estimation
 *
 * org: top left pel of source reference frame
 * ref: top left pel of reconstructed reference frame
 * mb: macroblock to be matched
 * i,j: location of mb relative to ref (=center of search window)
 * sx,sy: half widths of search window
 * iminp,jminp,dframep: location and value of best frame prediction
 * imintp,jmintp,tselp: location of best field pred. for top field of mb
 * iminbp,jminbp,bselp: location of best field pred. for bottom field of mb
 * dfieldp: value of field prediction
 */
static void frame_estimate(org,ref,mb,i,j,sx,sy,
    iminp,jminp,imintp,jmintp,iminbp,jminbp,dframep,dfieldp,tselp,bselp,
    imins,jmins)
unsigned char *org,*ref,*mb;
int i,j,sx,sy;
int *iminp,*jminp;
int *imintp,*jmintp,*iminbp,*jminbp;
int *dframep,*dfieldp;
int *tselp,*bselp;
int imins[2][2],jmins[2][2];
{
    int dt,db,dmint,dminb;

```

```

int imint,iminb,jmint,jminb;

/* frame prediction */
*dframep = fullsearch(org,ref,mb,width,i,j,sx,sy,16,width,height,
    iminp,jminp);

/* predict top field from top field */
dt = fullsearch(org,ref,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
    &imint,&jmint);

/* predict top field from bottom field */
db = fullsearch(org+width,ref+width,mb,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
    &iminb,&jminb);

imins[0][0] = imint;
jmins[0][0] = jmint;
imins[1][0] = iminb;
jmins[1][0] = jminb;

/* select prediction for top field */
if (dt<=db)
{
    dmint=dt; *imintp=imint; *jmintp=jmint; *tsel=0;
}
else
{
    dmint=db; *imintp=iminb; *jmintp=jminb; *tsel=1;
}

/* predict bottom field from top field */
dt = fullsearch(org,ref,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
    &imint,&jmint);

/* predict bottom field from bottom field */
db = fullsearch(org+width,ref+width,mb+width,width<<1,i,j>>1,sx,sy>>1,8,width,height>>1,
    &iminb,&jminb);

imins[0][1] = imint;
jmins[0][1] = jmint;
imins[1][1] = iminb;
jmins[1][1] = jminb;

/* select prediction for bottom field */
if (db<=dt)
{
    dminb=db; *iminbp=iminb; *jminbp=jminb; *bselp=1;
}
else
{
    dminb=dt; *iminbp=imint; *jminbp=jmint; *bselp=0;
}

*dfieldp=dmint+dminb;
}

/*
* field picture motion estimation subroutine
*
* toporg: address of original top reference field

```

```

* topref: address of reconstructed top reference field
* botorg: address of original bottom reference field
* botref: address of reconstructed bottom reference field
* mb: macroblock to be matched
* i,j: location of mb (=center of search window)
* sx,sy: half width/height of search window
*
* iminp,jminp,selp,dfieldp: location and distance of best field prediction
* imin8up,jmin8up,sel8up: location of best 16x8 pred. for upper half of mb
* imin8lp,jmin8lp,sel8lp: location of best 16x8 pred. for lower half of mb
* d8p: distance of best 16x8 prediction
* iminsp,jminsp,dsp: location and distance of best same parity field
*
*       prediction (needed for dual prime, only valid if
*       ipflag==0)
*/
static void field_estimate(toporg,topref,botorg,botref,mb,i,j,sx,sy,ipflag,
    iminp,jminp,imin8up,jmin8up,imin8lp,jmin8lp,dfieldp,d8p,selp,sel8up,sel8lp,
    iminsp,jminsp,dsp)
unsigned char *toporg, *topref, *botorg, *botref, *mb;
int i,j,sx,sy;
int ipflag;
int *iminp, *jminp;
int *imin8up, *jmin8up, *imin8lp, *jmin8lp;
int *dfieldp, *d8p;
int *selp, *sel8up, *sel8lp;
int *iminsp, *jminsp, *dsp;
{
    int dt, db, imint, jmint, iminb, jminb, notop, nobot;

    /* if ipflag is set, predict from field of opposite parity only */
    notop = ipflag && (pict_struct==TOP_FIELD);
    nobot = ipflag && (pict_struct==BOTTOM_FIELD);

    /* field prediction */

    /* predict current field from top field */
    if (notop)
        dt = 65536; /* infinity */
    else
        dt = fullsearch(toporg,topref,mb,width<<1,
            i,j,sx,sy>>1,16,width,height>>1,
            &imint,&jmint);

    /* predict current field from bottom field */
    if (nobot)
        db = 65536; /* infinity */
    else
        db = fullsearch(botorg,botref,mb,width<<1,
            i,j,sx,sy>>1,16,width,height>>1,
            &iminb,&jminb);

    /* same parity prediction (only valid if ipflag==0) */
    if (pict_struct==TOP_FIELD)
    {
        *iminsp = imint; *jminsp = jmint; *dsp = dt;
    }
    else
    {
        *iminsp = iminb; *jminsp = jminb; *dsp = db;
    }
}

```

```

}

/* select field prediction */
if (dt<=db)
{
    *dfieldp = dt; *iminp = imint; *jminp = jmint; *selp = 0;
}
else
{
    *dfieldp = db; *iminp = iminb; *jminp = jminb; *selp = 1;
}

/* 16x8 motion compensation */

/* predict upper half field from top field */
if (notop)
    dt = 65536;
else
    dt = fullsearch(toporg,topref,mb,width<<1,
                    i,j,sx,sy>>1,8,width,height>>1,
                    &imint,&jmint);

/* predict upper half field from bottom field */
if (nobot)
    db = 65536;
else
    db = fullsearch(botorg,botref,mb,width<<1,
                    i,j,sx,sy>>1,8,width,height>>1,
                    &iminb,&jminb);

/* select prediction for upper half field */
if (dt<=db)
{
    *d8p = dt; *imin8up = imint; *jmin8up = jmint; *sel8up = 0;
}
else
{
    *d8p = db; *imin8up = iminb; *jmin8up = jminb; *sel8up = 1;
}

/* predict lower half field from top field */
if (notop)
    dt = 65536;
else
    dt = fullsearch(toporg,topref,mb+(width<<4),width<<1,
                    i,j+8,sx,sy>>1,8,width,height>>1,
                    &imint,&jmint);

/* predict lower half field from bottom field */
if (nobot)
    db = 65536;
else
    db = fullsearch(botorg,botref,mb+(width<<4),width<<1,
                    i,j+8,sx,sy>>1,8,width,height>>1,
                    &iminb,&jminb);

/* select prediction for lower half field */
if (dt<=db)

```



```

{
    *d8p += dt; *imin8lp = imint; *jmin8lp = jmint; *sel8lp = 0;
}
else
{
    *d8p += db; *imin8lp = iminb; *jmin8lp = jminb; *sel8lp = 1;
}
}

```

```

static void dpframe_estimate(ref,mb,i,j,iminf,jminf,
    iminp,jminp,imindmvp, jmindmvp, dmcp, vmcp)
unsigned char *ref, *mb;
int i,j;
int iminf[2][2], jminf[2][2];
int *iminp, *jminp;
int *imindmvp, *jmindmvp;
int *dmcp,*vmcp;
{
    int pref,ppred,delta_x,delta_y;
    int is,js,it,jt,ib,jb,it0,jt0,ib0,jb0;
    int imins,jmins,imint,jmint,iminb,jminb,imindmv,jmindmv;
    int vmc,local_dist;

```

```

/* Calculate Dual Prime distortions for 9 delta candidates
 * for each of the four minimum field vectors
 * Note: only for P pictures!
 */

```

```

/* initialize minimum dual prime distortion to large value */
vmc = 1 << 30;

```

```

for (pref=0; pref<2; pref++)
{
    for (ppred=0; ppred<2; ppred++)
    {
        /* convert Cartesian absolute to relative motion vector
         * values (wrt current macroblock address (i,j)
         */
        is = iminf[pref][ppred] - (i<<1);
        js = jminf[pref][ppred] - (j<<1);

        if (pref!=ppred)
        {
            /* vertical field shift adjustment */
            if (ppred==0)
                js++;
            else
                js--;

            /* mvxs and mvys scaling*/
            is<<=1;
            js<<=1;
            if (topfirst == ppred)
            {
                /* second field: scale by 1/3 */
                is = (is>=0) ? (is+1)/3 : -((-is+1)/3);
                js = (js>=0) ? (js+1)/3 : -((-js+1)/3);
            }
            else

```

```

    continue;
}

/* vector for prediction from field of opposite 'parity' */
if (topfirst)
{
    /* vector for prediction of top field from bottom field */
    it0 = ((is+(is>0))>>1);
    jt0 = ((js+(js>0))>>1) - 1;

    /* vector for prediction of bottom field from top field */
    ib0 = ((3*is+(is>0))>>1);
    jb0 = ((3*js+(js>0))>>1) + 1;
}
else
{
    /* vector for prediction of top field from bottom field */
    it0 = ((3*is+(is>0))>>1);
    jt0 = ((3*js+(js>0))>>1) - 1;

    /* vector for prediction of bottom field from top field */
    ib0 = ((is+(is>0))>>1);
    jb0 = ((js+(js>0))>>1) + 1;
}

/* convert back to absolute half-pel field picture coordinates */
is += i<<1;
js += j<<1;
it0 += i<<1;
jt0 += j<<1;
ib0 += i<<1;
jb0 += j<<1;

if (is >= 0 && is <= (width-16)<<1 &&
    js >= 0 && js <= (height-16))
{
    for (delta_y=-1; delta_y<=1; delta_y++)
    {
        for (delta_x=-1; delta_x<=1; delta_x++)
        {
            /* opposite field coordinates */
            it = it0 + delta_x;
            jt = jt0 + delta_y;
            ib = ib0 + delta_x;
            jb = jb0 + delta_y;

            if (it >= 0 && it <= (width-16)<<1 &&
                jt >= 0 && jt <= (height-16) &&
                ib >= 0 && ib <= (width-16)<<1 &&
                jb >= 0 && jb <= (height-16))
            {
                /* compute prediction error */
                local_dist = bdist2(
                    ref + (is>>1) + (width<<1)*(js>>1),
                    ref + width + (it>>1) + (width<<1)*(jt>>1),
                    mb, /* current mb location */
                    width<<1, /* adjacent line distance */
                    is&1, js&1, it&1, jt&1, /* half-pel flags */
                    8); /* block height */
            }
        }
    }
}

```

```

    local_dist += bdist2(
        ref + width + (is>>1) + (width<<1)*(js>>1),
        ref + (ib>>1) + (width<<1)*(jb>>1),
        mb + width, /* current mb location */
        width<<1, /* adjacent line distance */
        is&1, js&1, ib&1, jb&1, /* half-pel flags */
        8); /* block height */

    /* update delta with least distortion vector */
    if (local_dist < vmc)
    {
        imins = is;
        jmins = js;
        imint = it;
        jmint = jt;
        iminb = ib;
        jminb = jb;
        imindmv = delta_x;
        jmindmv = delta_y;
        vmc = local_dist;
    }
} /* end delta x loop */
} /* end delta y loop */
}
}

```

```

/* Compute L1 error for decision purposes */
local_dist = bdist1(
    ref + (imins>>1) + (width<<1)*(jmins>>1),
    ref + width + (imint>>1) + (width<<1)*(jmint>>1),
    mb,
    width<<1,
    imins&1, jmins&1, imint&1, jmint&1,
    8);
local_dist += bdist1(
    ref + width + (imins>>1) + (width<<1)*(jmins>>1),
    ref + (iminb>>1) + (width<<1)*(jminb>>1),
    mb + width,
    width<<1,
    imins&1, jmins&1, iminb&1, jminb&1,
    8);

*dmcp = local_dist;
*iminp = imins;
*jminp = jmins;
*imindmvp = imindmv;
*jmindmvp = jmindmv;
*vmcp = vmc;
}

```

```

static void dpfield_estimate(topref,botref,mb,i,j,imins,jmins,
    imindmvp, jmindmvp, dmcp, vmcp)
unsigned char *topref, *botref, *mb;
int i,j;
int imins, jmins;
int *imindmvp, *jmindmvp;
int *dmcp,*vmcp;

```

```

{
    unsigned char *sameref, *oppref;
    int io0,jo0,io,jo,delta_x,delta_y,mvxs,mvys,mvxo0,mvyo0;
    int imino,jmino,imindmv,jmindmv,vmc_dp,local_dist;

    /* Calculate Dual Prime distortions for 9 delta candidates */
    /* Note: only for P pictures! */

    /* Assign opposite and same reference pointer */
    if (pict_struct==TOP_FIELD)
    {
        sameref = topref;
        oppref = botref;
    }
    else
    {
        sameref = botref;
        oppref = topref;
    }

    /* convert Cartesian absolute to relative motion vector
     * values (wrt current macroblock address (i,j)
     */
    mvxs = imins - (i<<1);
    mvys = jmins - (j<<1);

    /* vector for prediction from field of opposite 'parity' */
    mvxo0 = (mvxs+(mvxs>0)) >> 1; /* mvxs // 2 */
    mvyo0 = (mvys+(mvys>0)) >> 1; /* mvys // 2 */

    /* vertical field shift correction */
    if (pict_struct==TOP_FIELD)
        mvyo0--;
    else
        mvyo0++;

    /* convert back to absolute coordinates */
    io0 = mvxo0 + (i<<1);
    jo0 = mvyo0 + (j<<1);

    /* initialize minimum dual prime distortion to large value */
    vmc_dp = 1 << 30;

    for (delta_y = -1; delta_y <= 1; delta_y++)
    {
        for (delta_x = -1; delta_x <= 1; delta_x++)
        {
            /* opposite field coordinates */
            io = io0 + delta_x;
            jo = jo0 + delta_y;

            if (io >= 0 && io <= (width-16)<<1 &&
                jo >= 0 && jo <= (height2-16)<<1)
            {
                /* compute prediction error */
                local_dist = bdist2(
                    sameref + (imins>>1) + width2*(jmins>>1),
                    oppref + (io>>1) + width2*(jo>>1),
                    mb, /* current mb location */

```

```

        width2,      /* adjacent line distance */
        imins&1, jmins&1, io&1, jo&1, /* half-pel flags */
        16);        /* block height */

/* update delta with least distortion vector */
if (local_dist < vmc_dp)
{
    imino = io;
    jmino = jo;
    imindmv = delta_x;
    jmindmv = delta_y;
    vmc_dp = local_dist;
}
}
} /* end delta x loop */
} /* end delta y loop */

/* Compute L1 error for decision purposes */
*dmcp = bdist1(
    sameref + (imins>>1) + width2*(jmins>>1),
    oppref + (imino>>1) + width2*(jmino>>1),
    mb,          /* current mb location */
    width2,      /* adjacent line distance */
    imins&1, jmins&1, imino&1, jmino&1, /* half-pel flags */
    16);        /* block height */

*imindmvp = imindmv;
*jmindmvp = jmindmv;
*vmcp = vmc_dp;
}

/*
 * full search block matching
 *
 * blk: top left pel of (16*h) block
 * h: height of block
 * lx: distance (in bytes) of vertically adjacent pels in ref,blk
 * org: top left pel of source reference picture
 * ref: top left pel of reconstructed reference picture
 * i0,j0: center of search window
 * sx,sy: half widths of search window
 * xmax,ymax: right/bottom limits of search area
 * iminp,jminp: pointers to where the result is stored
 *          result is given as half pel offset from ref(0,0)
 *          i.e. NOT relative to (i0,j0)
 */
static int fullsearch(org,ref,blk,lx,i0,j0,sx,sy,h,xmax,ymax,iminp,jminp)
unsigned char *org,*ref,*blk;
int lx,i0,j0,sx,sy,h,xmax,ymax;
int *iminp,*jminp;
{
    int i,j,imin,jmin,ilow,ihigh,jlow,jhigh;
    int d,dmin;
    int k,l,sxy;

    ilow = i0 - sx;
    ihigh = i0 + sx;

    if (ilow<0)

```

```

    ilow = 0;

    if (ihigh>xmax-16)
        ihigh = xmax-16;

    jlow = j0 - sy;
    jhigh = j0 + sy;

    if (jlow<0)
        jlow = 0;

    if (jhigh>ymax-h)
        jhigh = ymax-h;

    /* full pel search, spiraling outwards */

    imin = i0;
    jmin = j0;
    dmin = dist1(org+imin+lx*jmin,blk,lx,0,0,h,65536);

    sxy = (sx>sy) ? sx : sy;

    for (l=1; l<=sxy; l++)
    {
        i = i0 - l;
        j = j0 - l;
        for (k=0; k<8*l; k++)
        {
            if (i>=ilow && i<=ihigh && j>=jlow && j<=jhigh)
            {
                d = dist1(org+i+lx*j,blk,lx,0,0,h,dmin);

                if (d<dmin)
                {
                    dmin = d;
                    imin = i;
                    jmin = j;
                }
            }

            if (k<2*l) i++;
            else if (k<4*l) j++;
            else if (k<6*l) i--;
            else j--;
        }
    }

    /* half pel */
    dmin = 65536;
    imin <<= 1;
    jmin <<= 1;
    ilow = imin - (imin>0);
    ihigh = imin + (imin<((xmax-16)<<1));
    jlow = jmin - (jmin>0);
    jhigh = jmin + (jmin<((ymax-h)<<1));

    for (j=jlow; j<=jhigh; j++)
        for (i=ilow; i<=ihigh; i++)
            {

```

```

    d = dist1(ref+(i>>1)+lx*(j>>1),blk,lx,i&1,j&1,h,dmin);

    if (d<dmin)
    {
        dmin = d;
        imin = i;
        jmin = j;
    }
}

*iminp = imin;
*jminp = jmin;

return dmin;
}

/*
 * total absolute difference between two (16*h) blocks
 * including optional half pel interpolation of blk1 (hx,hy)
 * blk1,blk2: addresses of top left pels of both blocks
 * lx:      distance (in bytes) of vertically adjacent pels
 * hx,hy:   flags for horizontal and/or vertical interpolation
 * h:      height of block (usually 8 or 16)
 * distlim: bail out if sum exceeds this value
 */
static int dist1(blk1,blk2,lx,hx,hy,h,distlim)
unsigned char *blk1,*blk2;
int lx,hx,hy,h;
int distlim;
{
    unsigned char *p1,*p1a,*p2;
    int i,j;
    int s,v;

    s = 0;
    p1 = blk1;
    p2 = blk2;

    if (!hx && !hy)
        for (j=0; j<h; j++)
        {
            if ((v = p1[0] - p2[0])<0) v = -v; s+= v;
            if ((v = p1[1] - p2[1])<0) v = -v; s+= v;
            if ((v = p1[2] - p2[2])<0) v = -v; s+= v;
            if ((v = p1[3] - p2[3])<0) v = -v; s+= v;
            if ((v = p1[4] - p2[4])<0) v = -v; s+= v;
            if ((v = p1[5] - p2[5])<0) v = -v; s+= v;
            if ((v = p1[6] - p2[6])<0) v = -v; s+= v;
            if ((v = p1[7] - p2[7])<0) v = -v; s+= v;
            if ((v = p1[8] - p2[8])<0) v = -v; s+= v;
            if ((v = p1[9] - p2[9])<0) v = -v; s+= v;
            if ((v = p1[10] - p2[10])<0) v = -v; s+= v;
            if ((v = p1[11] - p2[11])<0) v = -v; s+= v;
            if ((v = p1[12] - p2[12])<0) v = -v; s+= v;
            if ((v = p1[13] - p2[13])<0) v = -v; s+= v;
            if ((v = p1[14] - p2[14])<0) v = -v; s+= v;
            if ((v = p1[15] - p2[15])<0) v = -v; s+= v;

            if (s >= distlim)

```

```

        break;

        p1+= lx;
        p2+= lx;
    }
    else if (hx && !hy)
        for (j=0; j<h; j++)
        {
            for (i=0; i<16; i++)
            {
                v = ((unsigned int)(p1[i]+p1[i+1]+1)>>1) - p2[i];
                if (v>=0)
                    s+= v;
                else
                    s-= v;
            }
            p1+= lx;
            p2+= lx;
        }
    else if (!hx && hy)
    {
        p1a = p1 + lx;
        for (j=0; j<h; j++)
        {
            for (i=0; i<16; i++)
            {
                v = ((unsigned int)(p1[i]+p1a[i]+1)>>1) - p2[i];
                if (v>=0)
                    s+= v;
                else
                    s-= v;
            }
            p1 = p1a;
            p1a+= lx;
            p2+= lx;
        }
    }
    else /* if (hx && hy) */
    {
        p1a = p1 + lx;
        for (j=0; j<h; j++)
        {
            for (i=0; i<16; i++)
            {
                v = ((unsigned int)(p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2)>>2) - p2[i];
                if (v>=0)
                    s+= v;
                else
                    s-= v;
            }
            p1 = p1a;
            p1a+= lx;
            p2+= lx;
        }
    }

    return s;
}

```



```

/*
 * total squared difference between two (16*h) blocks
 * including optional half pel interpolation of blk1 (hx,hy)
 * blk1,blk2: addresses of top left pels of both blocks
 * lx:      distance (in bytes) of vertically adjacent pels
 * hx,hy:   flags for horizontal and/or vertical interpolation
 * h:      height of block (usually 8 or 16)
 */
static int dist2(blk1,blk2,lx,hx,hy,h)
unsigned char *blk1,*blk2;
int lx,hx,hy,h;
{
    unsigned char *p1,*p1a,*p2;
    int i,j;
    int s,v;

    s = 0;
    p1 = blk1;
    p2 = blk2;
    if (!hx && !hy)
        for (j=0; j<h; j++)
        {
            for (i=0; i<16; i++)
            {
                v = p1[i] - p2[i];
                s+= v*v;
            }
            p1+= lx;
            p2+= lx;
        }
    else if (hx && !hy)
        for (j=0; j<h; j++)
        {
            for (i=0; i<16; i++)
            {
                v = ((unsigned int)(p1[i]+p1[i+1]+1)>>1) - p2[i];
                s+= v*v;
            }
            p1+= lx;
            p2+= lx;
        }
    else if (!hx && hy)
    {
        p1a = p1 + lx;
        for (j=0; j<h; j++)
        {
            for (i=0; i<16; i++)
            {
                v = ((unsigned int)(p1[i]+p1a[i]+1)>>1) - p2[i];
                s+= v*v;
            }
            p1 = p1a;
            p1a+= lx;
            p2+= lx;
        }
    }
    else /* if (hx && hy) */
    {
        p1a = p1 + lx;

```

```

    for (j=0; j<h; j++)
    {
        for (i=0; i<16; i++)
        {
            v = ((unsigned int)(p1[i]+p1[i+1]+p1a[i]+p1a[i+1]+2)>>2) - p2[i];
            s+= v*v;
        }
        p1 = p1a;
        p1a+= lx;
        p2+= lx;
    }
}

return s;
}

/*
 * absolute difference error between a (16*h) block and a bidirectional
 * prediction
 */
/*
 * p2: address of top left pel of block
 * pf,hxf,hyf: address and half pel flags of forward ref. block
 * pb,hxb,hyb: address and half pel flags of backward ref. block
 * h: height of block
 * lx: distance (in bytes) of vertically adjacent pels in p2,pf,pb
 */
static int bdist1(pf,pb,p2,lx,hxf,hyf,hxb,hyb,h)
unsigned char *pf,*pb,*p2;
int lx,hxf,hyf,hxb,hyb,h;
{
    unsigned char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
    int i,j;
    int s,v;

    pfa = pf + hxf;
    pfb = pf + lx*hyf;
    pfc = pfb + hxf;

    pba = pb + hxb;
    pbb = pb + lx*hyb;
    pbc = pbb + hxb;

    s = 0;

    for (j=0; j<h; j++)
    {
        for (i=0; i<16; i++)
        {
            v = (((unsigned int)(*pf++ + *pfa++ + *pfb++ + *pfc++ + 2)>>2) +
                ((unsigned int)(*pb++ + *pba++ + *pbb++ + *pbc++ + 2)>>2) + 1)>>1)
                - *p2++;
            if (v>=0)
                s+= v;
            else
                s-= v;
        }
        p2+= lx-16;
        pf+= lx-16;
        pfa+= lx-16;

```

```

    pfb+= lx-16;
    pfc+= lx-16;
    pb+= lx-16;
    pba+= lx-16;
    pbb+= lx-16;
    pbc+= lx-16;
}

return s;
}

/*
 * squared error between a (16*h) block and a bidirectional
 * prediction
 *
 * p2: address of top left pel of block
 * pf,hxf,hyf: address and half pel flags of forward ref. block
 * pb,hxb,hyb: address and half pel flags of backward ref. block
 * h: height of block
 * lx: distance (in bytes) of vertically adjacent pels in p2,pf,pb
 */
static int bdist2(pf,pb,p2,lx,hxf,hyf,hxb,hyb,h)
unsigned char *pf,*pb,*p2;
int lx,hxf,hyf,hxb,hyb,h;
{
    unsigned char *pfa,*pfb,*pfc,*pba,*pbb,*pbc;
    int i,j;
    int s,v;

    pfa = pf + hxf;
    pfb = pf + lx*hyf;
    pfc = pfb + hxf;

    pba = pb + hxb;
    pbb = pb + lx*hyb;
    pbc = pbb + hxb;

    s = 0;

    for (j=0; j<h; j++)
    {
        for (i=0; i<16; i++)
        {
            v = (((unsigned int)(*pf++ + *pfa++ + *pfb++ + *pfc++ + 2)>>2) +
                ((unsigned int)(*pb++ + *pba++ + *pbb++ + *pbc++ + 2)>>2) + 1)>>1)
                - *p2++;
            s+=v*v;
        }
        p2+= lx-16;
        pf+= lx-16;
        pfa+= lx-16;
        pfb+= lx-16;
        pfc+= lx-16;
        pb+= lx-16;
        pba+= lx-16;
        pbb+= lx-16;
        pbc+= lx-16;
    }
}

```

```

    return s;
}

/*
 * variance of a (16*16) block, multiplied by 256
 * p: address of top left pel of block
 * lx: distance (in bytes) of vertically adjacent pels
 */
static int variance(p,lx)
unsigned char *p;
int lx;
{
    int i,j;
    unsigned int v,s,s2;

    s = s2 = 0;

    for (j=0; j<16; j++)
    {
        for (i=0; i<16; i++)
        {
            v = *p++;
            s += v;
            s2 += v*v;
        }
        p += lx-16;
    }
    return s2 - (s*s)/256;
}

```

mpeg2enc.c

```

/* mpeg2enc.c, main() and parameter file reading */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
*
*/

#include <stdio.h>
#include <stdlib.h>

#define GLOBAL /* used by global.h */
#include "config.h"
#include "global.h"

/* private prototypes */
static void init _ANSI_ARGS__((void));
static void readparmfile _ANSI_ARGS__((char *fname));
static void readquantmat _ANSI_ARGS__((void));

int main(argc,argv)
int argc;
char *argv[];
{
    if (argc!=3)
    {
        printf("\n%s, %s\n",version,author);
        printf("Usage: mpeg2encode in.par out.m2v\n");
        exit(0);
    }

    /* read parameter file */
    readparmfile(argv[1]);

    /* read quantization matrices */
    readquantmat();

    /* open output file */
    if (!(outfile=fopen(argv[2],"wb")))
    {
        sprintf(errortext,"Couldn't create output file %s",argv[2]);
        error(errortext);
    }

    init();
    putseq();

    fclose(outfile);
    fclose(statfile);

    return 0;
}

static void init()
{
    int i, size;
    static int block_count_tab[3] = {6,8,12};

    initbits();
    init_fdct();
    init_idct();

```

```

/* round picture dimensions to nearest multiple of 16 or 32 */
mb_width = (horizontal_size+15)/16;
mb_height = prog_seq ? (vertical_size+15)/16 : 2*((vertical_size+31)/32);
mb_height2 = fieldpic ? mb_height>>1 : mb_height; /* for field pictures */
width = 16*mb_width;
height = 16*mb_height;

chrom_width = (chroma_format==CHROMA444) ? width : width>>1;
chrom_height = (chroma_format!=CHROMA420) ? height : height>>1;

height2 = fieldpic ? height>>1 : height;
width2 = fieldpic ? width<<1 : width;
chrom_width2 = fieldpic ? chrom_width<<1 : chrom_width;

block_count = block_count_tab[chroma_format-1];

/* clip table */
if (!(clp = (unsigned char *)malloc(1024)))
    error("malloc failed\n");
clp+= 384;
for (i=-384; i<640; i++)
    clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

for (i=0; i<3; i++)
{
    size = (i==0) ? width*height : chrom_width*chrom_height;

    if (!(newrefframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
    if (!(oldrefframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
    if (!(auxframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
    if (!(neworgframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
    if (!(oldorgframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
    if (!(auxorgframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
    if (!(predframe[i] = (unsigned char *)malloc(size)))
        error("malloc failed\n");
}

mbinfo = (struct mbinfo *)malloc(mb_width*mb_height2*sizeof(struct mbinfo));

if (!mbinfo)
    error("malloc failed\n");

blocks =
    (short (*)(64))malloc(mb_width*mb_height2*block_count*sizeof(short [64]));

if (!blocks)
    error("malloc failed\n");

/* open statistics output file */
if (statname[0]!='-')
    statfile = stdout;
else if (!(statfile = fopen(statname,"w")))
{

```

```

    sprintf(errortext,"Couldn't create statistics output file %s",statname);
    error(errortext);
}
}

void error(text)
char *text;
{
    fprintf(stderr,text);
    putc('\n',stderr);
    exit(1);
}

static void readparmfile(fname)
char *fname;
{
    int i;
    int h,m,s,f;
    FILE *fd;
    char line[256];
    static double ratetab[8]=
        { 24000.0/1001.0,24.0,25.0,30000.0/1001.0,30.0,50.0,60000.0/1001.0,60.0};
    extern int r,Xi,Xb,Xp,d0i,d0p,d0b; /* rate control */
    extern double avg_act; /* rate control */

    if (!(fd = fopen(fname,"r")))
    {
        sprintf(errortext,"Couldn't open parameter file %s",fname);
        error(errortext);
    }

    fgets(id_string,254,fd);
    fgets(line,254,fd); sscanf(line,"%s",tplorg);
    fgets(line,254,fd); sscanf(line,"%s",tplref);
    fgets(line,254,fd); sscanf(line,"%s",iqname);
    fgets(line,254,fd); sscanf(line,"%s",niqname);
    fgets(line,254,fd); sscanf(line,"%s",statname);
    fgets(line,254,fd); sscanf(line,"%d",&inputtype);
    fgets(line,254,fd); sscanf(line,"%d",&nframes);
    fgets(line,254,fd); sscanf(line,"%d",&frame0);
    fgets(line,254,fd); sscanf(line,"%d:%d:%d:%d",&h,&m,&s,&f);
    fgets(line,254,fd); sscanf(line,"%d",&N);
    fgets(line,254,fd); sscanf(line,"%d",&M);
    fgets(line,254,fd); sscanf(line,"%d",&mpeg1);
    fgets(line,254,fd); sscanf(line,"%d",&fieldpic);
    fgets(line,254,fd); sscanf(line,"%d",&horizontal_size);
    fgets(line,254,fd); sscanf(line,"%d",&vertical_size);
    fgets(line,254,fd); sscanf(line,"%d",&aspectratio);
    fgets(line,254,fd); sscanf(line,"%d",&frame_rate_code);
    fgets(line,254,fd); sscanf(line,"%lf",&bit_rate);
    fgets(line,254,fd); sscanf(line,"%d",&vbv_buffer_size);
    fgets(line,254,fd); sscanf(line,"%d",&low_delay);
    fgets(line,254,fd); sscanf(line,"%d",&constrparms);
    fgets(line,254,fd); sscanf(line,"%d",&profile);
    fgets(line,254,fd); sscanf(line,"%d",&level);
    fgets(line,254,fd); sscanf(line,"%d",&prog_seq);
    fgets(line,254,fd); sscanf(line,"%d",&chroma_format);
    fgets(line,254,fd); sscanf(line,"%d",&video_format);
    fgets(line,254,fd); sscanf(line,"%d",&color primaries);

```

```

fgets(line,254,fd); sscanf(line,"%d",&transfer_characteristics);
fgets(line,254,fd); sscanf(line,"%d",&matrix_coefficients);
fgets(line,254,fd); sscanf(line,"%d",&display_horizontal_size);
fgets(line,254,fd); sscanf(line,"%d",&display_vertical_size);
fgets(line,254,fd); sscanf(line,"%d",&dc_prec);
fgets(line,254,fd); sscanf(line,"%d",&topfirst);
fgets(line,254,fd); sscanf(line,"%d %d %d",
    frame_pred_dct_tab,frame_pred_dct_tab+1,frame_pred_dct_tab+2);

fgets(line,254,fd); sscanf(line,"%d %d %d",
    conceal_tab,conceal_tab+1,conceal_tab+2);

fgets(line,254,fd); sscanf(line,"%d %d %d",
    qscale_tab,qscale_tab+1,qscale_tab+2);

fgets(line,254,fd); sscanf(line,"%d %d %d",
    intravlc_tab,intravlc_tab+1,intravlc_tab+2);
fgets(line,254,fd); sscanf(line,"%d %d %d",
    altscan_tab,altscan_tab+1,altscan_tab+2);
fgets(line,254,fd); sscanf(line,"%d",&repeatfirst);
fgets(line,254,fd); sscanf(line,"%d",&prog_frame);
/* intra slice interval refresh period */
fgets(line,254,fd); sscanf(line,"%d",&P);
fgets(line,254,fd); sscanf(line,"%d",&r);
fgets(line,254,fd); sscanf(line,"%lf",&avg_act);
fgets(line,254,fd); sscanf(line,"%d",&Xi);
fgets(line,254,fd); sscanf(line,"%d",&Xp);
fgets(line,254,fd); sscanf(line,"%d",&Xb);
fgets(line,254,fd); sscanf(line,"%d",&d0i);
fgets(line,254,fd); sscanf(line,"%d",&d0p);
fgets(line,254,fd); sscanf(line,"%d",&d0b);

if (N<1)
    error("N must be positive");
if (M<1)
    error("M must be positive");
if (N%M != 0)
    error("N must be an integer multiple of M");

motion_data = (struct motion_data *)malloc(M*sizeof(struct motion_data));
if (!motion_data)
    error("malloc failed\n");

for (i=0; i<M; i++)
{
    fgets(line,254,fd);
    sscanf(line,"%d %d %d %d",
        &motion_data[i].forw_hor_f_code, &motion_data[i].forw_vert_f_code,
        &motion_data[i].sxf, &motion_data[i].syf);

    if (i!=0)
    {
        fgets(line,254,fd);
        sscanf(line,"%d %d %d %d",
            &motion_data[i].back_hor_f_code, &motion_data[i].back_vert_f_code,
            &motion_data[i].sxb, &motion_data[i].syb);
    }
}

```



```

fclose(fd);

/* make flags boolean (x!=0 -> x=1) */
mpeg1 = !!mpeg1;
fieldpic = !!fieldpic;
low_delay = !!low_delay;
constrparms = !!constrparms;
prog_seq = !!prog_seq;
topfirst = !!topfirst;

for (i=0; i<3; i++)
{
    frame_pred_dct_tab[i] = !!frame_pred_dct_tab[i];
    conceal_tab[i] = !!conceal_tab[i];
    qscale_tab[i] = !!qscale_tab[i];
    intravlc_tab[i] = !!intravlc_tab[i];
    altscan_tab[i] = !!altscan_tab[i];
}
repeatfirst = !!repeatfirst;
prog_frame = !!prog_frame;

/* make sure MPEG specific parameters are valid */
range_checks();

frame_rate = ratetab[frame_rate_code-1];

/* timecode -> frame number */
tc0 = h;
tc0 = 60*tc0 + m;
tc0 = 60*tc0 + s;
tc0 = (int)(frame_rate+0.5)*tc0 + f;

if (!mpeg1)
{
    profile_and_level_checks();
}
else
{
    /* MPEG-1 */
    if (constrparms)
    {
        if (horizontal_size>768
            || vertical_size>576
            || ((horizontal_size+15)/16)*((vertical_size+15)/16)>396
            || ((horizontal_size+15)/16)*((vertical_size+15)/16)*frame_rate>396*25.0
            || frame_rate>30.0)
        {
            if (!quiet)
                fprintf(stderr, "Warning: setting constrained_parameters_flag = 0\n");
            constrparms = 0;
        }
    }

    if (constrparms)
    {
        for (i=0; i<M; i++)
        {
            if (motion_data[i].forw_hor_f_code>4)
            {

```

```

    if (!quiet)
        fprintf(stderr, "Warning: setting constrained_parameters_flag = 0\n");
    constrparms = 0;
    break;
}

if (motion_data[i].forw_vert_f_code > 4)
{
    if (!quiet)
        fprintf(stderr, "Warning: setting constrained_parameters_flag = 0\n");
    constrparms = 0;
    break;
}

if (i != 0)
{
    if (motion_data[i].back_hor_f_code > 4)
    {
        if (!quiet)
            fprintf(stderr, "Warning: setting constrained_parameters_flag = 0\n");
        constrparms = 0;
        break;
    }

    if (motion_data[i].back_vert_f_code > 4)
    {
        if (!quiet)
            fprintf(stderr, "Warning: setting constrained_parameters_flag = 0\n");
        constrparms = 0;
        break;
    }
}
}
}

/* relational checks */

if (mpeg1)
{
    if (!prog_seq)
    {
        if (!quiet)
            fprintf(stderr, "Warning: setting progressive_sequence = 1\n");
        prog_seq = 1;
    }

    if (chroma_format != CHROMA420)
    {
        if (!quiet)
            fprintf(stderr, "Warning: setting chroma_format = 1 (4:2:0)\n");
        chroma_format = CHROMA420;
    }

    if (dc_prec != 0)
    {
        if (!quiet)
            fprintf(stderr, "Warning: setting intra_dc_precision = 0\n");
        dc_prec = 0;
    }
}

```

```

    }

    for (i=0; i<3; i++)
        if (qscale_tab[i])
        {
            if (!quiet)
                fprintf(stderr,"Warning: setting qscale_tab[%d] = 0\n",i);
            qscale_tab[i] = 0;
        }

    for (i=0; i<3; i++)
        if (intravlc_tab[i])
        {
            if (!quiet)
                fprintf(stderr,"Warning: setting intravlc_tab[%d] = 0\n",i);
            intravlc_tab[i] = 0;
        }

    for (i=0; i<3; i++)
        if (altscan_tab[i])
        {
            if (!quiet)
                fprintf(stderr,"Warning: setting altscan_tab[%d] = 0\n",i);
            altscan_tab[i] = 0;
        }
    }

    if (!mpeg1 && constrparms)
    {
        if (!quiet)
            fprintf(stderr,"Warning: setting constrained_parameters_flag = 0\n");
        constrparms = 0;
    }

    if (prog_seq && !prog_frame)
    {
        if (!quiet)
            fprintf(stderr,"Warning: setting progressive_frame = 1\n");
        prog_frame = 1;
    }

    if (prog_frame && fieldpic)
    {
        if (!quiet)
            fprintf(stderr,"Warning: setting field_pictures = 0\n");
        fieldpic = 0;
    }

    if (!prog_frame && repeatfirst)
    {
        if (!quiet)
            fprintf(stderr,"Warning: setting repeat_first_field = 0\n");
        repeatfirst = 0;
    }

    if (prog_frame)
    {
        for (i=0; i<3; i++)
            if (!frame_pred_dct_tab[i])

```

```

    {
        if (!quiet)
            fprintf(stderr, "Warning: setting frame_pred_frame_dct[%d] = 1\n", i);
        frame_pred_dct_tab[i] = 1;
    }
}

if (prog_seq && !repeatfirst && topfirst)
{
    if (!quiet)
        fprintf(stderr, "Warning: setting top_field_first = 0\n");
    topfirst = 0;
}

/* search windows */
for (i=0; i<M; i++)
{
    if (motion_data[i].sxf > (4<<motion_data[i].forw_hor_f_code)-1)
    {
        if (!quiet)
            fprintf(stderr,
                "Warning: reducing forward horizontal search width to %d\n",
                (4<<motion_data[i].forw_hor_f_code)-1);
        motion_data[i].sxf = (4<<motion_data[i].forw_hor_f_code)-1;
    }

    if (motion_data[i].syf > (4<<motion_data[i].forw_vert_f_code)-1)
    {
        if (!quiet)
            fprintf(stderr,
                "Warning: reducing forward vertical search width to %d\n",
                (4<<motion_data[i].forw_vert_f_code)-1);
        motion_data[i].syf = (4<<motion_data[i].forw_vert_f_code)-1;
    }

    if (i!=0)
    {
        if (motion_data[i].sxb > (4<<motion_data[i].back_hor_f_code)-1)
        {
            if (!quiet)
                fprintf(stderr,
                    "Warning: reducing backward horizontal search width to %d\n",
                    (4<<motion_data[i].back_hor_f_code)-1);
            motion_data[i].sxb = (4<<motion_data[i].back_hor_f_code)-1;
        }

        if (motion_data[i].syb > (4<<motion_data[i].back_vert_f_code)-1)
        {
            if (!quiet)
                fprintf(stderr,
                    "Warning: reducing backward vertical search width to %d\n",
                    (4<<motion_data[i].back_vert_f_code)-1);
            motion_data[i].syb = (4<<motion_data[i].back_vert_f_code)-1;
        }
    }
}
}

```

```

static void readquantmat()
{
    int i,v;
    FILE *fd;

    if (iqname[0]!='-')
    {
        /* use default intra matrix */
        load_iquant = 0;
        for (i=0; i<64; i++)
            intra_q[i] = default_intra_quantizer_matrix[i];
    }
    else
    {
        /* read customized intra matrix */
        load_iquant = 1;
        if (!(fd = fopen(iqname,"r")))
        {
            sprintf(errortext,"Couldn't open quant matrix file %s",iqname);
            error(errortext);
        }

        for (i=0; i<64; i++)
        {
            fscanf(fd,"%d",&v);
            if (v<1 || v>255)
                error("invalid value in quant matrix");
            intra_q[i] = v;
        }

        fclose(fd);
    }

    if (niqname[0]!='-')
    {
        /* use default non-intra matrix */
        load_niquant = 0;
        for (i=0; i<64; i++)
            inter_q[i] = 16;
    }
    else
    {
        /* read customized non-intra matrix */
        load_niquant = 1;
        if (!(fd = fopen(niqname,"r")))
        {
            sprintf(errortext,"Couldn't open quant matrix file %s",niqname);
            error(errortext);
        }

        for (i=0; i<64; i++)
        {
            fscanf(fd,"%d",&v);
            if (v<1 || v>255)
                error("invalid value in quant matrix");
            inter_q[i] = v;
        }

        fclose(fd);
    }

```

```

}
}

```

mpeg2enc.h

```

/* mpeg2enc.h, defines and types */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#define PICTURE_START_CODE 0x100L
#define SLICE_MIN_START 0x101L
#define SLICE_MAX_START 0x1AFL
#define USER_START_CODE 0x1B2L
#define SEQ_START_CODE 0x1B3L
#define EXT_START_CODE 0x1B5L
#define SEQ_END_CODE 0x1B7L
#define GOP_START_CODE 0x1B8L
#define ISO_END_CODE 0x1B9L
#define PACK_START_CODE 0x1BAL
#define SYSTEM_START_CODE 0x1BBL

/* picture coding type */
#define I_TYPE 1
#define P_TYPE 2
#define B_TYPE 3
#define D_TYPE 4

/* picture structure */
#define TOP_FIELD 1
#define BOTTOM_FIELD 2
#define FRAME_PICTURE 3

```

```

/* macroblock type */
#define MB_INTRA 1
#define MB_PATTERN 2
#define MB_BACKWARD 4
#define MB_FORWARD 8
#define MB_QUANT 16

/* motion_type */
#define MC_FIELD 1
#define MC_FRAME 2
#define MC_16X8 2
#define MC_DMV 3

/* mv_format */
#define MV_FIELD 0
#define MV_FRAME 1

/* chroma_format */
#define CHROMA420 1
#define CHROMA422 2
#define CHROMA444 3

/* extension start code IDs */

#define SEQ_ID 1
#define DISP_ID 2
#define QUANT_ID 3
#define SEQSCAL_ID 5
#define PANSCAN_ID 7
#define CODING_ID 8
#define SPATSCAL_ID 9
#define TEMPSCAL_ID 10

/* inputtype */
#define T_Y_U_V 0
#define T_YUV 1
#define T_PPM 2

/* macroblock information */
struct mbinfo {
    int mb_type; /* intra/forward/backward/interpolated */
    int motion_type; /* frame/field/16x8/dual_prime */
    int dct_type; /* field/frame DCT */
    int mquant; /* quantization parameter */
    int cbp; /* coded block pattern */
    int skipped; /* skipped macroblock */
    int MV[2][2][2]; /* motion vectors */
    int mv_field_sel[2][2]; /* motion vertical field select */
    int dmvector[2]; /* dual prime vectors */
    double act; /* activity measure */
    int var; /* for debugging */
};

/* motion data */
struct motion_data {
    int forw_hor_f_code, forw_vert_f_code; /* vector range */
    int sx, sy; /* search range */
    int back_hor_f_code, back_vert_f_code;
    int sx, sy;

```

```
};
```

```
predict.c
```

```
/* predict.c, motion compensated prediction
```

```
*/
```

```
/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
```

```
 *
```

```
 * The MPEG Software Simulation Group 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.
```

```
 *
```

```
*/
```

```
#include <stdio.h>
```

```
#include "config.h"
```

```
#include "global.h"
```

```
/* private prototypes */
```

```
static void predict_mb _ANSI_ARGS__((
    unsigned char *oldref[], unsigned char *newref[], unsigned char *cur[],
    int lx, int bx, int by, int pict_type, int pict_struct, int mb_type,
    int motion_type, int secondfield,
    int PMV[2][2][2], int mv_field_sel[2][2], int dmvector[2]));
```

```
static void pred _ANSI_ARGS__((unsigned char *src[], int sfield,
    unsigned char *dst[], int dfield,
    int lx, int w, int h, int x, int y, int dx, int dy, int addflag));
```

```
static void pred_comp _ANSI_ARGS__((unsigned char *src, unsigned char *dst,
    int lx, int w, int h, int x, int y, int dx, int dy, int addflag));
```

```
static void calc_DMV _ANSI_ARGS__((int DMV[][2], int *dmvector, int mvx,
    int mvy));
```

```
static void clearblock _ANSI_ARGS__((unsigned char *cur[], int i0, int j0));
```

```
/* form prediction for a complete picture (frontend for predict_mb)
```



```

*
* reff: reference frame for forward prediction
* refb: reference frame for backward prediction
* cur: destination (current) frame
* secondfield: predict second field of a frame
* mbi: macroblock info
*
* Notes:
* - cf. predict_mb
*/

void predict(reff,refb,cur,secondfield,mbi)
unsigned char *reff[],*refb[],*cur[3];
int secondfield;
struct mbinfo *mbi;
{
    int i, j, k;

    k = 0;

    /* loop through all macroblocks of the picture */
    for (j=0; j<height2; j+=16)
        for (i=0; i<width; i+=16)
        {
            predict_mb(reff,refb,cur,width,i,j,pict_type,pict_struct,
                      mbi[k].mb_type,mbi[k].motion_type,secondfield,
                      mbi[k].MV,mbi[k].mv_field_sel,mbi[k].dmvector);

            k++;
        }
}

/* form prediction for one macroblock
*
* oldref: reference frame for forward prediction
* newref: reference frame for backward prediction
* cur: destination (current) frame
* lx: frame width (identical to global var `width')
* bx,by: picture (field or frame) coordinates of macroblock to be predicted
* pict_type: I, P or B
* pict_struct: FRAME_PICTURE, TOP_FIELD, BOTTOM_FIELD
* mb_type: MB_FORWARD, MB_BACKWARD, MB_INTRA
* motion_type: MC_FRAME, MC_FIELD, MC_16X8, MC_DMV
* secondfield: predict second field of a frame
* PMV[2][2][2]: motion vectors (in half pel picture coordinates)
* mv_field_sel[2][2]: motion vertical field selects (for field predictions)
* dmvector: differential motion vectors (for dual prime)
*
* Notes:
* - when predicting a P type picture which is the second field of
* a frame, the same parity reference field is in oldref, while the
* opposite parity reference field is assumed to be in newref!
* - intra macroblocks are modelled to have a constant prediction of 128
* for all pels; this results in a DC DCT coefficient symmetric to 0
* - vectors for field prediction in frame pictures are in half pel frame
* coordinates (vertical component is twice the field value and always
* even)
*
* already covers dual prime (not yet used)

```

```

*/

static void predict_mb(oldref,newref,cur,lx,bx,by,pict_type,pict_struct,
    mb_type,motion_type,secondfield,PMV,mv_field_sel,dmvector)
unsigned char *oldref[],*newref[],*cur[];
int lx;
int bx,by;
int pict_type;
int pict_struct;
int mb_type;
int motion_type;
int secondfield;
int PMV[2][2][2], mv_field_sel[2][2], dmvector[2];
{
    int addflag, currentfield;
    unsigned char **predframe;
    int DMV[2][2];

    if (mb_type&MB_INTRA)
    {
        clearblock(cur,bx,by);
        return;
    }

    addflag = 0; /* first prediction is stored, second is added and averaged */

    if ((mb_type & MB_FORWARD) || (pict_type==P_TYPE))
    {
        /* forward prediction, including zero MV in P pictures */

        if (pict_struct==FRAME_PICTURE)
        {
            /* frame picture */

            if ((motion_type==MC_FRAME) || !(mb_type & MB_FORWARD))
            {
                /* frame-based prediction in frame picture */
                pred(oldref,0,cur,0,
                    lx,16,16,bx,by,PMV[0][0][0],PMV[0][0][1],0);
            }
            else if (motion_type==MC_FIELD)
            {
                /* field-based prediction in frame picture
                 *
                 * note scaling of the vertical coordinates (by, PMV[][0][1])
                 * from frame to field!
                 */

                /* top field prediction */
                pred(oldref,mv_field_sel[0][0],cur,0,
                    lx<<1,16,8,bx,by>>1,PMV[0][0][0],PMV[0][0][1]>>1,0);

                /* bottom field prediction */
                pred(oldref,mv_field_sel[1][0],cur,1,
                    lx<<1,16,8,bx,by>>1,PMV[1][0][0],PMV[1][0][1]>>1,0);
            }
            else if (motion_type==MC_DMV)
            {
                /* dual prime prediction */

```

```

/* calculate derived motion vectors */
calc_DMV(DMV,dmvector,PMV[0][0][0],PMV[0][0][1]>>1);

/* predict top field from top field */
pred(oldref,0,cur,0,
  lx<<1,16,8,bx,by>>1,PMV[0][0][0],PMV[0][0][1]>>1,0);

/* predict bottom field from bottom field */
pred(oldref,1,cur,1,
  lx<<1,16,8,bx,by>>1,PMV[0][0][0],PMV[0][0][1]>>1,0);

/* predict and add to top field from bottom field */
pred(oldref,1,cur,0,
  lx<<1,16,8,bx,by>>1,DMV[0][0],DMV[0][1],1);

/* predict and add to bottom field from top field */
pred(oldref,0,cur,1,
  lx<<1,16,8,bx,by>>1,DMV[1][0],DMV[1][1],1);
}
else
{
/* invalid motion_type in frame picture */
if (!quiet)
  fprintf(stderr,"invalid motion_type\n");
}
}
else /* TOP_FIELD or BOTTOM_FIELD */
{
/* field picture */

currentfield = (pict_struct==BOTTOM_FIELD);

/* determine which frame to use for prediction */
if ((pict_type==P_TYPE) && secondfield
  && (currentfield!=mv_field_sel[0][0]))
  predframe = newref; /* same frame */
else
  predframe = oldref; /* previous frame */

if ((motion_type==MC_FIELD) || !(mb_type & MB_FORWARD))
{
/* field-based prediction in field picture */
pred(predframe,mv_field_sel[0][0],cur,currentfield,
  lx<<1,16,16,bx,by,PMV[0][0][0],PMV[0][0][1],0);
}
else if (motion_type==MC_16X8)
{
/* 16 x 8 motion compensation in field picture */

/* upper half */
pred(predframe,mv_field_sel[0][0],cur,currentfield,
  lx<<1,16,8,bx,by,PMV[0][0][0],PMV[0][0][1],0);

/* determine which frame to use for lower half prediction */
if ((pict_type==P_TYPE) && secondfield
  && (currentfield!=mv_field_sel[1][0]))
  predframe = newref; /* same frame */
else

```

```

    predframe = oldref; /* previous frame */

    /* lower half */
    pred(predframe,mv_field_sel[1][0],cur,currentfield,
        lx<<1,16,8,bx,by+8,PMV[1][0][0],PMV[1][0][1],0);
}
else if (motion_type==MC_DMV)
{
    /* dual prime prediction */

    /* determine which frame to use for prediction */
    if (secondfield)
        predframe = newref; /* same frame */
    else
        predframe = oldref; /* previous frame */

    /* calculate derived motion vectors */
    calc_DMV(DMV,dmvector,PMV[0][0][0],PMV[0][0][1]);

    /* predict from field of same parity */
    pred(oldref,currentfield,cur,currentfield,
        lx<<1,16,16,bx,by,PMV[0][0][0],PMV[0][0][1],0);

    /* predict from field of opposite parity */
    pred(predframe,!currentfield,cur,currentfield,
        lx<<1,16,16,bx,by,DMV[0][0],DMV[0][1],1);
}
else
{
    /* invalid motion_type in field picture */
    if (!quiet)
        fprintf(stderr,"invalid motion_type\n");
}
}
addflag = 1; /* next prediction (if any) will be averaged with this one */
}

if (mb_type & MB_BACKWARD)
{
    /* backward prediction */

    if (pict_struct==FRAME_PICTURE)
    {
        /* frame picture */

        if (motion_type==MC_FRAME)
        {
            /* frame-based prediction in frame picture */
            pred(newref,0,cur,0,
                lx,16,16,bx,by,PMV[0][1][0],PMV[0][1][1],addflag);
        }
        else
        {
            /* field-based prediction in frame picture
             *
             * note scaling of the vertical coordinates (by, PMV[][1][1])
             * from frame to field!
             */

```

```

/* top field prediction */
pred(newref,mv_field_sel[0][1],cur,0,
    lx<<1,16,8,bx,by>>1,PMV[0][1][0],PMV[0][1][1]>>1,addflag);

/* bottom field prediction */
pred(newref,mv_field_sel[1][1],cur,1,
    lx<<1,16,8,bx,by>>1,PMV[1][1][0],PMV[1][1][1]>>1,addflag);
}
}
else /* TOP_FIELD or BOTTOM_FIELD */
{
/* field picture */

currentfield = (pict_struct==BOTTOM_FIELD);

if (motion_type==MC_FIELD)
{
/* field-based prediction in field picture */
pred(newref,mv_field_sel[0][1],cur,currentfield,
    lx<<1,16,16,bx,by,PMV[0][1][0],PMV[0][1][1],addflag);
}
else if (motion_type==MC_16X8)
{
/* 16 x 8 motion compensation in field picture */

/* upper half */
pred(newref,mv_field_sel[0][1],cur,currentfield,
    lx<<1,16,8,bx,by,PMV[0][1][0],PMV[0][1][1],addflag);

/* lower half */
pred(newref,mv_field_sel[1][1],cur,currentfield,
    lx<<1,16,8,bx,by+8,PMV[1][1][0],PMV[1][1][1],addflag);
}
else
{
/* invalid motion_type in field picture */
if (!quiet)
    fprintf(stderr,"invalid motion_type\n");
}
}
}
}

/* predict a rectangular block (all three components)
*
* src:    source frame (Y,U,V)
* sfield: source field select (0: frame or top field, 1: bottom field)
* dst:    destination frame (Y,U,V)
* dfield: destination field select (0: frame or top field, 1: bottom field)
*
* the following values are in luminance picture (frame or field) dimensions
* lx:     distance of vertically adjacent pels (selects frame or field pred.)
* w,h:    width and height of block (only 16x16 or 16x8 are used)
* x,y:    coordinates of destination block
* dx,dy:  half pel motion vector
* addflag: store or add (= average) prediction
*/
static void pred(src,sfield,dst,dfield,lx,w,h,x,y,dx,dy,addflag)
unsigned char *src[];

```

```

int sfield;
unsigned char *dst[];
int dfield;
int lx;
int w, h;
int x, y;
int dx, dy;
int addflag;
{
    int cc;

    for (cc=0; cc<3; cc++)
    {
        if (cc==1)
        {
            /* scale for color components */
            if (chroma_format==CHROMA420)
            {
                /* vertical */
                h >>= 1; y >>= 1; dy /= 2;
            }
            if (chroma_format!=CHROMA444)
            {
                /* horizontal */
                w >>= 1; x >>= 1; dx /= 2;
                lx >>= 1;
            }
        }
        pred_comp(src[cc]+(sfield?lx>>1:0),dst[cc]+(dfield?lx>>1:0),
            lx,w,h,x,y,dx,dy,addflag);
    }
}

/* low level prediction routine
*
* src:    prediction source
* dst:    prediction destination
* lx:     line width (for both src and dst)
* x,y:    destination coordinates
* dx,dy:  half pel motion vector
* w,h:    size of prediction block
* addflag: store or add prediction
*/

static void pred_comp(src,dst,lx,w,h,x,y,dx,dy,addflag)
unsigned char *src;
unsigned char *dst;
int lx;
int w, h;
int x, y;
int dx, dy;
int addflag;
{
    int xint, xh, yint, yh;
    int i, j;
    unsigned char *s, *d;

    /* half pel scaling */
    xint = dx>>1; /* integer part */

```

```

    xh = dx & 1; /* half pel flag */
    yint = dy>>1;
    yh = dy & 1;

    /* origins */
    s = src + lx*(y+yint) + (x+xint); /* motion vector */
    d = dst + lx*y + x;

    if (!xh && !yh)
        if (addflag)
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = (unsigned int)(d[i]+s[i]+1)>>1;
                s+= lx;
                d+= lx;
            }
        else
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = s[i];
                s+= lx;
                d+= lx;
            }
    else if (!xh && yh)
        if (addflag)
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = (d[i] + ((unsigned int)(s[i]+s[i+lx]+1)>>1)+1)>>1;
                s+= lx;
                d+= lx;
            }
        else
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = (unsigned int)(s[i]+s[i+lx]+1)>>1;
                s+= lx;
                d+= lx;
            }
    else if (xh && !yh)
        if (addflag)
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = (d[i] + ((unsigned int)(s[i]+s[i+1]+1)>>1)+1)>>1;
                s+= lx;
                d+= lx;
            }
        else
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = (unsigned int)(s[i]+s[i+1]+1)>>1;
                s+= lx;
                d+= lx;
            }
    }

```

```

else /* if (xh && yh) */
    if (addflag)
        for (j=0; j<h; j++)
        {
            for (i=0; i<w; i++)
                d[i] = (d[i] + ((unsigned int)(s[i]+s[i+1]+s[i+lx]+s[i+lx+1]+2)>>2)+1)>>1;
            s+= lx;
            d+= lx;
        }
    else
        for (j=0; j<h; j++)
        {
            for (i=0; i<w; i++)
                d[i] = (unsigned int)(s[i]+s[i+1]+s[i+lx]+s[i+lx+1]+2)>>2;
            s+= lx;
            d+= lx;
        }
}

```

```

/* calculate derived motion vectors (DMV) for dual prime prediction
* dmvector[2]: differential motion vectors (-1,0,+1)
* mvx,mvy: motion vector (for same parity)
*
* DMV[2][2]: derived motion vectors (for opposite parity)
*
* uses global variables pict_struct and topfirst
*
* Notes:
* - all vectors are in field coordinates (even for frame pictures)
*/

```

```

static void calc_DMV(DMV,dmvector,mvx,mvy)
int DMV[][2];
int *dmvector;
int mvx, mvy;
{
    if (pict_struct==FRAME_PICTURE)
    {
        if (topfirst)
        {
            /* vector for prediction of top field from bottom field */
            DMV[0][0] = ((mvx + (mvx>0))>>1) + dmvector[0];
            DMV[0][1] = ((mvy + (mvy>0))>>1) + dmvector[1] - 1;

            /* vector for prediction of bottom field from top field */
            DMV[1][0] = ((3*mvx+(mvx>0))>>1) + dmvector[0];
            DMV[1][1] = ((3*mvy+(mvy>0))>>1) + dmvector[1] + 1;
        }
        else
        {
            /* vector for prediction of top field from bottom field */
            DMV[0][0] = ((3*mvx+(mvx>0))>>1) + dmvector[0];
            DMV[0][1] = ((3*mvy+(mvy>0))>>1) + dmvector[1] - 1;

            /* vector for prediction of bottom field from top field */
            DMV[1][0] = ((mvx + (mvx>0))>>1) + dmvector[0];
            DMV[1][1] = ((mvy + (mvy>0))>>1) + dmvector[1] + 1;
        }
    }
}

```



```

    }
    else
    {
        /* vector for prediction from field of opposite 'parity' */
        DMV[0][0] = ((mvx+(mvx>0))>>1) + dmvector[0];
        DMV[0][1] = ((mvy+(mvy>0))>>1) + dmvector[1];

        /* correct for vertical field shift */
        if (pict_struct==TOP_FIELD)
            DMV[0][1]--;
        else
            DMV[0][1]++;
    }
}

static void clearblock(cur,i0,j0)
unsigned char *cur[];
int i0,j0;
{
    int i, j, w, h;
    unsigned char *p;

    p = cur[0] + ((pict_struct==BOTTOM_FIELD) ? width : 0) + i0 + width2*j0;

    for (j=0; j<16; j++)
    {
        for (i=0; i<16; i++)
            p[i] = 128;
        p+= width2;
    }

    w = h = 16;

    if (chroma_format!=CHROMA444)
    {
        i0>>=1; w>>=1;
    }

    if (chroma_format==CHROMA420)
    {
        j0>>=1; h>>=1;
    }

    p = cur[1] + ((pict_struct==BOTTOM_FIELD) ? chrom_width : 0) + i0
        + chrom_width2*j0;

    for (j=0; j<h; j++)
    {
        for (i=0; i<w; i++)
            p[i] = 128;
        p+= chrom_width2;
    }

    p = cur[2] + ((pict_struct==BOTTOM_FIELD) ? chrom_width : 0) + i0
        + chrom_width2*j0;

    for (j=0; j<h; j++)
    {
        for (i=0; i<w; i++)

```

```

    p[i] = 128;
    p+= chrom_width2;
}
}

putbits.c
/* putbits.c, bit-level output */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include "config.h"

extern FILE *outfile; /* the only global var we need here */

/* private data */
static unsigned char outbfr;
static int outcnt;
static int bytecnt;

/* initialize buffer, call once before first putbits or alignbits */
void initbits()
{
    outcnt = 8;
    bytecnt = 0;
}

/* write rightmost n (0<=n<=32) bits of val to outfile */
void putbits(val,n)
int val;
int n;
{
    int i;

```

```

unsigned int mask;

mask = 1 << (n-1); /* selects first (leftmost) bit */

for (i=0; i<n; i++)
{
    outbfr <<= 1;

    if (val & mask)
        outbfr|= 1;

    mask >>= 1; /* select next bit */
    outcnt--;

    if (outcnt==0) /* 8 bit buffer full */
    {
        putc(outbfr,outfile);
        outcnt = 8;
        bytecnt++;
    }
}

/* zero bit stuffing to next byte boundary (5.2.3, 6.2.1) */
void alignbits()
{
    if (outcnt!=8)
        putbits(0,outcnt);
}

/* return total number of generated bits */
int bitcount()
{
    return 8*bytecnt + (8-outcnt);
}

puthdr.c
/* puthdr.c, generation of headers */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
*
*/

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "global.h"

/* private prototypes */
static int frametotc _ANSI_ARGS__((int frame));

/* generate sequence header (6.2.2.1, 6.3.3)
*
* matrix download not implemented
*/
void putseqhdr()
{
    int i;

    alignbits();
    putbits(SEQ_START_CODE,32); /* sequence_header_code */
    putbits(horizontal_size,12); /* horizontal_size_value */
    putbits(vertical_size,12); /* vertical_size_value */
    putbits(aspectratio,4); /* aspect_ratio_information */
    putbits(frame_rate_code,4); /* frame_rate_code */
    putbits(((int)ceil(bit_rate/400.0)),18); /* bit_rate_value */
    putbits(1,1); /* marker_bit */
    putbits(vbv_buffer_size,10); /* vbv_buffer_size_value */
    putbits(constrparms,1); /* constrained_parameters_flag */

    putbits(load_iquant,1); /* load_intra_quantizer_matrix */
    if (load_iquant)
        for (i=0; i<64; i++) /* matrices are always downloaded in zig-zag order */
            putbits(intra_q[zig_zag_scan[i]],8); /* intra_quantizer_matrix */

    putbits(load_niquant,1); /* load_non_intra_quantizer_matrix */
    if (load_niquant)
        for (i=0; i<64; i++)
            putbits(inter_q[zig_zag_scan[i]],8); /* non_intra_quantizer_matrix */
}

/* generate sequence extension (6.2.2.3, 6.3.5) header (MPEG-2 only) */
void putseqext()
{
    alignbits();
    putbits(EXT_START_CODE,32); /* extension_start_code */
    putbits(SEQ_ID,4); /* extension_start_code_identifier */
    putbits((profile<<4)|level,8); /* profile_and_level_indication */
    putbits(prog_seq,1); /* progressive sequence */
    putbits(chroma_format,2); /* chroma_format */
    putbits(horizontal_size>>12,2); /* horizontal_size_extension */
    putbits(vertical_size>>12,2); /* vertical_size_extension */
    putbits(((int)ceil(bit_rate/400.0))>>18,12); /* bit_rate_extension */
    putbits(1,1); /* marker_bit */

```

```

    putbits(vbv_buffer_size>>10,8); /* vbv_buffer_size_extension */
    putbits(0,1); /* low_delay -- currently not implemented */
    putbits(0,2); /* frame_rate_extension_n */
    putbits(0,5); /* frame_rate_extension_d */
}

/* generate sequence display extension (6.2.2.4, 6.3.6)
 *
 * content not yet user settable
 */
void putseqdispest()
{
    alignbits();
    putbits(EXT_START_CODE,32); /* extension_start_code */
    putbits(DISP_ID,4); /* extension_start_code_identifier */
    putbits(video_format,3); /* video_format */
    putbits(1,1); /* colour_description */
    putbits(color_primaries,8); /* colour_primaries */
    putbits(transfer_characteristics,8); /* transfer_characteristics */
    putbits(matrix_coefficients,8); /* matrix_coefficients */
    putbits(display_horizontal_size,14); /* display_horizontal_size */
    putbits(1,1); /* marker_bit */
    putbits(display_vertical_size,14); /* display_vertical_size */
}

/* output a zero terminated string as user data (6.2.2.2.2, 6.3.4.1)
 *
 * string must not emulate start codes
 */
void putuserdata(userdata)
char *userdata;
{
    alignbits();
    putbits(USER_START_CODE,32); /* user_data_start_code */
    while (*userdata)
        putbits(*userdata++,8);
}

/* generate group of pictures header (6.2.2.6, 6.3.9)
 *
 * uses tc0 (timecode of first frame) and frame0 (number of first frame)
 */
void putgophdr(frame,closed_gop)
int frame,closed_gop;
{
    int tc;

    alignbits();
    putbits(GOP_START_CODE,32); /* group_start_code */
    tc = frametotc(tc0+frame);
    putbits(tc,25); /* time_code */
    putbits(closed_gop,1); /* closed_gop */
    putbits(0,1); /* broken_link */
}

/* convert frame number to time_code
 *
 * drop_frame not implemented
 */

```

```

static int frametotc(frame)
int frame;
{
    int fps, pict, sec, minute, hour, tc;

    fps = (int)(frame_rate+0.5);
    pict = frame%fps;
    frame = (frame-pict)/fps;
    sec = frame%60;
    frame = (frame-sec)/60;
    minute = frame%60;
    frame = (frame-minute)/60;
    hour = frame%24;
    tc = (hour<<19) | (minute<<13) | (1<<12) | (sec<<6) | pict;

    return tc;
}

/* generate picture header (6.2.3, 6.3.10) */
void putpichdr()
{
    alignbits();
    putbits(PICTURE_START_CODE,32); /* picture_start_code */
    calc_vbv_delay();
    putbits(temp_ref,10); /* temporal_reference */
    putbits(pict_type,3); /* picture_coding_type */
    putbits(vbv_delay,16); /* vbv_delay */

    if (pict_type==P_TYPE || pict_type==B_TYPE)
    {
        putbits(0,1); /* full_pel_forward_vector */
        if (mpeg1)
            putbits(forw_hor_f_code,3);
        else
            putbits(7,3); /* forward_f_code */
    }

    if (pict_type==B_TYPE)
    {
        putbits(0,1); /* full_pel_backward_vector */
        if (mpeg1)
            putbits(back_hor_f_code,3);
        else
            putbits(7,3); /* backward_f_code */
    }

    putbits(0,1); /* extra_bit_picture */
}

/* generate picture coding extension (6.2.3.1, 6.3.11)
 *
 * composite display information (v_axis etc.) not implemented
 */
void putpictcodext()
{
    alignbits();
    putbits(EXT_START_CODE,32); /* extension_start_code */
    putbits(CODING_ID,4); /* extension_start_code_identifier */
    putbits(forw_hor_f_code,4); /* forward_horizontal_f_code */

```

```

    putbits(forw_vert_f_code,4); /* forward_vertical_f_code */
    putbits(back_hor_f_code,4); /* backward_horizontal_f_code */
    putbits(back_vert_f_code,4); /* backward_vertical_f_code */
    putbits(dc_prec,2); /* intra_dc_precision */
    putbits(pict_struct,2); /* picture_structure */
    putbits((pict_struct==FRAME_PICTURE)?topfirst:0,1); /* top_field_first */
    putbits(frame_pred_dct,1); /* frame_pred_frame_dct */
    putbits(0,1); /* concealment_motion_vectors -- currently not implemented */
    putbits(q_scale_type,1); /* q_scale_type */
    putbits(intravlc,1); /* intra_vlc_format */
    putbits(altscan,1); /* alternate_scan */
    putbits(repeatfirst,1); /* repeat_first_field */
    putbits(prog_frame,1); /* chroma_420_type */
    putbits(prog_frame,1); /* progressive_frame */
    putbits(0,1); /* composite_display_flag */
}

/* generate sequence_end_code (6.2.2) */
void putseqend()
{
    alignbits();
    putbits(SEQ_END_CODE,32);
}

putmpg.c
/* putmpg.c, block and motion vector encoding routines */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include "config.h"
#include "global.h"

```

```

/* generate variable length codes for an intra-coded block (6.2.6, 6.3.17) */
void putintrablkc(blk,cc)
short *blk;
int cc;
{
    int n, dct_diff, run, signed_level;

    /* DC coefficient (7.2.1) */
    dct_diff = blk[0] - dc_dct_pred[cc]; /* difference to previous block */
    dc_dct_pred[cc] = blk[0];

    if (cc==0)
        putDClum(dct_diff);
    else
        putDCchrom(dct_diff);

    /* AC coefficients (7.2.2) */
    run = 0;
    for (n=1; n<64; n++)
    {
        /* use appropriate entropy scanning pattern */
        signed_level = blk[(altscan ? alternate_scan : zig_zag_scan)[n]];
        if (signed_level!=0)
        {
            putAC(run,signed_level,intravlc);
            run = 0;
        }
        else
            run++; /* count zero coefficients */
    }

    /* End of Block -- normative block punctuation */
    if (intravlc)
        putbits(6,4); /* 0110 (Table B-15) */
    else
        putbits(2,2); /* 10 (Table B-14) */
}

/* generate variable length codes for a non-intra-coded block (6.2.6, 6.3.17) */
void putnonintrablkc(blk)
short *blk;
{
    int n, run, signed_level, first;

    run = 0;
    first = 1;

    for (n=0; n<64; n++)
    {
        /* use appropriate entropy scanning pattern */
        signed_level = blk[(altscan ? alternate_scan : zig_zag_scan)[n]];

        if (signed_level!=0)
        {
            if (first)
            {
                /* first coefficient in non-intra block */
                putACfirst(run,signed_level);
                first = 0;
            }
        }
    }
}

```



```

    }
    else
        putAC(run,signed_level,0);

    run = 0;
}
else
    run++; /* count zero coefficients */
}

/* End of Block -- normative block punctuation */
putbits(2,2);
}

/* generate variable length code for a motion vector component (7.6.3.1) */
void putmv(dmv,f_code)
int dmv,f_code;
{
    int r_size, f, vmin, vmax, dv, temp, motion_code, motion_residual;

    r_size = f_code - 1; /* number of fixed length code ('residual') bits */
    f = 1<<r_size;
    vmin = -16*f; /* lower range limit */
    vmax = 16*f - 1; /* upper range limit */
    dv = 32*f;

    /* fold vector difference into [vmin...vmax] */
    if (dmv>vmax)
        dmv-= dv;
    else if (dmv<vmin)
        dmv+= dv;

    /* check value */
    if (dmv<vmin || dmv>vmax)
        if (!quiet)
            fprintf(stderr,"invalid motion vector\n");

    /* split dmv into motion_code and motion_residual */
    temp = ((dmv<0) ? -dmv : dmv) + f - 1;
    motion_code = temp>>r_size;
    if (dmv<0)
        motion_code = -motion_code;
    motion_residual = temp & (f-1);

    putmotioncode(motion_code); /* variable length code */

    if (r_size!=0 && motion_code!=0)
        putbits(motion_residual,r_size); /* fixed length code */
}

```

putpic.c

```

/* putpic.c, block and motion vector encoding routines          */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
* any and all warranties, whether express, implied, or statutory, 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.
*
* The MPEG Software Simulation Group 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.
*
*/

```

```

#include <stdio.h>
#include "config.h"
#include "global.h"

/* private prototypes */
static void putmvs_ANSI_ARGS_((int MV[2][2][2], int PMV[2][2][2],
    int mv_field_sel[2][2], int dmvector[2], int s, int motion_type,
    int hor_f_code, int vert_f_code));

/* quantization / variable length encoding of a complete picture */
void putpict(frame)
unsigned char *frame;
{
    int i, j, k, comp, cc;
    int mb_type;
    int PMV[2][2][2];
    int prev_mquant;
    int cbp, MBAinc;

    rc_init_pict(frame); /* set up rate control */

    /* picture header and picture coding extension */
    putpict_hdr();

    if (!mpeg1)
        putpict_codext();

    prev_mquant = rc_start_mb(); /* initialize quantization parameter */

    k = 0;

    for (j=0; j<mb_height2; j++)
    {
        /* macroblock row loop */

        for (i=0; i<mb_width; i++)

```

```

{
/* macroblock loop */
if (i==0)
{
/* slice header (6.2.4) */
alignbits();

if (mpeg1 || vertical_size<=2800)
putbits(SLICE_MIN_START+j,32); /* slice_start_code */
else
{
putbits(SLICE_MIN_START+(j&127),32); /* slice_start_code */
putbits(j>>7,3); /* slice_vertical_position_extension */
}

/* quantiser_scale_code */
putbits(q_scale_type ? map_non_linear_mquant[prev_mquant]
: prev_mquant >> 1, 5);

putbits(0,1); /* extra_bit_slice */

/* reset predictors */

for (cc=0; cc<3; cc++)
dc_dct_pred[cc] = 0;

PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;

MBAinc = i + 1; /* first MBAinc denotes absolute position */
}

mb_type = mbinfo[k].mb_type;

/* determine mquant (rate control) */
mbinfo[k].mquant = rc_calc_mquant(k);

/* quantize macroblock */
if (mb_type & MB_INTRA)
{
for (comp=0; comp<block_count; comp++)
quant_intra(blocks[k*block_count+comp],blocks[k*block_count+comp],
dc_prec,intra_q,mbinfo[k].mquant);
mbinfo[k].cbp = cbp = (1<<block_count) - 1;
}
else
{
cbp = 0;
for (comp=0;comp<block_count;comp++)
cbp = (cbp<<1) | quant_non_intra(blocks[k*block_count+comp],
blocks[k*block_count+comp],
inter_q,mbinfo[k].mquant);

mbinfo[k].cbp = cbp;

if (cbp)
mb_type|= MB_PATTERN;
}
}

```

```

/* output mquant if it has changed */
if (cbp && prev_mquant!=mbinfo[k].mquant)
    mb_type|= MB_QUANT;

/* check if macroblock can be skipped */
if (i!=0 && i!=mb_width-1 && !cbp)
{
    /* no DCT coefficients and neither first nor last macroblock of slice */

    if (pict_type==P_TYPE && !(mb_type&MB_FORWARD))
    {
        /* P picture, no motion vectors -> skip */

        /* reset predictors */

        for (cc=0; cc<3; cc++)
            dc_dct_pred[cc] = 0;

        PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
        PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;

        mbinfo[k].mb_type = mb_type;
        mbinfo[k].skipped = 1;
        MBAinc++;
        k++;
        continue;
    }

    if (pict_type==B_TYPE && pict_struct==FRAME_PICTURE
        && mbinfo[k].motion_type==MC_FRAME
        && ((mbinfo[k-1].mb_type^mb_type)&(MB_FORWARD|MB_BACKWARD))==0
        && (!(mb_type&MB_FORWARD) ||
            (PMV[0][0][0]==mbinfo[k].MV[0][0][0] &&
             PMV[0][0][1]==mbinfo[k].MV[0][0][1]))
        && (!(mb_type&MB_BACKWARD) ||
            (PMV[0][1][0]==mbinfo[k].MV[0][1][0] &&
             PMV[0][1][1]==mbinfo[k].MV[0][1][1])))
    {
        /* conditions for skipping in B frame pictures:
        * - must be frame predicted
        * - must be the same prediction type (forward/backward/interp.)
        *   as previous macroblock
        * - relevant vectors (forward/backward/both) have to be the same
        *   as in previous macroblock
        */

        mbinfo[k].mb_type = mb_type;
        mbinfo[k].skipped = 1;
        MBAinc++;
        k++;
        continue;
    }

    if (pict_type==B_TYPE && pict_struct!=FRAME_PICTURE
        && mbinfo[k].motion_type==MC_FIELD
        && ((mbinfo[k-1].mb_type^mb_type)&(MB_FORWARD|MB_BACKWARD))==0
        && (!(mb_type&MB_FORWARD) ||
            (PMV[0][0][0]==mbinfo[k].MV[0][0][0] &&
             PMV[0][0][1]==mbinfo[k].MV[0][0][1] &&

```

```

        mbinfo[k].mv_field_sel[0][0]==(pict_struct==BOTTOM_FIELD)))
        && (!(mb_type&MB_BACKWARD) ||
        (PMV[0][1][0]==mbinfo[k].MV[0][1][0] &&
        PMV[0][1][1]==mbinfo[k].MV[0][1][1] &&
        mbinfo[k].mv_field_sel[0][1]==(pict_struct==BOTTOM_FIELD))))
    {
        /* conditions for skipping in B field pictures:
        * - must be field predicted
        * - must be the same prediction type (forward/backward/interp.)
        *   as previous macroblock
        * - relevant vectors (forward/backward/both) have to be the same
        *   as in previous macroblock
        * - relevant motion_vertical_field_selects have to be of same
        *   parity as current field
        */

        mbinfo[k].mb_type = mb_type;
        mbinfo[k].skipped = 1;
        MBAinc++;
        k++;
        continue;
    }
}

/* macroblock cannot be skipped */
mbinfo[k].skipped = 0;

/* there's no VLC for 'No MC, Not Coded':
* we have to transmit (0,0) motion vectors
*/
if (pict_type==P_TYPE && !cbp && !(mb_type&MB_FORWARD))
    mb_type|= MB_FORWARD;

putaddrinc(MBAinc); /* macroblock_address_increment */
MBAinc = 1;

putmbtype(pict_type,mb_type); /* macroblock type */

if (mb_type & (MB_FORWARD|MB_BACKWARD) && !frame_pred_dct)
    putbits(mbinfo[k].motion_type,2);

if (pict_struct==FRAME_PICTURE && cbp && !frame_pred_dct)
    putbits(mbinfo[k].dct_type,1);

if (mb_type & MB_QUANT)
{
    putbits(q_scale_type ? map_non_linear_mquant[mbinfo[k].mquant]
        : mbinfo[k].mquant>>1,5);
    prev_mquant = mbinfo[k].mquant;
}

if (mb_type & MB_FORWARD)
{
    /* forward motion vectors, update predictors */
    putmvs(mbinfo[k].MV,PMV,mbinfo[k].mv_field_sel,mbinfo[k].dmvector,0,
        mbinfo[k].motion_type,forw_hor_f_code,forw_vert_f_code);
}

if (mb_type & MB_BACKWARD)

```

```

{
    /* backward motion vectors, update predictors */
    putmvs(mbinfo[k].MV,PMV,mbinfo[k].mv_field_sel,mbinfo[k].dmvector,1,
        mbinfo[k].motion_type,back_hor_f_code,back_vert_f_code);
}

if (mb_type & MB_PATTERN)
{
    putcbp((cbp >> (block_count-6)) & 63);
    if (chroma_format!=CHROMA420)
        putbits(cbp,block_count-6);
}

for (comp=0; comp<block_count; comp++)
{
    /* block loop */
    if (cbp & (1<<(block_count-1-comp)))
    {
        if (mb_type & MB_INTRA)
        {
            cc = (comp<4) ? 0 : (comp&1)+1;
            putintrablkc(blocks[k*block_count+comp],cc);
        }
        else
            putnonintrablkc(blocks[k*block_count+comp]);
    }
}

/* reset predictors */
if (!(mb_type & MB_INTRA))
    for (cc=0; cc<3; cc++)
        dc_dct_pred[cc] = 0;

if (mb_type & MB_INTRA || (pict_type==P_TYPE && !(mb_type & MB_FORWARD)))
{
    PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
    PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;
}

mbinfo[k].mb_type = mb_type;
k++;
}

rc_update_pict();
vbm_end_of_picture();
}

/* output motion vectors (6.2.5.2, 6.3.16.2)
*
* this routine also updates the predictions for motion vectors (PMV)
*/

static void putmvs(MV,PMV,mv_field_sel,dmvector,s,motion_type,
    hor_f_code,vert_f_code)
int MV[2][2][2],PMV[2][2][2];
int mv_field_sel[2][2];
int dmvector[2];

```

```

int s,motion_type,hor_f_code,vert_f_code;
{
  if (pict_struct==FRAME_PICTURE)
  {
    if (motion_type==MC_FRAME)
    {
      /* frame prediction */
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
      putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
      PMV[0][s][0]=MV[0][s][0];
      PMV[0][s][1]=MV[0][s][1];
    }
    else if (motion_type==MC_FIELD)
    {
      /* field prediction */
      putbits(mv_field_sel[0][s],1);
      putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
      putmv((MV[0][s][1]>>1)-(PMV[0][s][1]>>1),vert_f_code);
      putbits(mv_field_sel[1][s],1);
      putmv(MV[1][s][0]-PMV[1][s][0],hor_f_code);
      putmv((MV[1][s][1]>>1)-(PMV[1][s][1]>>1),vert_f_code);
      PMV[0][s][0]=MV[0][s][0];
      PMV[0][s][1]=MV[0][s][1];
      PMV[1][s][0]=MV[1][s][0];
      PMV[1][s][1]=MV[1][s][1];
    }
  }
  else
  {
    /* dual prime prediction */
    putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
    putdmv(dmvector[0]);
    putmv((MV[0][s][1]>>1)-(PMV[0][s][1]>>1),vert_f_code);
    putdmv(dmvector[1]);
    PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0];
    PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
  }
}
else
{
  /* field picture */
  if (motion_type==MC_FIELD)
  {
    /* field prediction */
    putbits(mv_field_sel[0][s],1);
    putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
    putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
    PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0];
    PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
  }
  else if (motion_type==MC_16X8)
  {
    /* 16x8 prediction */
    putbits(mv_field_sel[0][s],1);
    putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
    putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
    putbits(mv_field_sel[1][s],1);
    putmv(MV[1][s][0]-PMV[1][s][0],hor_f_code);
    putmv(MV[1][s][1]-PMV[1][s][1],vert_f_code);
    PMV[0][s][0]=MV[0][s][0];
  }
}

```

```

    PMV[0][s][1]=MV[0][s][1];
    PMV[1][s][0]=MV[1][s][0];
    PMV[1][s][1]=MV[1][s][1];
}
else
{
    /* dual prime prediction */
    putmv(MV[0][s][0]-PMV[0][s][0],hor_f_code);
    putdmv(dmvector[0]);
    putmv(MV[0][s][1]-PMV[0][s][1],vert_f_code);
    putdmv(dmvector[1]);
    PMV[0][s][0]=PMV[1][s][0]=MV[0][s][0];
    PMV[0][s][1]=PMV[1][s][1]=MV[0][s][1];
}
}
}

putseq.c
/* putseq.c, sequence level routines */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include <string.h>
#include "config.h"
#include "global.h"

void putseq()
{
    /* this routine assumes (N % M) == 0 */
    int i, j, k, f, f0, n, np, nb, sxf, syf, sxb, syb;
    int ipflag;
    FILE *fd;
    char name[256];

```



```

unsigned char *neworg[3], *newref[3];
static char ipb[5] = {' ', 'I', 'P', 'B', 'D'};

rc_init_seq(); /* initialize rate control */

/* sequence header, sequence extension and sequence display extension */
putseqhdr();
if (!mpeg1)
{
    putseqext();
    putseqdispest();
}

/* optionally output some text data (description, copyright or whatever) */
if (strlen(id_string) > 1)
    putuserdata(id_string);

/* loop through all frames in encoding/decoding order */
for (i=0; i<nframes; i++)
{
    if (!quiet)
    {
        fprintf(stderr, "Encoding frame %d ", i);
        fflush(stderr);
    }

    /* f0: lowest frame number in current GOP
     *
     * first GOP contains N-(M-1) frames,
     * all other GOPs contain N frames
     */
    f0 = N*((i+(M-1))/N) - (M-1);

    if (f0<0)
        f0=0;

    if (i==0 || (i-1)%M==0)
    {
        /* I or P frame */
        for (j=0; j<3; j++)
        {
            /* shuffle reference frames */
            neworg[j] = oldorgframe[j];
            newref[j] = oldrefframe[j];
            oldorgframe[j] = neworgframe[j];
            oldrefframe[j] = newrefframe[j];
            neworgframe[j] = neworg[j];
            newrefframe[j] = newref[j];
        }

        /* f: frame number in display order */
        f = (i==0) ? 0 : i+M-1;
        if (f>=nframes)
            f = nframes - 1;

        if (i==f0) /* first displayed frame in GOP is I */
        {
            /* I frame */
            pict_type = I_TYPE;

```

```

forw_hor_f_code = forw_vert_f_code = 15;
back_hor_f_code = back_vert_f_code = 15;

/* n: number of frames in current GOP
 *
 * first GOP contains (M-1) less (B) frames
 */
n = (i==0) ? N-(M-1) : N;

/* last GOP may contain less frames */
if (n > nframes-f0)
    n = nframes-f0;

/* number of P frames */
if (i==0)
    np = (n + 2*(M-1))/M - 1; /* first GOP */
else
    np = (n + (M-1))/M - 1;

/* number of B frames */
nb = n - np - 1;

rc_init_GOP(np,nb);

putgophdr(f0,i==0); /* set closed_GOP in first GOP only */
}
else
{
    /* P frame */
    pict_type = P_TYPE;
    forw_hor_f_code = motion_data[0].forw_hor_f_code;
    forw_vert_f_code = motion_data[0].forw_vert_f_code;
    back_hor_f_code = back_vert_f_code = 15;
    sxf = motion_data[0].sxf;
    syf = motion_data[0].syf;
}
}
else
{
    /* B frame */
    for (j=0; j<3; j++)
    {
        neworg[j] = auxorgframe[j];
        newref[j] = auxframe[j];
    }

    /* f: frame number in display order */
    f = i - 1;
    pict_type = B_TYPE;
    n = (i-2)%M + 1; /* first B: n=1, second B: n=2, ... */
    forw_hor_f_code = motion_data[n].forw_hor_f_code;
    forw_vert_f_code = motion_data[n].forw_vert_f_code;
    back_hor_f_code = motion_data[n].back_hor_f_code;
    back_vert_f_code = motion_data[n].back_vert_f_code;
    sxf = motion_data[n].sxf;
    syf = motion_data[n].syf;
    sxb = motion_data[n].sxb;
    syb = motion_data[n].syb;
}
}

```

```

temp_ref = f - f0;
frame_pred_dct = frame_pred_dct_tab[pict_type-1];
q_scale_type = qscale_tab[pict_type-1];
intravlc = intravlc_tab[pict_type-1];
altscan = altscan_tab[pict_type-1];

fprintf(statfile,"\nFrame %d (#%d in display order):\n",i,f);
fprintf(statfile," picture_type=%c\n",ipb[pict_type]);
fprintf(statfile," temporal_reference=%d\n",temp_ref);
fprintf(statfile," frame_pred_frame_dct=%d\n",frame_pred_dct);
fprintf(statfile," q_scale_type=%d\n",q_scale_type);
fprintf(statfile," intra_vlc_format=%d\n",intravlc);
fprintf(statfile," alternate_scan=%d\n",altscan);

if (pict_type!=I_TYPE)
{
    fprintf(statfile," forward search window: %d...%d / %d...%d\n",
        -sxf,sxf,-syf,syf);
    fprintf(statfile," forward vector range: %d...%d.5 / %d...%d.5\n",
        -(4<<forw_hor_f_code),(4<<forw_hor_f_code)-1,
        -(4<<forw_vert_f_code),(4<<forw_vert_f_code)-1);
}

if (pict_type==B_TYPE)
{
    fprintf(statfile," backward search window: %d...%d / %d...%d\n",
        -sxb,sxb,-syb,syb);
    fprintf(statfile," backward vector range: %d...%d.5 / %d...%d.5\n",
        -(4<<back_hor_f_code),(4<<back_hor_f_code)-1,
        -(4<<back_vert_f_code),(4<<back_vert_f_code)-1);
}

sprintf(name,tplorg,f+frame0);
readframe(name,neworg);

if (fieldpic)
{
    if (!quiet)
    {
        fprintf(stderr,"\nfirst field (%s) ",topfirst ? "top" : "bot");
        fflush(stderr);
    }
}

pict_struct = topfirst ? TOP_FIELD : BOTTOM_FIELD;

motion_estimation(oldorgframe[0],neworgframe[0],
    oldreframe[0],newreframe[0],
    neworg[0],newref[0],
    sxf,syf,sxb,syb,mbinfo,0,0);

predict(oldreframe,newreframe,predframe,0,mbinfo);
dct_type_estimation(predframe[0],neworg[0],mbinfo);
transform(predframe,neworg,mbinfo,blocks);

putpict(neworg[0]);

for (k=0; k<mb_height2*mb_width; k++)
{

```

```

    if (mbinfo[k].mb_type & MB_INTRA)
        for (j=0; j<block_count; j++)
            iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                        dc_prec,intra_q,mbinfo[k].mquant);
    else
        for (j=0;j<block_count;j++)
            iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                        inter_q,mbinfo[k].mquant);
}

itransform(predframe,newref,mbinfo,blocks);
calcSNR(neworg,newref);
stats();

if (!quiet)
{
    fprintf(stderr,"second field (%s) ",topfirst ? "bot" : "top");
    fflush(stderr);
}

pict_struct = topfirst ? BOTTOM_FIELD : TOP_FIELD;

ipflag = (pict_type==I_TYPE);
if (ipflag)
{
    /* first field = I, second field = P */
    pict_type = P_TYPE;
    forw_hor_f_code = motion_data[0].forw_hor_f_code;
    forw_vert_f_code = motion_data[0].forw_vert_f_code;
    back_hor_f_code = back_vert_f_code = 15;
    sxf = motion_data[0].sxf;
    syf = motion_data[0].syf;
}

motion_estimation(oldorgframe[0],neworgframe[0],
                oldrefframe[0],newrefframe[0],
                neworg[0],newref[0],
                sxf,syf,sxb,syb,mbinfo,1,ipflag);

predict(oldrefframe,newrefframe,predframe,1,mbinfo);
dct_type_estimation(predframe[0],neworg[0],mbinfo);
transform(predframe,neworg,mbinfo,blocks);

putpict(neworg[0]);

for (k=0; k<mb_height2*mb_width; k++)
{
    if (mbinfo[k].mb_type & MB_INTRA)
        for (j=0; j<block_count; j++)
            iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                        dc_prec,intra_q,mbinfo[k].mquant);
    else
        for (j=0;j<block_count;j++)
            iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                        inter_q,mbinfo[k].mquant);
}

itransform(predframe,newref,mbinfo,blocks);
calcSNR(neworg,newref);

```

```

    stats();
}
else
{
    pict_struct = FRAME_PICTURE;

    /* do motion_estimation
    *
    * uses source frames (...orgframe) for full pel search
    * and reconstructed frames (...refframe) for half pel search
    */

    motion_estimation(oldorgframe[0],neworgframe[0],
                      oldrefframe[0],newrefframe[0],
                      neworg[0],newref[0],
                      sxf,syf,sxb,syb,mbinfo,0,0);

    predict(oldrefframe,newrefframe,predframe,0,mbinfo);
    dct_type_estimation(predframe[0],neworg[0],mbinfo);
    transform(predframe,neworg,mbinfo,blocks);

    putpict(neworg[0]);

    for (k=0; k<mb_height*mb_width; k++)
    {
        if (mbinfo[k].mb_type & MB_INTRA)
            for (j=0; j<block_count; j++)
                iquant_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                             dc_prec,intra_q,mbinfo[k].mquant);
        else
            for (j=0;j<block_count;j++)
                iquant_non_intra(blocks[k*block_count+j],blocks[k*block_count+j],
                                 inter_q,mbinfo[k].mquant);
    }

    itransform(predframe,newref,mbinfo,blocks);
    calcSNR(neworg,newref);
    stats();
}

sprintf(name,tplref,f+frame0);
writeframe(name,newref);

}

putseqend();
}

putvlc.c
/* putvlc.c, generation of variable length codes */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims

```

```

* any and all warranties, whether express, implied, or statutory, 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.
*
* The MPEG Software Simulation Group 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.
*
*/

```

```
#include <stdio.h>
```

```
#include "config.h"
#include "global.h"
#include "vlc.h"
```

```
/* private prototypes */
static void putDC_ANSI_ARGS_((sVLCtable *tab, int val));
```

```
/* generate variable length code for luminance DC coefficient */
void putDClum(val)
int val;
{
    putDC(DClumtab,val);
}

```

```
/* generate variable length code for chrominance DC coefficient */
void putDCchrom(val)
int val;
{
    putDC(DCchromtab,val);
}

```

```
/* generate variable length code for DC coefficient (7.2.1) */
static void putDC(tab,val)
sVLCtable *tab;
int val;
{
    int absval, size;

    absval = (val<0) ? -val : val; /* abs(val) */

    if (absval>2047 || (mpeg1 && absval>255))
    {
        /* should never happen */
        sprintf(errortext,"DC value out of range (%d)\n",val);
        error(errortext);
    }
}

```

```

/* compute dct_dc_size */
size = 0;

while (absval)
{
    absval >>= 1;
    size++;
}

/* generate VLC for dct_dc_size (Table B-12 or B-13) */
putbits(tab[size].code,tab[size].len);

/* append fixed length code (dc_dct_differential) */
if (size!=0)
{
    if (val>=0)
        absval = val;
    else
        absval = val + (1<<size) - 1; /* val + (2 ^ size) - 1 */
    putbits(absval,size);
}
}

/* generate variable length code for first coefficient
 * of a non-intra block (7.2.2.2) */
void putACfirst(run,val)
int run,val;
{
    if (run==0 && (val==1 || val==-1)) /* these are treated differently */
        putbits(2|(val<0),2); /* generate '1s' (s=sign), (Table B-14, line 2) */
    else
        putAC(run,val,0); /* no difference for all others */
}

/* generate variable length code for other DCT coefficients (7.2.2) */
void putAC(run,signed_level,vlcformat)
int run,signed_level,vlcformat;
{
    int level, len;
    VLCtable *ptab;

    level = (signed_level<0) ? -signed_level : signed_level; /* abs(signed_level) */

    /* make sure run and level are valid */
    if (run<0 || run>63 || level==0 || level>2047 || (mpeg1 && level>255))
    {
        sprintf(errortext,"AC value out of range (run=%d, signed_level=%d)\n",
            run,signed_level);
        error(errortext);
    }

    len = 0;

    if (run<2 && level<41)
    {
        /* vlcformat selects either of Table B-14 / B-15 */
        if (vlcformat)
            ptab = &dct_code_tab1a[run][level-1];
        else

```

```

    ptab = &dct_code_tab1[run][level-1];

    len = ptab->len;
}
else if (run<32 && level<6)
{
    /* vlcformat selects either of Table B-14 / B-15 */
    if (vlcformat)
        ptab = &dct_code_tab2a[run-2][level-1];
    else
        ptab = &dct_code_tab2[run-2][level-1];

    len = ptab->len;
}

if (len!=0) /* a VLC code exists */
{
    putbits(ptab->code,len);
    putbits(signed_level<0,1); /* sign */
}
else
{
    /* no VLC for this (run, level) combination: use escape coding (7.2.2.3) */
    putbits(11,6); /* Escape */
    putbits(run,6); /* 6 bit code for run */
    if (mpeg1)
    {
        /* ISO/IEC 11172-2 uses a 8 or 16 bit code */
        if (signed_level>127)
            putbits(0,8);
        if (signed_level<-127)
            putbits(128,8);
        putbits(signed_level,8);
    }
    else
    {
        /* ISO/IEC 13818-2 uses a 12 bit code, Table B-16 */
        putbits(signed_level,12);
    }
}
}

/* generate variable length code for macroblock_address_increment (6.3.16) */
void putaddrinc(addrinc)
int addrinc;
{
    while (addrinc>33)
    {
        putbits(0x08,11); /* macroblock_escape */
        addrinc-= 33;
    }

    putbits(addrinctab[addrinc-1].code,addrinctab[addrinc-1].len);
}

/* generate variable length code for macroblock_type (6.3.16.1) */
void putmbtype(pict_type,mb_type)
int pict_type,mb_type;
{

```



```

    putbits(mbtypetab[pict_type-1][mb_type].code,
            mbtypetab[pict_type-1][mb_type].len);
}

/* generate variable length code for motion_code (6.3.16.3) */
void putmotioncode(motion_code)
int motion_code;
{
    int abscode;

    abscode = (motion_code>=0) ? motion_code : -motion_code; /* abs(motion_code) */
    putbits(motionvectab[abscode].code,motionvectab[abscode].len);
    if (motion_code!=0)
        putbits(motion_code<0,1); /* sign, 0=positive, 1=negative */
}

/* generate variable length code for dmvector[t] (6.3.16.3), Table B-11 */
void putdmv(dmv)
int dmv;
{
    if (dmv==0)
        putbits(0,1);
    else if (dmv>0)
        putbits(2,2);
    else
        putbits(3,2);
}

/* generate variable length code for coded_block_pattern (6.3.16.4)
 *
 * 4:2:2, 4:4:4 not implemented
 */
void putcbp(cbp)
int cbp;
{
    putbits(cbptable[cbp].code,cbptable[cbp].len);
}

```

## quantize.c

```

/* quantize.c, quantization / inverse quantization */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 */

```

```

* The MPEG Software Simulation Group 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.
*
*/

```

```

#include <stdio.h>
#include "config.h"
#include "global.h"

```

```

static void iquant1_intra _ANSI_ARGS_((short *src, short *dst,
    int dc_prec, unsigned char *quant_mat, int mquant));
static void iquant1_non_intra _ANSI_ARGS_((short *src, short *dst,
    unsigned char *quant_mat, int mquant));

```

```

/* Test Model 5 quantization

```

```

*
* this quantizer has a bias of 1/8 stepsize towards zero
* (except for the DC coefficient)
*/

```

```

int quant_intra(src,dst,dc_prec,quant_mat,mquant)
short *src, *dst;
int dc_prec;
unsigned char *quant_mat;
int mquant;
{
    int i;
    int x, y, d;

    x = src[0];
    d = 8>>dc_prec; /* intra_dc_mult */
    dst[0] = (x>=0) ? (x+(d>>1))/d : -((-x+(d>>1))/d); /* round(x/d) */

    for (i=1; i<64; i++)
    {
        x = src[i];
        d = quant_mat[i];
        y = (32*(x>=0 ? x : -x) + (d>>1))/d; /* round(32*x/quant_mat) */
        d = (3*mquant+2)>>2;
        y = (y+d)/(2*mquant); /* (y+0.75*mquant) / (2*mquant) */

        /* clip to syntax limits */
        if (y > 255)
        {
            if (mpeg1)
                y = 255;
            else if (y > 2047)
                y = 2047;
        }

        dst[i] = (x>=0) ? y : -y;
    }

    #if 0
    /* this quantizer is virtually identical to the above */

```

```

    if (x<0)
        x = -x;
    d = mquant*quant_mat[i];
    y = (16*x + ((3*d)>>3)) / d;
    dst[i] = (src[i]<0) ? -y : y;
#endif
}

return 1;
}

int quant_non_intra(src,dst,quant_mat,mquant)
short *src, *dst;
unsigned char *quant_mat;
int mquant;
{
    int i;
    int x, y, d;
    int nzflag;

    nzflag = 0;

    for (i=0; i<64; i++)
    {
        x = src[i];
        d = quant_mat[i];
        y = (32*(x>=0 ? x : -x) + (d>>1))/d; /* round(32*x/quant_mat) */
        y /= (2*mquant);

        /* clip to syntax limits */
        if (y > 255)
        {
            if (mpeg1)
                y = 255;
            else if (y > 2047)
                y = 2047;
        }

        if ((dst[i] = (x>=0 ? y : -y)) != 0)
            nzflag=1;
    }

    return nzflag;
}

/* MPEG-2 inverse quantization */
void iquant_intra(src,dst,dc_prec,quant_mat,mquant)
short *src, *dst;
int dc_prec;
unsigned char *quant_mat;
int mquant;
{
    int i, val, sum;

    if (mpeg1)
        iquantl_intra(src,dst,dc_prec,quant_mat,mquant);
    else
    {
        sum = dst[0] = src[0] << (3-dc_prec);

```

```

    for (i=1; i<64; i++)
    {
        val = (int)(src[i]*quant_mat[i]*mquant)/16;
        sum+= dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
    }

    /* mismatch control */
    if ((sum&1)==0)
        dst[63]^= 1;
    }
}

void iquant_non_intra(src,dst,quant_mat,mquant)
short *src, *dst;
unsigned char *quant_mat;
int mquant;
{
    int i, val, sum;

    if (mpeg1)
        iquant1_non_intra(src,dst,quant_mat,mquant);
    else
    {
        sum = 0;
        for (i=0; i<64; i++)
        {
            val = src[i];
            if (val!=0)
                val = (int)((2*val+(val>0 ? 1 : -1))*quant_mat[i]*mquant)/32;
            sum+= dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
        }

        /* mismatch control */
        if ((sum&1)==0)
            dst[63]^= 1;
        }
    }

    /* MPEG-1 inverse quantization */
    static void iquant1_intra(src,dst,dc_prec,quant_mat,mquant)
    short *src, *dst;
    int dc_prec;
    unsigned char *quant_mat;
    int mquant;
    {
        int i, val;

        dst[0] = src[0] << (3-dc_prec);
        for (i=1; i<64; i++)
        {
            val = (int)(src[i]*quant_mat[i]*mquant)/16;

            /* mismatch control */
            if ((val&1)==0 && val!=0)
                val+= (val>0) ? -1 : 1;

            /* saturation */
            dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
        }
    }
}

```

```

}

static void iquant1_non_intra(src,dst,quant_mat,mquant)
short *src, *dst;
unsigned char *quant_mat;
int mquant;
{
    int i, val;

    for (i=0; i<64; i++)
    {
        val = src[i];
        if (val!=0)
        {
            val = (int)((2*val+(val>0 ? 1 : -1))*quant_mat[i]*mquant)/32;

            /* mismatch control */
            if ((val&1)==0 && val!=0)
                val+= (val>0) ? -1 : 1;
        }

        /* saturation */
        dst[i] = (val>2047) ? 2047 : ((val<-2048) ? -2048 : val);
    }
}

```

ratectl.c

```

/* ratectl.c, bitrate control routines (linear quantization only currently) */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include <math.h>

```

```

#include "config.h"
#include "global.h"

/* private prototypes */
static void calc_actj _ANSI_ARGS_((unsigned char *frame));
static double var_sblk _ANSI_ARGS_((unsigned char *p, int lx));

/* rate control variables */
int Xi, Xp, Xb, r, d0i, d0p, d0b;
double avg_act;
static int R, T, d;
static double actsum;
static int Np, Nb, S, Q;
static int prev_mquant;

void rc_init_seq()
{
    /* reaction parameter (constant) */
    if (r==0) r = (int)floor(2.0*bit_rate/frame_rate + 0.5);

    /* average activity */
    if (avg_act==0.0) avg_act = 400.0;

    /* remaining # of bits in GOP */
    R = 0;

    /* global complexity measure */
    if (Xi==0) Xi = (int)floor(160.0*bit_rate/115.0 + 0.5);
    if (Xp==0) Xp = (int)floor( 60.0*bit_rate/115.0 + 0.5);
    if (Xb==0) Xb = (int)floor( 42.0*bit_rate/115.0 + 0.5);

    /* virtual buffer fullness */
    if (d0i==0) d0i = (int)floor(10.0*r/31.0 + 0.5);
    if (d0p==0) d0p = (int)floor(10.0*r/31.0 + 0.5);
    if (d0b==0) d0b = (int)floor(1.4*10.0*r/31.0 + 0.5);
    /*
    if (d0i==0) d0i = (int)floor(10.0*r/(qscale_tab[0] ? 56.0 : 31.0) + 0.5);
    if (d0p==0) d0p = (int)floor(10.0*r/(qscale_tab[1] ? 56.0 : 31.0) + 0.5);
    if (d0b==0) d0b = (int)floor(1.4*10.0*r/(qscale_tab[2] ? 56.0 : 31.0) + 0.5);
    */

    fprintf(statfile, "\nrate control: sequence initialization\n");
    fprintf(statfile,
        " initial global complexity measures (I,P,B): Xi=%d, Xp=%d, Xb=%d\n",
        Xi, Xp, Xb);
    fprintf(statfile, " reaction parameter: r=%d\n", r);
    fprintf(statfile,
        " initial virtual buffer fullness (I,P,B): d0i=%d, d0p=%d, d0b=%d\n",
        d0i, d0p, d0b);
    fprintf(statfile, " initial average activity: avg_act=%.1f\n", avg_act);
}

void rc_init_GOP(np,nb)
int np,nb;
{
    R += (int) floor((1 + np + nb) * bit_rate / frame_rate + 0.5);
    Np = fieldpic ? 2*np+1 : np;
    Nb = fieldpic ? 2*nb : nb;

```

```

fprintf(statfile, "\nrate control: new group of pictures (GOP)\n");
fprintf(statfile, " target number of bits for GOP: R=%d\n", R);
fprintf(statfile, " number of P pictures in GOP: Np=%d\n", Np);
fprintf(statfile, " number of B pictures in GOP: Nb=%d\n", Nb);
}

/* Note: we need to substitute K for the 1.4 and 1.0 constants -- this can
   be modified to fit image content */

/* Step 1: compute target bits for current picture being coded */
void rc_init_pict(frame)
unsigned char *frame;
{
    double Tmin;

    switch (pict_type)
    {
        case I_TYPE:
            T = (int) floor(R/(1.0+Np*Xp/(Xi*1.0)+Nb*Xb/(Xi*1.4)) + 0.5);
            d = d0i;
            break;
        case P_TYPE:
            T = (int) floor(R/(Np+Nb*1.0*Xb/(1.4*Xp)) + 0.5);
            d = d0p;
            break;
        case B_TYPE:
            T = (int) floor(R/(Nb+Np*1.4*Xp/(1.0*Xb)) + 0.5);
            d = d0b;
            break;
    }

    Tmin = (int) floor(bit_rate/(8.0*frame_rate) + 0.5);

    if (T<Tmin)
        T = Tmin;

    S = bitcount();
    Q = 0;

    calc_actj(frame);
    actsum = 0.0;

    fprintf(statfile, "\nrate control: start of picture\n");
    fprintf(statfile, " target number of bits: T=%d\n", T);
}

static void calc_actj(frame)
unsigned char *frame;
{
    int i,j,k;
    unsigned char *p;
    double actj,var;

    k = 0;

    for (j=0; j<height2; j+=16)
        for (i=0; i<width; i+=16)
        {

```

```

    p = frame + ((pict_struct==BOTTOM_FIELD)?width:0) + i + width2*j;

    /* take minimum spatial activity measure of luminance blocks */

    actj = var_sblk(p,width2);
    var = var_sblk(p+8,width2);
    if (var<actj) actj = var;
    var = var_sblk(p+8*width2,width2);
    if (var<actj) actj = var;
    var = var_sblk(p+8*width2+8,width2);
    if (var<actj) actj = var;

    if (!fieldpic && !prog_seq)
    {
        /* field */
        var = var_sblk(p,width<<1);
        if (var<actj) actj = var;
        var = var_sblk(p+8,width<<1);
        if (var<actj) actj = var;
        var = var_sblk(p+width,width<<1);
        if (var<actj) actj = var;
        var = var_sblk(p+width+8,width<<1);
        if (var<actj) actj = var;
    }

    actj+= 1.0;

    mbinfo[k++].act = actj;
}

void rc_update_pict()
{
    double X;

    S = bitcount() - S; /* total # of bits in picture */
    R-= S; /* remaining # of bits in GOP */
    X = (int) floor(S*((0.5*(double)Q)/(mb_width*mb_height2)) + 0.5);
    d+= S - T;
    avg_act = actsum/(mb_width*mb_height2);

    switch (pict_type)
    {
    case I_TYPE:
        Xi = X;
        d0i = d;
        break;
    case P_TYPE:
        Xp = X;
        d0p = d;
        Np--;
        break;
    case B_TYPE:
        Xb = X;
        d0b = d;
        Nb--;
        break;
    }
}

```



```

fprintf(statfile, "\nrate control: end of picture\n");
fprintf(statfile, " actual number of bits: S=%d\n", S);
fprintf(statfile, " average quantization parameter Q=%.1f\n",
(double)Q/(mb_width*mb_height2));
fprintf(statfile, " remaining number of bits in GOP: R=%d\n", R);
fprintf(statfile,
" global complexity measures (I,P,B): Xi=%d, Xp=%d, Xb=%d\n",
Xi, Xp, Xb);
fprintf(statfile,
" virtual buffer fullness (I,P,B): d0i=%d, d0p=%d, d0b=%d\n",
d0i, d0p, d0b);
fprintf(statfile, " remaining number of P pictures in GOP: Np=%d\n", Np);
fprintf(statfile, " remaining number of B pictures in GOP: Nb=%d\n", Nb);
fprintf(statfile, " average activity: avg_act=%.1f\n", avg_act);
}

/* compute initial quantization stepsize (at the beginning of picture) */
int rc_start_mb()
{
    int mquant;

    if (q_scale_type)
    {
        mquant = (int) floor(2.0*d*31.0/r + 0.5);

        /* clip mquant to legal (linear) range */
        if (mquant<1)
            mquant = 1;
        if (mquant>112)
            mquant = 112;

        /* map to legal quantization level */
        mquant = non_linear_mquant_table[map_non_linear_mquant[mquant]];
    }
    else
    {
        mquant = (int) floor(d*31.0/r + 0.5);
        mquant <=<= 1;

        /* clip mquant to legal (linear) range */
        if (mquant<2)
            mquant = 2;
        if (mquant>62)
            mquant = 62;

        prev_mquant = mquant;
    }

    /*
    fprintf(statfile, "rc_start_mb:\n");
    fprintf(statfile, "mquant=%d\n", mquant);
    */

    return mquant;
}

/* Step 2: measure virtual buffer - estimated buffer discrepancy */
int rc_calc_mquant(j)
int j;

```

```

{
    int mquant;
    double dj, Qj, actj, N_actj;

    /* measure virtual buffer discrepancy from uniform distribution model */
    dj = d + (bitcount()-S) - j*(T/(mb_width*mb_height2));

    /* scale against dynamic range of mquant and the bits/picture count */
    Qj = dj*31.0/r;
    /*Qj = dj*(q_scale_type ? 56.0 : 31.0)/r; */

    actj = mbinf[j].act;
    actsum+= actj;

    /* compute normalized activity */
    N_actj = (2.0*actj+avg_act)/(actj+2.0*avg_act);

    if (q_scale_type)
    {
        /* modulate mquant with combined buffer and local activity measures */
        mquant = (int) floor(2.0*Qj*N_actj + 0.5);

        /* clip mquant to legal (linear) range */
        if (mquant<1)
            mquant = 1;
        if (mquant>112)
            mquant = 112;

        /* map to legal quantization level */
        mquant = non_linear_mquant_table[map_non_linear_mquant[mquant]];
    }
    else
    {
        /* modulate mquant with combined buffer and local activity measures */
        mquant = (int) floor(Qj*N_actj + 0.5);
        mquant <=<= 1;

        /* clip mquant to legal (linear) range */
        if (mquant<2)
            mquant = 2;
        if (mquant>62)
            mquant = 62;

        /* ignore small changes in mquant */
        if (mquant>=8 && (mquant-prev_mquant)>=-4 && (mquant-prev_mquant)<=4)
            mquant = prev_mquant;

        prev_mquant = mquant;
    }

    Q+= mquant; /* for calculation of average mquant */

/*
    fprintf(statfile,"rc_calc_mquant(%d): ",j);
    fprintf(statfile,"bitcount=%d, dj=%f, Qj=%f, actj=%f, N_actj=%f, mquant=%d\n",
        bitcount(),dj,Qj,actj,N_actj,mquant);
*/

    return mquant;
}

```

```

}

/* compute variance of 8x8 block */
static double var_sblk(p,lx)
unsigned char *p;
int lx;
{
    int i, j;
    unsigned int v, s, s2;

    s = s2 = 0;

    for (j=0; j<8; j++)
    {
        for (i=0; i<8; i++)
        {
            v = *p++;
            s += v;
            s2 += v*v;
        }
        p += lx - 8;
    }

    return s2/64.0 - (s/64.0)*(s/64.0);
}

/* VBV calculations
 *
 * generates warnings if underflow or overflow occurs
 */

/* vbv_end_of_picture
 *
 * - has to be called directly after writing picture_data()
 * - needed for accurate VBV buffer overflow calculation
 * - assumes there is no byte stuffing prior to the next start code
 */

static int bitcnt_EOP;

void vbv_end_of_picture()
{
    bitcnt_EOP = bitcount();
    bitcnt_EOP = (bitcnt_EOP + 7) & ~7; /* account for bit stuffing */
}

/* calc_vbv_delay
 *
 * has to be called directly after writing the picture start code, the
 * reference point for vbv_delay
 */

void calc_vbv_delay()
{
    double picture_delay;
    static double next_ip_delay; /* due to frame reordering delay */
    static double decoding_time;

    /* number of 1/90000 s ticks until next picture is to be decoded */

```

```

if (pict_type == B_TYPE)
{
    if (prog_seq)
    {
        if (!repeatfirst)
            picture_delay = 90000.0/frame_rate; /* 1 frame */
        else
        {
            if (!topfirst)
                picture_delay = 90000.0*2.0/frame_rate; /* 2 frames */
            else
                picture_delay = 90000.0*3.0/frame_rate; /* 3 frames */
        }
    }
    else
    {
        /* interlaced */
        if (fieldpic)
            picture_delay = 90000.0/(2.0*frame_rate); /* 1 field */
        else
        {
            if (!repeatfirst)
                picture_delay = 90000.0*2.0/(2.0*frame_rate); /* 2 flds */
            else
                picture_delay = 90000.0*3.0/(2.0*frame_rate); /* 3 flds */
        }
    }
}
else
{
    /* I or P picture */
    if (fieldpic)
    {
        if (topfirst==(pict_struct==TOP_FIELD))
        {
            /* first field */
            picture_delay = 90000.0/(2.0*frame_rate);
        }
        else
        {
            /* second field */
            /* take frame reordering delay into account */
            picture_delay = next_ip_delay - 90000.0/(2.0*frame_rate);
        }
    }
    else
    {
        /* frame picture */
        /* take frame reordering delay into account */
        picture_delay = next_ip_delay;
    }
}

if (!fieldpic || topfirst!=(pict_struct==TOP_FIELD))
{
    /* frame picture or second field */
    if (prog_seq)
    {
        if (!repeatfirst)
            next_ip_delay = 90000.0/frame_rate;
    }
}

```

```

    else
    {
        if (!topfirst)
            next_ip_delay = 90000.0*2.0/frame_rate;
        else
            next_ip_delay = 90000.0*3.0/frame_rate;
    }
}
else
{
    if (fieldpic)
        next_ip_delay = 90000.0/(2.0*frame_rate);
    else
    {
        if (!repeatfirst)
            next_ip_delay = 90000.0*2.0/(2.0*frame_rate);
        else
            next_ip_delay = 90000.0*3.0/(2.0*frame_rate);
    }
}
}

if (decoding_time==0.0)
{
    /* first call of calc_vbv_delay */
    /* we start with a 7/8 filled VBV buffer (12.5% back-off) */
    picture_delay = ((vbv_buffer_size*16384*7)/8)*90000.0/bit_rate;
    if (fieldpic)
        next_ip_delay = (int)(90000.0/frame_rate+0.5);
}

/* VBV checks */

/* check for underflow (previous picture) */
if (!low_delay && (decoding_time < bitcnt_EOP*90000.0/bit_rate))
{
    /* picture not completely in buffer at intended decoding time */
    if (!quiet)
        fprintf(stderr, "vbv_delay underflow! (decoding_time=%.1f, t_EOP=%.1f\n)",
            decoding_time, bitcnt_EOP*90000.0/bit_rate);
}

/* when to decode current frame */
decoding_time += picture_delay;

/* warning: bitcount() may overflow (e.g. after 9 min. at 8 Mbit/s */
vbv_delay = (int)(decoding_time - bitcount()*90000.0/bit_rate);

/* check for overflow (current picture) */
if ((decoding_time - bitcnt_EOP*90000.0/bit_rate)
    > (vbv_buffer_size*16384)*90000.0/bit_rate)
{
    if (!quiet)
        fprintf(stderr, "vbv_delay overflow!\n");
}

fprintf(statfile,
    "\nvbv_delay=%d (bitcount=%d, decoding_time=%.2f, bitcnt_EOP=%d)\n",

```

```

    vbv_delay,bitcount(),decoding_time,bitcnt_EOP);

if (vbv_delay<0)
{
    if (!quiet)
        fprintf(stderr,"vbv_delay underflow: %d\n",vbv_delay);
    vbv_delay = 0;
}

if (vbv_delay>65535)
{
    if (!quiet)
        fprintf(stderr,"vbv_delay overflow: %d\n",vbv_delay);
    vbv_delay = 65535;
}
}

readpic.c
/* readpic.c, read source pictures */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "global.h"

/* private prototypes */
static void read_y_u_v _ANSI_ARGS_((char *fname, unsigned char *frame[]));
static void read_yuv _ANSI_ARGS_((char *fname, unsigned char *frame[]));
static void read_ppm _ANSI_ARGS_((char *fname, unsigned char *frame[]));
static void border_extend _ANSI_ARGS_((unsigned char *frame, int w1, int h1,

```

```

    int w2, int h2));
static void conv444to422 _ANSI_ARGS_((unsigned char *src, unsigned char *dst));
static void conv422to420 _ANSI_ARGS_((unsigned char *src, unsigned char *dst));

void readframe(fname,frame)
char *fname;
unsigned char *frame[];
{
    switch (inputtype)
    {
        case T_Y_U_V:
            read_y_u_v(fname,frame);
            break;
        case T_YUV:
            read_yuv(fname,frame);
            break;
        case T_PPM:
            read_ppm(fname,frame);
            break;
        default:
            break;
    }
}

static void read_y_u_v(fname,frame)
char *fname;
unsigned char *frame[];
{
    int i;
    int chrom_hsize, chrom_vsize;
    char name[128];
    FILE *fd;

    chrom_hsize = (chroma_format==CHROMA444) ? horizontal_size
        : horizontal_size>>1;
    chrom_vsize = (chroma_format!=CHROMA420) ? vertical_size
        : vertical_size>>1;

    sprintf(name,"%s.Y",fname);
    if (!(fd = fopen(name,"rb")))
    {
        sprintf(errortext,"Couldn't open %s\n",name);
        error(errortext);
    }
    for (i=0; i<vertical_size; i++)
        fread(frame[0]+i*width,1,horizontal_size,fd);
    fclose(fd);
    border_extend(frame[0],horizontal_size,vertical_size,width,height);

    sprintf(name,"%s.U",fname);
    if (!(fd = fopen(name,"rb")))
    {
        sprintf(errortext,"Couldn't open %s\n",name);
        error(errortext);
    }
    for (i=0; i<chrom_vsize; i++)
        fread(frame[1]+i*chrom_width,1,chrom_hsize,fd);
    fclose(fd);
    border_extend(frame[1],chrom_hsize,chrom_vsize,chrom_width,chrom_height);

```

```

sprintf(name,"%s.V",fname);
if (!(fd = fopen(name,"rb")))
{
    sprintf(errortext,"Couldn't open %s\n",name);
    error(errortext);
}
for (i=0; i<chrom_vsize; i++)
    fread(frame[2]+i*chrom_width,1,chrom_hsize,fd);
fclose(fd);
border_extend(frame[2],chrom_hsize,chrom_vsize,chrom_width,chrom_height);
}

```

```

static void read_yuv(fname,frame)
char *fname;
unsigned char *frame[];
{
    int i;
    int chrom_hsize, chrom_vsize;
    char name[128];
    FILE *fd;

    chrom_hsize = (chroma_format==CHROMA444) ? horizontal_size
        : horizontal_size>>1;
    chrom_vsize = (chroma_format!=CHROMA420) ? vertical_size
        : vertical_size>>1;

    sprintf(name,"%s.yuv",fname);
    if (!(fd = fopen(name,"rb")))
    {
        sprintf(errortext,"Couldn't open %s\n",name);
        error(errortext);
    }

    /* Y */
    for (i=0; i<vertical_size; i++)
        fread(frame[0]+i*width,1,horizontal_size,fd);
    border_extend(frame[0],horizontal_size,vertical_size,width,height);

    /* Cb */
    for (i=0; i<chrom_vsize; i++)
        fread(frame[1]+i*chrom_width,1,chrom_hsize,fd);
    border_extend(frame[1],chrom_hsize,chrom_vsize,chrom_width,chrom_height);

    /* Cr */
    for (i=0; i<chrom_vsize; i++)
        fread(frame[2]+i*chrom_width,1,chrom_hsize,fd);
    border_extend(frame[2],chrom_hsize,chrom_vsize,chrom_width,chrom_height);

    fclose(fd);
}

```

```

static void read_ppm(fname,frame)
char *fname;
unsigned char *frame[];
{
    int i, j;
    int r, g, b;
    double y, u, v;

```



```

double cr, cg, cb, cu, cv;
char name[128];
FILE *fd;
unsigned char *yp, *up, *vp;
static unsigned char *u444, *v444, *u422, *v422;
static double coef[7][3] = {
    {0.2125,0.7154,0.0721}, /* ITU-R Rec. 709 (1990) */
    {0.299, 0.587, 0.114}, /* unspecified */
    {0.299, 0.587, 0.114}, /* reserved */
    {0.30, 0.59, 0.11}, /* FCC */
    {0.299, 0.587, 0.114}, /* ITU-R Rec. 624-4 System B, G */
    {0.299, 0.587, 0.114}, /* SMPTE 170M */
    {0.212, 0.701, 0.087}}; /* SMPTE 240M (1987) */

i = matrix_coefficients;
if (i>8)
    i = 3;

cr = coef[i-1][0];
cg = coef[i-1][1];
cb = coef[i-1][2];
cu = 0.5/(1.0-cb);
cv = 0.5/(1.0-cr);

if (chroma_format==CHROMA444)
{
    u444 = frame[1];
    v444 = frame[2];
}
else
{
    if (!u444)
    {
        if (!(u444 = (unsigned char *)malloc(width*height)))
            error("malloc failed");
        if (!(v444 = (unsigned char *)malloc(width*height)))
            error("malloc failed");
        if (chroma_format==CHROMA420)
        {
            if (!(u422 = (unsigned char *)malloc((width>>1)*height)))
                error("malloc failed");
            if (!(v422 = (unsigned char *)malloc((width>>1)*height)))
                error("malloc failed");
        }
    }
}

sprintf(name,"%s.ppm",fname);

if (!(fd = fopen(name,"rb")))
{
    sprintf(errortext,"Couldn't open %s\n",name);
    error(errortext);
}

/* skip header */
getc(fd); getc(fd); /* magic number (P6) */
pbm_getint(fd); pbm_getint(fd); pbm_getint(fd); /* width height maxcolors */

```

```

for (i=0; i<vertical_size; i++)
{
    yp = frame[0] + i*width;
    up = u444 + i*width;
    vp = v444 + i*width;

    for (j=0; j<horizontal_size; j++)
    {
        r=getc(fd); g=getc(fd); b=getc(fd);
        /* convert to YUV */
        y = cr*r + cg*g + cb*b;
        u = cu*(b-y);
        v = cv*(r-y);
        yp[j] = (219.0/256.0)*y + 16.5; /* nominal range: 16..235 */
        up[j] = (224.0/256.0)*u + 128.5; /* 16..240 */
        vp[j] = (224.0/256.0)*v + 128.5; /* 16..240 */
    }
}
fclose(fd);

border_extend(frame[0],horizontal_size,vertical_size,width,height);
border_extend(u444,horizontal_size,vertical_size,width,height);
border_extend(v444,horizontal_size,vertical_size,width,height);

if (chroma_format==CHROMA422)
{
    conv444to422(u444,frame[1]);
    conv444to422(v444,frame[2]);
}

if (chroma_format==CHROMA420)
{
    conv444to422(u444,u422);
    conv444to422(v444,v422);
    conv422to420(u422,frame[1]);
    conv422to420(v422,frame[2]);
}
}

static void border_extend(frame,w1,h1,w2,h2)
unsigned char *frame;
int w1,h1,w2,h2;
{
    int i, j;
    unsigned char *fp;

    /* horizontal pixel replication (right border) */

    for (j=0; j<h1; j++)
    {
        fp = frame + j*w2;
        for (i=w1; i<w2; i++)
            fp[i] = fp[i-1];
    }

    /* vertical pixel replication (bottom border) */

    for (j=h1; j<h2; j++)
    {

```

```

    fp = frame + j*w2;
    for (i=0; i<w2; i++)
        fp[i] = fp[i-w2];
    }
}

/* horizontal filter and 2:1 subsampling */
static void conv444to422(src,dst)
unsigned char *src, *dst;
{
    int i, j, im5, im4, im3, im2, im1, ip1, ip2, ip3, ip4, ip5, ip6;

    if (mpeg1)
    {
        for (j=0; j<height; j++)
        {
            for (i=0; i<width; i+=2)
            {
                im5 = (i<5) ? 0 : i-5;
                im4 = (i<4) ? 0 : i-4;
                im3 = (i<3) ? 0 : i-3;
                im2 = (i<2) ? 0 : i-2;
                im1 = (i<1) ? 0 : i-1;
                ip1 = (i<width-1) ? i+1 : width-1;
                ip2 = (i<width-2) ? i+2 : width-1;
                ip3 = (i<width-3) ? i+3 : width-1;
                ip4 = (i<width-4) ? i+4 : width-1;
                ip5 = (i<width-5) ? i+5 : width-1;
                ip6 = (i<width-5) ? i+6 : width-1;

                /* FIR filter with 0.5 sample interval phase shift */
                dst[i>>1] = clp[(int)(228*(src[i]+src[ip1])
                    +70*(src[im1]+src[ip2])
                    -37*(src[im2]+src[ip3])
                    -21*(src[im3]+src[ip4])
                    +11*(src[im4]+src[ip5])
                    + 5*(src[im5]+src[ip6])+256)>>9];
            }
            src+= width;
            dst+= width>>1;
        }
    }
    else
    {
        /* MPEG-2 */
        for (j=0; j<height; j++)
        {
            for (i=0; i<width; i+=2)
            {
                im5 = (i<5) ? 0 : i-5;
                im3 = (i<3) ? 0 : i-3;
                im1 = (i<1) ? 0 : i-1;
                ip1 = (i<width-1) ? i+1 : width-1;
                ip3 = (i<width-3) ? i+3 : width-1;
                ip5 = (i<width-5) ? i+5 : width-1;

                /* FIR filter coefficients (*512): 22 0 -52 0 159 256 159 0 -52 0 22 */
                dst[i>>1] = clp[(int)( 22*(src[im5]+src[ip5])-52*(src[im3]+src[ip3])
                    +159*(src[im1]+src[ip1])+256*src[i]+256)>>9];
            }
        }
    }
}

```

```

    }
    src+= width;
    dst+= width>>1;
  }
}

/* vertical filter and 2:1 subsampling */
static void conv422to420(src,dst)
unsigned char *src, *dst;
{
  int w, i, j, jm6, jm5, jm4, jm3, jm2, jm1;
  int jp1, jp2, jp3, jp4, jp5, jp6;

  w = width>>1;

  if (prog_frame)
  {
    /* intra frame */
    for (i=0; i<w; i++)
    {
      for (j=0; j<height; j+=2)
      {
        jm5 = (j<5) ? 0 : j-5;
        jm4 = (j<4) ? 0 : j-4;
        jm3 = (j<3) ? 0 : j-3;
        jm2 = (j<2) ? 0 : j-2;
        jm1 = (j<1) ? 0 : j-1;
        jp1 = (j<height-1) ? j+1 : height-1;
        jp2 = (j<height-2) ? j+2 : height-1;
        jp3 = (j<height-3) ? j+3 : height-1;
        jp4 = (j<height-4) ? j+4 : height-1;
        jp5 = (j<height-5) ? j+5 : height-1;
        jp6 = (j<height-5) ? j+6 : height-1;

        /* FIR filter with 0.5 sample interval phase shift */
        dst[w*(j>>1)] = clp[(int)(228*(src[w*j]+src[w*jp1])
          +70*(src[w*jm1]+src[w*jp2])
          -37*(src[w*jm2]+src[w*jp3])
          -21*(src[w*jm3]+src[w*jp4])
          +11*(src[w*jm4]+src[w*jp5])
          + 5*(src[w*jm5]+src[w*jp6])+256)>>9];
      }
      src++;
      dst++;
    }
  }
  else
  {
    /* intra field */
    for (i=0; i<w; i++)
    {
      for (j=0; j<height; j+=4)
      {
        /* top field */
        jm5 = (j<10) ? 0 : j-10;
        jm4 = (j<8) ? 0 : j-8;
        jm3 = (j<6) ? 0 : j-6;
        jm2 = (j<4) ? 0 : j-4;

```

```

jm1 = (j<2) ? 0 : j-2;
jp1 = (j<height-2) ? j+2 : height-2;
jp2 = (j<height-4) ? j+4 : height-2;
jp3 = (j<height-6) ? j+6 : height-2;
jp4 = (j<height-8) ? j+8 : height-2;
jp5 = (j<height-10) ? j+10 : height-2;
jp6 = (j<height-12) ? j+12 : height-2;

/* FIR filter with 0.25 sample interval phase shift */
dst[w*(j>>1)] = clp[(int)(8*src[w*jm5]
    +5*src[w*jm4]
    -30*src[w*jm3]
    -18*src[w*jm2]
    +113*src[w*jm1]
    +242*src[w*j]
    +192*src[w*jp1]
    +35*src[w*jp2]
    -38*src[w*jp3]
    -10*src[w*jp4]
    +11*src[w*jp5]
    +2*src[w*jp6]+256)>>9];

/* bottom field */
jm6 = (j<9) ? 1 : j-9;
jm5 = (j<7) ? 1 : j-7;
jm4 = (j<5) ? 1 : j-5;
jm3 = (j<3) ? 1 : j-3;
jm2 = (j<1) ? 1 : j-1;
jm1 = (j<height-1) ? j+1 : height-1;
jp1 = (j<height-3) ? j+3 : height-1;
jp2 = (j<height-5) ? j+5 : height-1;
jp3 = (j<height-7) ? j+7 : height-1;
jp4 = (j<height-9) ? j+9 : height-1;
jp5 = (j<height-11) ? j+11 : height-1;
jp6 = (j<height-13) ? j+13 : height-1;

/* FIR filter with 0.25 sample interval phase shift */
dst[w*((j>>1)+1)] = clp[(int)(8*src[w*jp6]
    +5*src[w*jp5]
    -30*src[w*jp4]
    -18*src[w*jp3]
    +113*src[w*jp2]
    +242*src[w*jp1]
    +192*src[w*jm1]
    +35*src[w*jm2]
    -38*src[w*jm3]
    -10*src[w*jm4]
    +11*src[w*jm5]
    +2*src[w*jm6]+256)>>9];
}
src++;
dst++;
}
}
}

/* pbm_getc() and pbm_getint() are essentially taken from
 * PBMPLUS (C) Jef Poskanzer
 * but without error/EOF checking

```

```

*/
char pbm_getc(file)
FILE* file;
{
    char ch;

    ch = getc(file);

    if (ch=='#')
    {
        do
        {
            ch = getc(file);
        }
        while (ch!='\n' && ch!='\r');
    }

    return ch;
}

int pbm_getint(file)
FILE* file;
{
    char ch;
    int i;

    do
    {
        ch = pbm_getc(file);
    }
    while (ch==' ' || ch=='\t' || ch=='\n' || ch=='\r');

    i = 0;
    do
    {
        i = i*10 + ch-'0';
        ch = pbm_getc(file);
    }
    while (ch>='0' && ch<='9');

    return i;
}

```

stats.c

/\* stats.c, coding statistics

\*/

/\* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims  
 \* any and all warranties, whether express, implied, or statutory, 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.
*
* The MPEG Software Simulation Group 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.
*
*/

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "global.h"

/* private prototypes */
static void calcSNR1 _ANSI_ARGS_((unsigned char *org, unsigned char *rec,
    int lx, int w, int h, double *pv, double *pe));

void calcSNR(org,rec)
unsigned char *org[3];
unsigned char *rec[3];
{
    int w,h,offs;
    double v,e;

    w = horizontal_size;
    h = (pict_struct==FRAME_PICTURE) ? vertical_size : (vertical_size>>1);
    offs = (pict_struct==BOTTOM_FIELD) ? width : 0;

    calcSNR1(org[0]+offs,rec[0]+offs,width2,w,h,&v,&e);
    fprintf(statfile,"Y: variance=%4.4g, MSE=%3.3g (%3.3g dB), SNR=%3.3g dB\n",
        v, e, 10.0*log10(255.0*255.0/e), 10.0*log10(v/e));

    if (chroma_format!=CHROMA444)
    {
        w >>= 1;
        offs >>= 1;
    }

    if (chroma_format==CHROMA420)
        h >>= 1;

    calcSNR1(org[1]+offs,rec[1]+offs,chrom_width2,w,h,&v,&e);
    fprintf(statfile,"U: variance=%4.4g, MSE=%3.3g (%3.3g dB), SNR=%3.3g dB\n",
        v, e, 10.0*log10(255.0*255.0/e), 10.0*log10(v/e));

    calcSNR1(org[2]+offs,rec[2]+offs,chrom_width2,w,h,&v,&e);
    fprintf(statfile,"V: variance=%4.4g, MSE=%3.3g (%3.3g dB), SNR=%3.3g dB\n",
        v, e, 10.0*log10(255.0*255.0/e), 10.0*log10(v/e));
}

static void calcSNR1(org,rec,lx,w,h,pv,pe)

```

```

unsigned char *org;
unsigned char *rec;
int lx,w,h;
double *pv,*pe;
{
    int i, j;
    double v1, s1, s2, e2;

    s1 = s2 = e2 = 0.0;

    for (j=0; j<h; j++)
    {
        for (i=0; i<w; i++)
        {
            v1 = org[i];
            s1 += v1;
            s2 += v1*v1;
            v1 -= rec[i];
            e2 += v1*v1;
        }
        org += lx;
        rec += lx;
    }

    s1 /= w*h;
    s2 /= w*h;
    e2 /= w*h;

    *pv = s2 - s1*s1; /* variance */
    *pe = e2;        /* MSE */
}

void stats()
{
    int i, j, k, nmb, mb_type;
    int n_skipped, n_intra, n_ncoded, n_blocks, n_interp, n_forward, n_backward;
    struct mbinfo *mbi;

    nmb = mb_width*mb_height2;

    n_skipped=n_intra=n_ncoded=n_blocks=n_interp=n_forward=n_backward=0;

    for (k=0; k<nmb; k++)
    {
        mbi = mbinfo+k;
        if (mbi->skipped)
            n_skipped++;
        else if (mbi->mb_type & MB_INTRA)
            n_intra++;
        else if (!(mbi->mb_type & MB_PATTERN))
            n_ncoded++;

        for (i=0; i<block_count; i++)
            if (mbi->cbp & (1<<i))
                n_blocks++;

        if (mbi->mb_type & MB_FORWARD)
        {
            if (mbi->mb_type & MB_BACKWARD)

```



```

    n_interp++;
else
    n_forward++;
}
else if (mbi->mb_type & MB_BACKWARD)
    n_backward++;
}

fprintf(statfile, "\npicture statistics:\n");
fprintf(statfile, " # of intra coded macroblocks: %4d (%.1f%%)\n",
    n_intra, 100.0*(double)n_intra/nmb);
fprintf(statfile, " # of coded blocks: %4d (%.1f%%)\n",
    n_blocks, 100.0*(double)n_blocks/(block_count*nmb));
fprintf(statfile, " # of not coded macroblocks: %4d (%.1f%%)\n",
    n_ncoded, 100.0*(double)n_ncoded/nmb);
fprintf(statfile, " # of skipped macroblocks: %4d (%.1f%%)\n",
    n_skipped, 100.0*(double)n_skipped/nmb);
fprintf(statfile, " # of forw. pred. macroblocks: %4d (%.1f%%)\n",
    n_forward, 100.0*(double)n_forward/nmb);
fprintf(statfile, " # of backw. pred. macroblocks: %4d (%.1f%%)\n",
    n_backward, 100.0*(double)n_backward/nmb);
fprintf(statfile, " # of interpolated macroblocks: %4d (%.1f%%)\n",
    n_interp, 100.0*(double)n_interp/nmb);

fprintf(statfile, "\nmacroblock_type map:\n");

k = 0;

for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        mbi = mbinfo + k;
        mb_type = mbi->mb_type;
        if (mbi->skipped)
            putc('S', statfile);
        else if (mb_type & MB_INTRA)
            putc('I', statfile);
        else switch (mb_type & (MB_FORWARD|MB_BACKWARD))
        {
            case MB_FORWARD:
                putc(mbi->motion_type==MC_FIELD ? 'f' :
                    mbi->motion_type==MC_DMV ? 'p' :
                    'F', statfile); break;
            case MB_BACKWARD:
                putc(mbi->motion_type==MC_FIELD ? 'b' :
                    'B', statfile); break;
            case MB_FORWARD|MB_BACKWARD:
                putc(mbi->motion_type==MC_FIELD ? 'd' :
                    'D', statfile); break;
            default:
                putc('O', statfile); break;
        }

        if (mb_type & MB_QUANT)
            putc('Q', statfile);
        else if (mb_type & (MB_PATTERN|MB_INTRA))
            putc(' ', statfile);
        else

```

```

        putc('N',statfile);

        putc(' ',statfile);

        k++;
    }
    putc('\n',statfile);
}

fprintf(statfile,"\nmquant map:\n");

k=0;
for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        if (i==0 || mbinfo[k].mquant!=mbinfo[k-1].mquant)
            fprintf(statfile,"%3d",mbinfo[k].mquant);
        else
            fprintf(statfile," ");

        k++;
    }
    putc('\n',statfile);
}

#if 0
fprintf(statfile,"\ncbp map:\n");

k=0;
for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        fprintf(statfile,"%02x ",mbinfo[k].cbp);

        k++;
    }
    putc('\n',statfile);
}

if (pict_struct==FRAME_PICTURE && !frame_pred_dct)
{
    fprintf(statfile,"\ndct_type map:\n");

    k=0;
    for (j=0; j<mb_height2; j++)
    {
        for (i=0; i<mb_width; i++)
        {
            if (mbinfo[k].mb_type & (MB_PATTERN|MB_INTRA))
                fprintf(statfile,"%d ",mbinfo[k].dct_type);
            else
                fprintf(statfile," ");

            k++;
        }
        putc('\n',statfile);
    }
}

```

```

}

if (pict_type!=I_TYPE)
{
    fprintf(statfile, "\nforward motion vectors (first vector, horizontal):\n");

    k=0;
    for (j=0; j<mb_height2; j++)
    {
        for (i=0; i<mb_width; i++)
        {
            if (mbinfo[k].mb_type & MB_FORWARD)
                fprintf(statfile, "%4d", mbinfo[k].MV[0][0][0]);
            else
                fprintf(statfile, " .");

            k++;
        }
        putc('\n', statfile);
    }

    fprintf(statfile, "\nforward motion vectors (first vector, vertical):\n");

    k=0;
    for (j=0; j<mb_height2; j++)
    {
        for (i=0; i<mb_width; i++)
        {
            if (mbinfo[k].mb_type & MB_FORWARD)
                fprintf(statfile, "%4d", mbinfo[k].MV[0][0][1]);
            else
                fprintf(statfile, " .");

            k++;
        }
        putc('\n', statfile);
    }

    fprintf(statfile, "\nforward motion vectors (second vector, horizontal):\n");

    k=0;
    for (j=0; j<mb_height2; j++)
    {
        for (i=0; i<mb_width; i++)
        {
            if (mbinfo[k].mb_type & MB_FORWARD
                && ((pict_struct==FRAME_PICTURE && mbinfo[k].motion_type==MC_FIELD) ||
                    (pict_struct!=FRAME_PICTURE && mbinfo[k].motion_type==MC_16X8)))
                fprintf(statfile, "%4d", mbinfo[k].MV[1][0][0]);
            else
                fprintf(statfile, " .");

            k++;
        }
        putc('\n', statfile);
    }

    fprintf(statfile, "\nforward motion vectors (second vector, vertical):\n");

```

```

k=0;
for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        if (mbinfo[k].mb_type & MB_FORWARD
            && ((pict_struct==FRAME_PICTURE && mbinfo[k].motion_type==MC_FIELD) ||
                (pict_struct!=FRAME_PICTURE && mbinfo[k].motion_type==MC_16X8)))
            fprintf(statfile,"%4d",mbinfo[k].MV[1][0][1]);
        else
            fprintf(statfile," .");

        k++;
    }
    putc('\n',statfile);
}

}

if (pict_type==B_TYPE)
{
    fprintf(statfile,"\nbackward motion vectors (first vector, horizontal):\n");

    k=0;
    for (j=0; j<mb_height2; j++)
    {
        for (i=0; i<mb_width; i++)
        {
            if (mbinfo[k].mb_type & MB_BACKWARD)
                fprintf(statfile,"%4d",mbinfo[k].MV[0][1][0]);
            else
                fprintf(statfile," .");

            k++;
        }
        putc('\n',statfile);
    }

    fprintf(statfile,"\nbackward motion vectors (first vector, vertical):\n");

    k=0;
    for (j=0; j<mb_height2; j++)
    {
        for (i=0; i<mb_width; i++)
        {
            if (mbinfo[k].mb_type & MB_BACKWARD)
                fprintf(statfile,"%4d",mbinfo[k].MV[0][1][1]);
            else
                fprintf(statfile," .");

            k++;
        }
        putc('\n',statfile);
    }

    fprintf(statfile,"\nbackward motion vectors (second vector, horizontal):\n");

    k=0;

```

```

for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        if (mbinfo[k].mb_type & MB_BACKWARD
            && ((pict_struct==FRAME_PICTURE && mbinfo[k].motion_type==MC_FIELD) ||
                (pict_struct!=FRAME_PICTURE && mbinfo[k].motion_type==MC_16X8)))
            fprintf(statfile,"%4d",mbinfo[k].MV[1][1][0]);
        else
            fprintf(statfile," .");

        k++;
    }
    putc('\n',statfile);
}

fprintf(statfile,"\nbackward motion vectors (second vector, vertical):\n");

k=0;
for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        if (mbinfo[k].mb_type & MB_BACKWARD
            && ((pict_struct==FRAME_PICTURE && mbinfo[k].motion_type==MC_FIELD) ||
                (pict_struct!=FRAME_PICTURE && mbinfo[k].motion_type==MC_16X8)))
            fprintf(statfile,"%4d",mbinfo[k].MV[1][1][1]);
        else
            fprintf(statfile," .");

        k++;
    }
    putc('\n',statfile);
}

}
#endif

#if 0
/* useful for debugging */
fprintf(statfile,"\nmacroblock info dump:\n");

k=0;
for (j=0; j<mb_height2; j++)
{
    for (i=0; i<mb_width; i++)
    {
        fprintf(statfile,"%d: %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d\n",
            k,
            mbinfo[k].mb_type,
            mbinfo[k].motion_type,
            mbinfo[k].dct_type,
            mbinfo[k].mquant,
            mbinfo[k].cbp,
            mbinfo[k].skipped,
            mbinfo[k].MV[0][0][0],
            mbinfo[k].MV[0][0][1],
            mbinfo[k].MV[0][1][0],

```

```

        mbinfo[k].MV[0][1][1],
        mbinfo[k].MV[1][0][0],
        mbinfo[k].MV[1][0][1],
        mbinfo[k].MV[1][1][0],
        mbinfo[k].MV[1][1][1],
        mbinfo[k].mv_field_sel[0][0],
        mbinfo[k].mv_field_sel[0][1],
        mbinfo[k].mv_field_sel[1][0],
        mbinfo[k].mv_field_sel[1][1]);

    k++;
}
}
#endif
}

transfrm.c
/* transfrm.c, forward / inverse transformation */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "global.h"

/* private prototypes*/
static void add_pred _ANSI_ARGS__((unsigned char *pred, unsigned char *cur,
    int lx, short *blk));
static void sub_pred _ANSI_ARGS__((unsigned char *pred, unsigned char *cur,
    int lx, short *blk));

/* subtract prediction and transform prediction error */

```

```

void transform(pred,cur,mbi,blocks)
unsigned char *pred[], *cur[];
struct mbinfo *mbi;
short blocks[][64];
{
    int i, j, i1, j1, k, n, cc, offs, lx;

    k = 0;

    for (j=0; j<height2; j+=16)
        for (i=0; i<width; i+=16)
        {
            for (n=0; n<block_count; n++)
            {
                cc = (n<4) ? 0 : (n&1)+1; /* color component index */
                if (cc==0)
                {
                    /* luminance */
                    if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
                    {
                        /* field DCT */
                        offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
                        lx = width<<1;
                    }
                    else
                    {
                        /* frame DCT */
                        offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
                        lx = width2;
                    }

                    if (pict_struct==BOTTOM_FIELD)
                        offs += width;
                }
                else
                {
                    /* chrominance */

                    /* scale coordinates */
                    i1 = (chroma_format==CHROMA444) ? i : i>>1;
                    j1 = (chroma_format!=CHROMA420) ? j : j>>1;

                    if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
                        && (chroma_format!=CHROMA420))
                    {
                        /* field DCT */
                        offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
                        lx = chrom_width<<1;
                    }
                    else
                    {
                        /* frame DCT */
                        offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
                        lx = chrom_width2;
                    }

                    if (pict_struct==BOTTOM_FIELD)
                        offs += chrom_width;
                }
            }
        }
    }

```

```

        sub_pred(pred[cc]+offs,cur[cc]+offs,lx,blocks[k*block_count+n]);
        fdct(blocks[k*block_count+n]);
    }

    k++;
}
}

/* inverse transform prediction error and add prediction */
void itransform(pred,cur,mbi,blocks)
unsigned char *pred[],*cur[];
struct mbinfo *mbi;
short blocks[][64];
{
    int i, j, i1, j1, k, n, cc, offs, lx;

    k = 0;

    for (j=0; j<height2; j+=16)
        for (i=0; i<width; i+=16)
        {
            for (n=0; n<block_count; n++)
            {
                cc = (n<4) ? 0 : (n&1)+1; /* color component index */

                if (cc==0)
                {
                    /* luminance */
                    if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type)
                    {
                        /* field DCT */
                        offs = i + ((n&1)<<3) + width*(j+((n&2)>>1));
                        lx = width<<1;
                    }
                    else
                    {
                        /* frame DCT */
                        offs = i + ((n&1)<<3) + width2*(j+((n&2)<<2));
                        lx = width2;
                    }

                    if (pict_struct==BOTTOM_FIELD)
                        offs += width;
                }
                else
                {
                    /* chrominance */

                    /* scale coordinates */
                    i1 = (chroma_format==CHROMA444) ? i : i>>1;
                    j1 = (chroma_format!=CHROMA420) ? j : j>>1;

                    if ((pict_struct==FRAME_PICTURE) && mbi[k].dct_type
                        && (chroma_format!=CHROMA420))
                    {
                        /* field DCT */
                        offs = i1 + (n&8) + chrom_width*(j1+((n&2)>>1));
                        lx = chrom_width<<1;
                    }
                }
            }
        }
    }

```



```

    }
    else
    {
        /* frame DCT */
        offs = i1 + (n&8) + chrom_width2*(j1+((n&2)<<2));
        lx = chrom_width2;
    }

    if (pict_struct==BOTTOM_FIELD)
        offs += chrom_width;
    }

    idct(blocks[k*block_count+n]);
    add_pred(pred[cc]+offs,cur[cc]+offs,lx,blocks[k*block_count+n]);
}

k++;
}
}

/* add prediction and prediction error, saturate to 0...255 */
static void add_pred(pred,cur,lx,blk)
unsigned char *pred, *cur;
int lx;
short *blk;
{
    int i, j;

    for (j=0; j<8; j++)
    {
        for (i=0; i<8; i++)
            cur[i] = clp[blk[i] + pred[i]];
        blk+= 8;
        cur+= lx;
        pred+= lx;
    }
}

/* subtract prediction from block data */
static void sub_pred(pred,cur,lx,blk)
unsigned char *pred, *cur;
int lx;
short *blk;
{
    int i, j;

    for (j=0; j<8; j++)
    {
        for (i=0; i<8; i++)
            blk[i] = cur[i] - pred[i];
        blk+= 8;
        cur+= lx;
        pred+= lx;
    }
}

/*
 * select between frame and field DCT
 */

```

```

* preliminary version: based on inter-field correlation
*/
void dct_type_estimation(pred,cur,mbi)
unsigned char *pred,*cur;
struct mbinfo *mbi;
{
    short blk0[128], blk1[128];
    int i, j, i0, j0, k, offs, s0, s1, sq0, sq1, s01;
    double d, r;

    k = 0;

    for (j0=0; j0<height2; j0+=16)
        for (i0=0; i0<width; i0+=16)
        {
            if (frame_pred_dct || pict_struct!=FRAME_PICTURE)
                mbi[k].dct_type = 0;
            else
            {
                /* interlaced frame picture */
                /*
                 * calculate prediction error (cur-pred) for top (blk0)
                 * and bottom field (blk1)
                 */
                for (j=0; j<8; j++)
                {
                    offs = width*((j<<1)+j0) + i0;
                    for (i=0; i<16; i++)
                    {
                        blk0[16*j+i] = cur[offs] - pred[offs];
                        blk1[16*j+i] = cur[offs+width] - pred[offs+width];
                        offs++;
                    }
                }
                /* correlate fields */
                s0=s1=sq0=sq1=s01=0;

                for (i=0; i<128; i++)
                {
                    s0+= blk0[i];
                    sq0+= blk0[i]*blk0[i];
                    s1+= blk1[i];
                    sq1+= blk1[i]*blk1[i];
                    s01+= blk0[i]*blk1[i];
                }

                d = (sq0-(s0*s0)/128.0)*(sq1-(s1*s1)/128.0);

                if (d>0.0)
                {
                    r = (s01-(s0*s1)/128.0)/sqrt(d);
                    if (r>0.5)
                        mbi[k].dct_type = 0; /* frame DCT */
                    else
                        mbi[k].dct_type = 1; /* field DCT */
                }
                else
                    mbi[k].dct_type = 1; /* field DCT */
            }
        }
}

```

```

    k++;
}
}

```

vlc.h

/\* vlc.h, variable length code tables (used by routines in putvlc.c) \*/

/\* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims  
 \* any and all warranties, whether express, implied, or statutory, 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.

\*

\* The MPEG Software Simulation Group 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.

\*

\*/

/\* type definitions for variable length code table entries \*/

typedef struct

```

{
    unsigned char code; /* right justified */
    char len;
} VLCtable;

```

/\* for codes longer than 8 bits (excluding leading zeroes) \*/

typedef struct

```

{
    unsigned short code; /* right justified */
    char len;
} sVLCtable;

```

/\* data from ISO/IEC 13818-2 DIS, Annex B, variable length code tables \*/

/\* Table B-1, variable length codes for macroblock\_address\_increment

\*

\* indexed by [macroblock\_address\_increment-1]

\* 'macroblock\_escape' is treated elsewhere

\*/

```

static VLCtable addrinctab[33]=
{
    {0x01,1}, {0x03,3}, {0x02,3}, {0x03,4},
    {0x02,4}, {0x03,5}, {0x02,5}, {0x07,7},
    {0x06,7}, {0x0b,8}, {0x0a,8}, {0x09,8},
    {0x08,8}, {0x07,8}, {0x06,8}, {0x17,10},
    {0x16,10}, {0x15,10}, {0x14,10}, {0x13,10},
    {0x12,10}, {0x23,11}, {0x22,11}, {0x21,11},
    {0x20,11}, {0x1f,11}, {0x1e,11}, {0x1d,11},
    {0x1c,11}, {0x1b,11}, {0x1a,11}, {0x19,11},
    {0x18,11}
};

/* Table B-2, B-3, B-4 variable length codes for macroblock_type
 *
 * indexed by [macroblock_type]
 */

static VLCtable mbtypetab[3][32]=
{
    /* I */
    {
        {0,0}, {1,1}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
        {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
        {0,0}, {1,2}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
        {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}
    },
    /* P */
    {
        {0,0}, {3,5}, {1,2}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
        {1,3}, {0,0}, {1,1}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
        {0,0}, {1,6}, {1,5}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
        {0,0}, {0,0}, {2,5}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}
    },
    /* B */
    {
        {0,0}, {3,5}, {0,0}, {0,0}, {2,3}, {0,0}, {3,3}, {0,0},
        {2,4}, {0,0}, {3,4}, {0,0}, {2,2}, {0,0}, {3,2}, {0,0},
        {0,0}, {1,6}, {0,0}, {0,0}, {0,0}, {0,0}, {2,6}, {0,0},
        {0,0}, {0,0}, {3,6}, {0,0}, {0,0}, {0,0}, {2,5}, {0,0}
    }
};

/* Table B-5 ... B-8 variable length codes for macroblock_type in
 * scalable sequences
 *
 * not implemented
 */

/* Table B-9, variable length codes for coded_block_pattern
 *
 * indexed by [coded_block_pattern]
 */

static VLCtable cbptable[64]=
{
    {0x01,9}, {0x0b,5}, {0x09,5}, {0x0d,6},

```

```

    {0x0d,4}, {0x17,7}, {0x13,7}, {0x1f,8},
    {0x0c,4}, {0x16,7}, {0x12,7}, {0x1e,8},
    {0x13,5}, {0x1b,8}, {0x17,8}, {0x13,8},
    {0x0b,4}, {0x15,7}, {0x11,7}, {0x1d,8},
    {0x11,5}, {0x19,8}, {0x15,8}, {0x11,8},
    {0x0f,6}, {0x0f,8}, {0x0d,8}, {0x03,9},
    {0x0f,5}, {0x0b,8}, {0x07,8}, {0x07,9},
    {0x0a,4}, {0x14,7}, {0x10,7}, {0x1c,8},
    {0x0e,6}, {0x0e,8}, {0x0c,8}, {0x02,9},
    {0x10,5}, {0x18,8}, {0x14,8}, {0x10,8},
    {0x0e,5}, {0x0a,8}, {0x06,8}, {0x06,9},
    {0x12,5}, {0x1a,8}, {0x16,8}, {0x12,8},
    {0x0d,5}, {0x09,8}, {0x05,8}, {0x05,9},
    {0x0c,5}, {0x08,8}, {0x04,8}, {0x04,9},
    {0x07,3}, {0x0a,5}, {0x08,5}, {0x0c,6}
};

```

```

/* Table B-10, variable length codes for motion_code

```

```

*
* indexed by [abs(motion_code)]
* sign of motion_code is treated elsewhere
*/

```

```

static VLCtable motionvectab[17]=

```

```

{
    {0x01,1}, {0x01,2}, {0x01,3}, {0x01,4},
    {0x03,6}, {0x05,7}, {0x04,7}, {0x03,7},
    {0x0b,9}, {0x0a,9}, {0x09,9}, {0x11,10},
    {0x10,10}, {0x0f,10}, {0x0e,10}, {0x0d,10},
    {0x0c,10}
};

```

```

/* Table B-11, variable length codes for dmvector

```

```

*
* treated elsewhere
*/

```

```

/* Table B-12, variable length codes for dct_dc_size_luminance

```

```

*
* indexed by [dct_dc_size_luminance]
*/

```

```

static sVLCtable DClumtab[12]=

```

```

{
    {0x0004,3}, {0x0000,2}, {0x0001,2}, {0x0005,3}, {0x0006,3}, {0x000e,4},
    {0x001e,5}, {0x003e,6}, {0x007e,7}, {0x00fe,8}, {0x01fe,9}, {0x01ff,9}
};

```

```

/* Table B-13, variable length codes for dct_dc_size_chrominance

```

```

*
* indexed by [dct_dc_size_chrominance]
*/

```

```

static sVLCtable DCchromtab[12]=

```

```

{
    {0x0000,2}, {0x0001,2}, {0x0002,2}, {0x0006,3}, {0x000e,4}, {0x001e,5},

```

```
{0x003e,6}, {0x007e,7}, {0x00fe,8}, {0x01fe,9}, {0x03fe,10},{0x03ff,10}
};
```

```
/* Table B-14, DCT coefficients table zero
```

```
*
```

```
* indexed by [run][level-1]
```

```
* split into two tables (dct_code_tab1, dct_code_tab2) to reduce size
```

```
* 'first DCT coefficient' condition and 'End of Block' are treated elsewhere
```

```
* codes do not include s (sign bit)
```

```
*/
```

```
static VLCtable dct_code_tab1[2][40]=
```

```
{
/* run = 0, level = 1...40 */
{
{0x03, 2}, {0x04, 4}, {0x05, 5}, {0x06, 7},
{0x26, 8}, {0x21, 8}, {0x0a,10}, {0x1d,12},
{0x18,12}, {0x13,12}, {0x10,12}, {0x1a,13},
{0x19,13}, {0x18,13}, {0x17,13}, {0x1f,14},
{0x1e,14}, {0x1d,14}, {0x1c,14}, {0x1b,14},
{0x1a,14}, {0x19,14}, {0x18,14}, {0x17,14},
{0x16,14}, {0x15,14}, {0x14,14}, {0x13,14},
{0x12,14}, {0x11,14}, {0x10,14}, {0x18,15},
{0x17,15}, {0x16,15}, {0x15,15}, {0x14,15},
{0x13,15}, {0x12,15}, {0x11,15}, {0x10,15}
},
/* run = 1, level = 1...18 */
{
{0x03, 3}, {0x06, 6}, {0x25, 8}, {0x0c,10},
{0x1b,12}, {0x16,13}, {0x15,13}, {0x1f,15},
{0x1e,15}, {0x1d,15}, {0x1c,15}, {0x1b,15},
{0x1a,15}, {0x19,15}, {0x13,16}, {0x12,16},
{0x11,16}, {0x10,16}, {0x00, 0}, {0x00, 0},
{0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
{0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
{0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
{0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
{0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}
}
};
```

```
static VLCtable dct_code_tab2[30][5]=
```

```
{
/* run = 2...31, level = 1...5 */
{{0x05, 4}, {0x04, 7}, {0x0b,10}, {0x14,12}, {0x14,13}},
{{0x07, 5}, {0x24, 8}, {0x1c,12}, {0x13,13}, {0x00, 0}},
{{0x06, 5}, {0x0f,10}, {0x12,12}, {0x00, 0}, {0x00, 0}},
{{0x07, 6}, {0x09,10}, {0x12,13}, {0x00, 0}, {0x00, 0}},
{{0x05, 6}, {0x1e,12}, {0x14,16}, {0x00, 0}, {0x00, 0}},
{{0x04, 6}, {0x15,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x07, 7}, {0x11,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x05, 7}, {0x11,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x27, 8}, {0x10,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x23, 8}, {0x1a,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x22, 8}, {0x19,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x20, 8}, {0x18,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x0e,10}, {0x17,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x0d,10}, {0x16,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},

```

```

    {{0x08,10}, {0x15,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1f,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1a,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x19,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x17,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x16,12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1f,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1e,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1d,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1c,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1b,13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1f,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1e,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1d,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1c,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
    {{0x1b,16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}}
};

```

```

/* Table B-15, DCT coefficients table one

```

```

*
* indexed by [run][level-1]
* split into two tables (dct_code_tab1a, dct_code_tab2a) to reduce size
* 'End of Block' is treated elsewhere
* codes do not include s (sign bit)
*/

```

```

static VLCtable dct_code_tab1a[2][40]=

```

```

{
/* run = 0, level = 1...40 */
{
    {0x02, 2}, {0x06, 3}, {0x07, 4}, {0x1c, 5},
    {0x1d, 5}, {0x05, 6}, {0x04, 6}, {0x7b, 7},
    {0x7c, 7}, {0x23, 8}, {0x22, 8}, {0xfa, 8},
    {0xfb, 8}, {0xfe, 8}, {0xff, 8}, {0x1f,14},
    {0x1e,14}, {0x1d,14}, {0x1c,14}, {0x1b,14},
    {0x1a,14}, {0x19,14}, {0x18,14}, {0x17,14},
    {0x16,14}, {0x15,14}, {0x14,14}, {0x13,14},
    {0x12,14}, {0x11,14}, {0x10,14}, {0x18,15},
    {0x17,15}, {0x16,15}, {0x15,15}, {0x14,15},
    {0x13,15}, {0x12,15}, {0x11,15}, {0x10,15}
},
/* run = 1, level = 1...18 */
{
    {0x02, 3}, {0x06, 5}, {0x79, 7}, {0x27, 8},
    {0x20, 8}, {0x16,13}, {0x15,13}, {0x1f,15},
    {0x1e,15}, {0x1d,15}, {0x1c,15}, {0x1b,15},
    {0x1a,15}, {0x19,15}, {0x13,16}, {0x12,16},
    {0x11,16}, {0x10,16}, {0x00, 0}, {0x00, 0},
    {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
    {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
    {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0},
    {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}
}
};

```

```

static VLCtable dct_code_tab2a[30][5]=

```

```

{

```

```

/* run = 2...31, level = 1...5 */
{{0x05, 5}, {0x07, 7}, {0xfc, 8}, {0x0c, 10}, {0x14, 13}},
{{0x07, 5}, {0x26, 8}, {0x1c, 12}, {0x13, 13}, {0x00, 0}},
{{0x06, 6}, {0xfd, 8}, {0x12, 12}, {0x00, 0}, {0x00, 0}},
{{0x07, 6}, {0x04, 9}, {0x12, 13}, {0x00, 0}, {0x00, 0}},
{{0x06, 7}, {0x1e, 12}, {0x14, 16}, {0x00, 0}, {0x00, 0}},
{{0x04, 7}, {0x15, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x05, 7}, {0x11, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x78, 7}, {0x11, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x7a, 7}, {0x10, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x21, 8}, {0x1a, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x25, 8}, {0x19, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x24, 8}, {0x18, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x05, 9}, {0x17, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x07, 9}, {0x16, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x0d, 10}, {0x15, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1f, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1a, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x19, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x17, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x16, 12}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1f, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1e, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1d, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1c, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1b, 13}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1f, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1e, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1d, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1c, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}},
{{0x1b, 16}, {0x00, 0}, {0x00, 0}, {0x00, 0}, {0x00, 0}}
};

```

writepic.c

```

/* writepic.c, write reconstructed pictures */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
*
*/

```

```

#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "global.h"

void writeframe(fname,frame)
char *fname;
unsigned char *frame[];
{
    int chrom_hsize, chrom_vsize;
    char name[128];
    FILE *fd;

    chrom_hsize = (chroma_format==CHROMA444) ? horizontal_size
                : horizontal_size>>1;

    chrom_vsize = (chroma_format!=CHROMA420) ? vertical_size
                : vertical_size>>1;

    if (fname[0]!='-')
        return;

    /* Y */
    sprintf(name,"%s.Y",fname);
    if (!(fd = fopen(name,"wb")))
    {
        sprintf(errortext,"Couldn't create %s\n",name);
        error(errortext);
    }
    fwrite(frame[0],1,horizontal_size*vertical_size,fd);
    fclose(fd);

    /* Cb */
    sprintf(name,"%s.U",fname);
    if (!(fd = fopen(name,"wb")))
    {
        sprintf(errortext,"Couldn't create %s\n",name);
        error(errortext);
    }
    fwrite(frame[1],1,chrom_hsize*chrom_vsize,fd);
    fclose(fd);

    /* Cr */
    sprintf(name,"%s.V",fname);
    if (!(fd = fopen(name,"wb")))
    {
        sprintf(errortext,"Couldn't create %s\n",name);
        error(errortext);
    }
    fwrite(frame[2],1,chrom_hsize*chrom_vsize,fd);
    fclose(fd);
}

```

## Decoder

## config.h

```

/* config.h, configuration defines */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

/* define NON_ANSI_COMPILER for compilers without function prototyping */
/* #define NON_ANSI_COMPILER */

#ifndef NON_ANSI_COMPILER
#define _ANSI_ARGS_(x) ()
#else
#define _ANSI_ARGS_(x) x
#endif

#define RB "rb"
#define WB "wb"

#ifndef O_BINARY
#define O_BINARY 0
#endif

```

## display.c

```

/* display.c, X11 interface */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#ifdef DISPLAY

/* the Xlib interface is closely modeled after
 * mpeg_play 2.0 by the Berkeley Plateau Research Group
 */

#include <stdio.h>
#include <stdlib.h>

#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include "config.h"
#include "global.h"

/* private prototypes */
static void display_image _ANSI_ARGS_((XImage *ximage, unsigned char *dithered_image));
static void ditherframe _ANSI_ARGS_((unsigned char *src[]);
static void dithertop _ANSI_ARGS_((unsigned char *src[], unsigned char *dst));
static void ditherbot _ANSI_ARGS_((unsigned char *src[], unsigned char *dst));
static void dithertop420 _ANSI_ARGS_((unsigned char *src[],
                                     unsigned char *dst));
static void ditherbot420 _ANSI_ARGS_((unsigned char *src[],
                                     unsigned char *dst));

/* local data */
static unsigned char *dithered_image, *dithered_image2;

static unsigned char ytab[256+16];
static unsigned char utab[128+16];
static unsigned char vtab[128+16];

/* X11 related variables */

```

```

static Display *display;
static Window window;
static GC gc;
static XImage *ximage, *ximage2;
static unsigned char pixel[256];

#ifdef SH_MEM

#include <sys/ipc.h>
#include <sys/shm.h>
#include <X11/extensions/XShm.h>

static int HandleXError _ANSI_ARGS__((Display *dpy, XErrorEvent *event));
static void InstallXErrorHandler _ANSI_ARGS__((void));
static void DeInstallXErrorHandler _ANSI_ARGS__((void));

static int shmem_flag;
static XShmSegmentInfo shminfo1, shminfo2;
static int gXErrorFlag;
static int CompletionType = -1;

static int HandleXError(dpy, event)
Display *dpy;
XErrorEvent *event;
{
    gXErrorFlag = 1;

    return 0;
}

static void InstallXErrorHandler()
{
    XSetErrorHandler(HandleXError);
    XFlush(display);
}

static void DeInstallXErrorHandler()
{
    XSetErrorHandler(NULL);
    XFlush(display);
}

#endif

/* connect to server, create and map window,
 * allocate colors and (shared) memory
 */
void init_display(name)
char *name;
{
    int crv, cbu, cgu, cgV;
    int y, u, v, r, g, b;
    int i;
    char dummy;
    int screen;
    Colormap cmap;
    int private;
    XColor xcolor;
    unsigned int fg, bg;

```

```

char *hello = "MPEG-2 Display";
XSizeHints hint;
XVisualInfo vinfo;
XEvent xev;
unsigned long tmp_pixel;
XWindowAttributes xwa;

display = XOpenDisplay(name);

if (display == NULL)
    error("Can not open display\n");

screen = DefaultScreen(display);

hint.x = 200;
hint.y = 200;
hint.width = horizontal_size;
hint.height = vertical_size;
hint.flags = PPosition | PSize;

/* Get some colors */

bg = WhitePixel (display, screen);
fg = BlackPixel (display, screen);

/* Make the window */

if (!XMatchVisualInfo(display, screen, 8, PseudoColor, &vinfo))
{
    if (!XMatchVisualInfo(display, screen, 8, GrayScale, &vinfo))
        error("requires 8 bit display\n");
}

window = XCreateSimpleWindow (display, DefaultRootWindow (display),
                              hint.x, hint.y, hint.width, hint.height, 4, fg, bg);

XSelectInput(display, window, StructureNotifyMask);

/* Tell other applications about this window */

XSetStandardProperties (display, window, hello, hello, None, NULL, 0, &hint);

/* Map window. */

XMapWindow(display, window);

/* Wait for map. */
do
{
    XNextEvent(display, &xev);
}
while (xev.type != MapNotify || xev.xmap.event != window);

XSelectInput(display, window, NoEventMask);

/* matrix coefficients */
crv = convmat[matrix_coefficients][0];
cbu = convmat[matrix_coefficients][1];
cgu = convmat[matrix_coefficients][2];

```

```

cgv = convmat[matrix_coefficients][3];

/* allocate colors */

gc = DefaultGC(display, screen);
cmap = DefaultColormap(display, screen);
private = 0;

/* color allocation:
 * i is the (internal) 8 bit color number, it consists of separate
 * bit fields for Y, U and V: i = (yyyyuuvv), we don't use yyyy=0000
 * and yyyy=1111, this leaves 32 colors for other applications
 *
 * the allocated colors correspond to the following Y, U and V values:
 * Y: 24, 40, 56, 72, 88, 104, 120, 136, 152, 168, 184, 200, 216, 232
 * U,V: -48, -16, 16, 48
 *
 * U and V values span only about half the color space; this gives
 * usually much better quality, although highly saturated colors can
 * not be displayed properly
 *
 * translation to R,G,B is implicitly done by the color look-up table
 */
for (i=16; i<240; i++)
{
    /* color space conversion */
    y = 16*((i>>4)&15) + 8;
    u = 32*((i>>2)&3) - 48;
    v = 32*(i&3) - 48;

    y = 76309 * (y - 16); /* (255/219)*65536 */

    r = clp[(y + crv*v + 32768)>>16];
    g = clp[(y - cgu*u - cvg*v + 32768)>>16];
    b = clp[(y + cbu*u + 32786)>>16];

    /* X11 colors are 16 bit */
    xcolor.red = r << 8;
    xcolor.green = g << 8;
    xcolor.blue = b << 8;

    if (XAllocColor(display, cmap, &xcolor) != 0)
        pixel[i] = xcolor.pixel;
    else
    {
        /* allocation failed, have to use a private colormap */

        if (private)
            error("Couldn't allocate private colormap");

        private = 1;

        if (!quiet)
            fprintf(stderr, "Using private colormap (%d colors were available).\n",
                i-16);

        /* Free colors. */
        while (--i >= 16)
        {

```

```

    tmp_pixel = pixel[i]; /* because XFreeColors expects unsigned long */
    XFreeColors(display, cmap, &tmp_pixel, 1, 0);
}

/* i is now 15, this restarts the outer loop */

/* create private colormap */

XGetWindowAttributes(display, window, &xwa);
cmap = XCreateColormap(display, window, xwa.visual, AllocNone);
XSetWindowColormap(display, window, cmap);
}
}

#ifdef SH_MEM
if (XShmQueryExtension(display))
    shmem_flag = 1;
else
{
    shmem_flag = 0;
    if (!quiet)
        fprintf(stderr, "Shared memory not supported\nReverting to normal Xlib\n");
}

if (shmem_flag)
    CompletionType = XShmGetEventBase(display) + ShmCompletion;

InstallXErrorHandler();

if (shmem_flag)
{
    ximage = XShmCreateImage(display, None, 8, ZPixmap, NULL,
                             &shminfo1,
                             coded_picture_width, coded_picture_height);

    if (!prog_seq)
        ximage2 = XShmCreateImage(display, None, 8, ZPixmap, NULL,
                                   &shminfo2,
                                   coded_picture_width, coded_picture_height);

    /* If no go, then revert to normal Xlib calls. */

    if (ximage==NULL || (!prog_seq && ximage2==NULL))
    {
        if (ximage!=NULL)
            XDestroyImage(ximage);
        if (!prog_seq && ximage2!=NULL)
            XDestroyImage(ximage2);
        if (!quiet)
            fprintf(stderr, "Shared memory error, disabling (Ximage error)\n");
        goto shmemerror;
    }

    /* Success here, continue. */

    shminfo1.shmid = shmget(IPC_PRIVATE,
                           ximage->bytes_per_line * ximage->height,
                           IPC_CREAT | 0777);

```

```

if (!prog_seq)
    shminfo2.shmid = shmget(IPC_PRIVATE,
                           ximage2->bytes_per_line * ximage2->height,
                           IPC_CREAT | 0777);

if (shminfo1.shmid<0 || (!prog_seq && shminfo2.shmid<0))
{
    XDestroyImage(ximage);
    if (!prog_seq)
        XDestroyImage(ximage2);
    if (!quiet)
        fprintf(stderr, "Shared memory error, disabling (seg id error)\n");
    goto shmerror;
}

shminfo1.shmaddr = (char *) shmat(shminfo1.shmid, 0, 0);
shminfo2.shmaddr = (char *) shmat(shminfo2.shmid, 0, 0);

if (shminfo1.shmaddr==((char *) -1) ||
    (!prog_seq && shminfo2.shmaddr==((char *) -1)))
{
    XDestroyImage(ximage);
    if (shminfo1.shmaddr!=((char *) -1))
        shmdt(shminfo1.shmaddr);
    if (!prog_seq)
    {
        XDestroyImage(ximage2);
        if (shminfo2.shmaddr!=((char *) -1))
            shmdt(shminfo2.shmaddr);
    }
    if (!quiet)
    {
        fprintf(stderr, "Shared memory error, disabling (address error)\n");
    }
    goto shmerror;
}

ximage->data = shminfo1.shmaddr;
dithered_image = (unsigned char *)ximage->data;
shminfo1.readOnly = False;
XShmAttach(display, &shminfo1);
if (!prog_seq)
{
    ximage2->data = shminfo2.shmaddr;
    dithered_image2 = (unsigned char *)ximage2->data;
    shminfo2.readOnly = False;
    XShmAttach(display, &shminfo2);
}

XSync(display, False);

if (gXErrorFlag)
{
    /* Ultimate failure here. */
    XDestroyImage(ximage);
    shmdt(shminfo1.shmaddr);
    if (!prog_seq)
    {
        XDestroyImage(ximage2);
    }
}

```



```

        shmdt(shminfo2.shmaddr);
    }
    if (!quiet)
        fprintf(stderr, "Shared memory error, disabling.\n");
    gXErrorFlag = 0;
    goto shmerror;
}
else
{
    shmctl(shminfo1.shmid, IPC_RMID, 0);
    if (!prog_seq)
        shmctl(shminfo2.shmid, IPC_RMID, 0);
}

if (!quiet)
{
    fprintf(stderr, "Sharing memory.\n");
}
}
else
{
shmerror:
    shmем_flag = 0;
#endif

    ximage = XCreateImage(display, None, 8, ZPixmap, 0, &dummy,
                          coded_picture_width, coded_picture_height, 8, 0);

    if (!(dithered_image = (unsigned char *)malloc(coded_picture_width*
                                                  coded_picture_height)))
        error("malloc failed");

    if (!prog_seq)
    {
        ximage2 = XCreateImage(display, None, 8, ZPixmap, 0, &dummy,
                              coded_picture_width, coded_picture_height, 8, 0);

        if (!(dithered_image2 = (unsigned char *)malloc(coded_picture_width*
                                                         coded_picture_height)))
            error("malloc failed");
    }

#ifdef SH_MEM
}

DeInstallXErrorHandler();
#endif
}

void exit_display()
{
#ifdef SH_MEM
    if (shmем_flag)
    {
        XShmDetach(display, &shminfo1);
        XDestroyImage(ximage);
        shmdt(shminfo1.shmaddr);
        if (!prog_seq)
        {

```

```

        XShmDetach(display, &shminfo2);
        XDestroyImage(ximage2);
        shmdt(shminfo2.shmaddr);
    }
}
#endif
}

static void display_image(ximage,dithered_image)
XImage *ximage;
unsigned char *dithered_image;
{
    /* display dithered image */
#ifdef SH_MEM
    if (shmem_flag)
    {
        XShmPutImage(display, window, gc, ximage,
                     0, 0, 0, ximage->width, ximage->height, True);
        XFlush(display);

        while (1)
        {
            XEvent xev;

            XNextEvent(display, &xev);
            if (xev.type == CompletionType)
                break;
        }
    }
    else
#endif
    {
        ximage->data = (char *) dithered_image;
        XPutImage(display, window, gc, ximage, 0, 0, 0, ximage->width, ximage->height);
    }
}

void display_second_field()
{
    display_image(ximage2,dithered_image2);
}

/* 4x4 ordered dither
 *
 * threshold pattern:
 *  0  8  2 10
 * 12  4 14  6
 *  3 11  1  9
 * 15  7 13  5
 */

void init_dither()
{
    int i, v;

    for (i=-8; i<256+8; i++)
    {
        v = i>>4;
        if (v<1)

```

```

    v = 1;
    else if (v>14)
        v = 14;
    ytab[i+8] = v<<4;
}

for (i=0; i<128+16; i++)
{
    v = (i-40)>>4;
    if (v<0)
        v = 0;
    else if (v>3)
        v = 3;
    utab[i] = v<<2;
    vtab[i] = v;
}
}

void dither(src)
unsigned char *src[];
{
    if (prog_seq)
        ditherframe(src);
    else
    {
        if ((pict_struct==FRAME_PICTURE && topfirst) || pict_struct==BOTTOM_FIELD)
        {
            /* top field first */
            if (chroma_format==CHROMA420 && hiQdither)
            {
                dithertop420(src,dithered_image);
                ditherbot420(src,dithered_image2);
            }
            else
            {
                dithertop(src,dithered_image);
                ditherbot(src,dithered_image2);
            }
        }
        else
        {
            /* bottom field first */
            if (chroma_format==CHROMA420 && hiQdither)
            {
                ditherbot420(src,dithered_image);
                dithertop420(src,dithered_image2);
            }
            else
            {
                ditherbot(src,dithered_image);
                dithertop(src,dithered_image2);
            }
        }
    }
}

display_image(ximage,dithered_image);
}

static void ditherframe(src)

```

```

unsigned char *src[];
{
    int i,j;
    int y,u,v;
    unsigned char *py,*pu,*pv,*dst;

    py = src[0];
    pu = src[1];
    pv = src[2];
    dst = dithered_image;

    for (j=0; j<coded_picture_height; j+=4)
    {
        /* line j + 0 */
        for (i=0; i<coded_picture_width; i+=4)
        {
            y = *py++;
            u = *pu++ >> 1;
            v = *pv++ >> 1;
            *dst++ = pixel[ytab[y]|utab[u]|vtab[v]];
            y = *py++;
            if (chroma_format==CHROMA444)
            {
                u = *pu++ >> 1;
                v = *pv++ >> 1;
            }
            *dst++ = pixel[ytab[y+8]|utab[u+8]|vtab[v+8]];
            y = *py++;
            u = *pu++ >> 1;
            v = *pv++ >> 1;
            *dst++ = pixel[ytab[y+2]|utab[u+2]|vtab[v+2]];
            y = *py++;
            if (chroma_format==CHROMA444)
            {
                u = *pu++ >> 1;
                v = *pv++ >> 1;
            }
            *dst++ = pixel[ytab[y+10]|utab[u+10]|vtab[v+10]];
        }

        if (chroma_format==CHROMA420)
        {
            pu -= chrom_width;
            pv -= chrom_width;
        }

        /* line j + 1 */
        for (i=0; i<coded_picture_width; i+=4)
        {
            y = *py++;
            u = *pu++ >> 1;
            v = *pv++ >> 1;
            *dst++ = pixel[ytab[y+12]|utab[u+12]|vtab[v+12]];
            y = *py++;
            if (chroma_format==CHROMA444)
            {
                u = *pu++ >> 1;
                v = *pv++ >> 1;
            }
        }
    }
}

```

```

    *dst++ = pixel[ytab[y+4]|utab[u+4]|vtab[v+4]];
    y = *py++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+14]|utab[u+14]|vtab[v+14]];
    y = *py++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+6]|utab[u+6]|vtab[v+6]];
}

/* line j + 2 */
for (i=0; i<coded_picture_width; i+=4)
{
    y = *py++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+3]|utab[u+3]|vtab[v+3]];
    y = *py++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+11]|utab[u+11]|vtab[v+11]];
    y = *py++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+1]|utab[u+1]|vtab[v+1]];
    y = *py++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+9]|utab[u+9]|vtab[v+9]];
}

if (chroma_format==CHROMA420)
{
    pu -= chrom_width;
    pv -= chrom_width;
}

/* line j + 3 */
for (i=0; i<coded_picture_width; i+=4)
{
    y = *py++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+15]|utab[u+15]|vtab[v+15]];
    y = *py++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }

```

```

    }
    *dst++ = pixel[ytab[y+7]|utab[u+7]|vtab[v+7]];
    y = *py++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+13]|utab[u+13]|vtab[v+13]];
    y = *py++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+5]|utab[u+5]|vtab[v+5]];
}
}

static void dithertop(src,dst)
unsigned char *src[];
unsigned char *dst;
{
    int i,j;
    int y,y2,u,v;
    unsigned char *py,*py2,*pu,*pv,*dst2;

    py = src[0];
    py2 = src[0] + (coded_picture_width<<1);
    pu = src[1];
    pv = src[2];
    dst2 = dst + coded_picture_width;

    for (j=0; j<coded_picture_height; j+=4)
    {
        /* line j + 0, j + 1 */
        for (i=0; i<coded_picture_width; i+=4)
        {
            y = *py++;
            y2 = *py2++;
            u = *pu++ >> 1;
            v = *pv++ >> 1;
            *dst++ = pixel[ytab[y]|utab[u]|vtab[v]];
            *dst2++ = pixel[ytab[(y+y2)>>1+12]|utab[u+12]|vtab[v+12]];

            y = *py++;
            y2 = *py2++;
            if (chroma_format==CHROMA444)
            {
                u = *pu++ >> 1;
                v = *pv++ >> 1;
            }
            *dst++ = pixel[ytab[y+8]|utab[u+8]|vtab[v+8]];
            *dst2++ = pixel[ytab[(y+y2)>>1+4]|utab[u+4]|vtab[v+4]];

            y = *py++;
            y2 = *py2++;
            u = *pu++ >> 1;
            v = *pv++ >> 1;
            *dst++ = pixel[ytab[y+2]|utab[u+2]|vtab[v+2]];

```

```

    *dst2++ = pixel[ytab[((y+y2)>>1)+14]|utab[u+14]|vtab[v+14]];

    y = *py++;
    y2 = *py2++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+10]|utab[u+10]|vtab[v+10]];
    *dst2++ = pixel[ytab[((y+y2)>>1)+6]|utab[u+6]|vtab[v+6]];
}

py += coded_picture_width;

if (j!=(coded_picture_height-4))
    py2 += coded_picture_width;
else
    py2 -= coded_picture_width;

dst += coded_picture_width;
dst2 += coded_picture_width;

if (chroma_format==CHROMA420)
{
    pu -= chrom_width;
    pv -= chrom_width;
}
else
{
    pu += chrom_width;
    pv += chrom_width;
}

/* line j + 2, j + 3 */
for (i=0; i<coded_picture_width; i+=4)
{
    y = *py++;
    y2 = *py2++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+3]|utab[u+3]|vtab[v+3]];
    *dst2++ = pixel[ytab[((y+y2)>>1)+15]|utab[u+15]|vtab[v+15]];

    y = *py++;
    y2 = *py2++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+11]|utab[u+11]|vtab[v+11]];
    *dst2++ = pixel[ytab[((y+y2)>>1)+7]|utab[u+7]|vtab[v+7]];

    y = *py++;
    y2 = *py2++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[y+1]|utab[u+1]|vtab[v+1]];

```

```

        *dst2++ = pixel[ytab[((y+y2)>>1)+13]|utab[u+13]|vtab[v+13]];

    y = *py++;
    y2 = *py2++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[y+9]|utab[u+9]|vtab[v+9]];
    *dst2++ = pixel[ytab[((y+y2)>>1)+5]|utab[u+5]|vtab[v+5]];
}

py += coded_picture_width;
py2 += coded_picture_width;
dst += coded_picture_width;
dst2 += coded_picture_width;
pu += chrom_width;
pv += chrom_width;
}
}

static void ditherbot(src,dst)
unsigned char *src[];
unsigned char *dst;
{
    int i,j;
    int y,y2,u,v;
    unsigned char *py,*py2,*pu,*pv,*dst2;

    py = src[0] + coded_picture_width;
    py2 = py;
    pu = src[1] + chrom_width;
    pv = src[2] + chrom_width;
    dst2 = dst + coded_picture_width;

    for (j=0; j<coded_picture_height; j+=4)
    {
        /* line j + 0, j + 1 */
        for (i=0; i<coded_picture_width; i+=4)
        {
            y = *py++;
            y2 = *py2++;
            u = *pu++ >> 1;
            v = *pv++ >> 1;
            *dst++ = pixel[ytab[((y+y2)>>1)]|utab[u]|vtab[v]];
            *dst2++ = pixel[ytab[y2+12]|utab[u+12]|vtab[v+12]];

            y = *py++;
            y2 = *py2++;
            if (chroma_format==CHROMA444)
            {
                u = *pu++ >> 1;
                v = *pv++ >> 1;
            }
            *dst++ = pixel[ytab[((y+y2)>>1)+8]|utab[u+8]|vtab[v+8]];
            *dst2++ = pixel[ytab[y2+4]|utab[u+4]|vtab[v+4]];

            y = *py++;

```



```

    y2 = *py2++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[((y+y2)>>1)+2]|utab[u+2]|vtab[v+2]];
    *dst2++ = pixel[ytab[y2+14]|utab[u+14]|vtab[v+14]];

    y = *py++;
    y2 = *py2++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[((y+y2)>>1)+10]|utab[u+10]|vtab[v+10]];
    *dst2++ = pixel[ytab[y2+6]|utab[u+6]|vtab[v+6]];
}

if (j==0)
    py -= coded_picture_width;
else
    py += coded_picture_width;

py2 += coded_picture_width;
dst += coded_picture_width;
dst2 += coded_picture_width;

if (chroma_format==CHROMA420)
{
    pu -= chrom_width;
    pv -= chrom_width;
}
else
{
    pu += chrom_width;
    pv += chrom_width;
}

/* line j + 2. j + 3 */
for (i=0; i<coded_picture_width; i+=4)
{
    y = *py++;
    y2 = *py2++;
    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[((y+y2)>>1)+3]|utab[u+3]|vtab[v+3]];
    *dst2++ = pixel[ytab[y2+15]|utab[u+15]|vtab[v+15]];

    y = *py++;
    y2 = *py2++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[((y+y2)>>1)+11]|utab[u+11]|vtab[v+11]];
    *dst2++ = pixel[ytab[y2+7]|utab[u+7]|vtab[v+7]];

    y = *py++;
    y2 = *py2++;

```

```

    u = *pu++ >> 1;
    v = *pv++ >> 1;
    *dst++ = pixel[ytab[((y+y2)>>1)+1]|utab[u+1]|vtab[v+1]];
    *dst2++ = pixel[ytab[y2+13]|utab[u+13]|vtab[v+13]];

    y = *py++;
    y2 = *py2++;
    if (chroma_format==CHROMA444)
    {
        u = *pu++ >> 1;
        v = *pv++ >> 1;
    }
    *dst++ = pixel[ytab[((y+y2)>>1)+9]|utab[u+9]|vtab[v+9]];
    *dst2++ = pixel[ytab[y2+5]|utab[u+5]|vtab[v+5]];
}

py += coded_picture_width;
py2 += coded_picture_width;
dst += coded_picture_width;
dst2 += coded_picture_width;
pu += chrom_width;
pv += chrom_width;
}
}

static void dithertop420(src,dst)
unsigned char *src[];
unsigned char *dst;
{
    int i,j;
    int y1,u1,v1,y2,u2,v2;
    unsigned char *py1,*pu1,*pv1,*py2,*pu2,*pv2,*dst2;

    py1 = src[0];
    pu1 = src[1];
    pv1 = src[2];

    py2 = py1 + (coded_picture_width<<1);
    pu2 = pu1 + (chrom_width<<1);
    pv2 = pv1 + (chrom_width<<1);

    dst2 = dst + coded_picture_width;

    for (j=0; j<coded_picture_height; j+=4)
    {
        /* line j + 0, j + 1 */
        for (i=0; i<coded_picture_width; i+=4)
        {
            y1 = *py1++;
            y2 = *py2++;
            u1 = *pu1++ >> 1;
            v1 = *pv1++ >> 1;
            u2 = *pu2++ >> 1;
            v2 = *pv2++ >> 1;
            *dst++ = pixel[ytab[((3*y1+y2)>>2)]|utab[u1]|vtab[v1]];
            *dst2++ = pixel[ytab[((y1+3*y2)>>2)+12]|utab[((3*u1+u2)>>2)+12]|
                vtab[((3*v1+v2)>>2)+12]];

            y1 = *py1++;

```

```

y2 = *py2++;
*dst++ = pixel[ytab[(((3*y1+y2)>>2)+8)|utab[u1+8]|vtab[v1+8]]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+4]|utab[(((3*u1+u2)>>2)+4)
|vtab[(((3*v1+v2)>>2)+4)]];

y1 = *py1++;
y2 = *py2++;
u1 = *pu1++ >> 1;
v1 = *pv1++ >> 1;
u2 = *pu2++ >> 1;
v2 = *pv2++ >> 1;
*dst++ = pixel[ytab[(((3*y1+y2)>>2)+2)|utab[u1+2]|vtab[v1+2]]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+14]|utab[(((3*u1+u2)>>2)+14)
|vtab[(((3*v1+v2)>>2)+14)]];

y1 = *py1++;
y2 = *py2++;
*dst++ = pixel[ytab[(((3*y1+y2)>>2)+10)|utab[u1+10]|vtab[v1+10]]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+6]|utab[(((3*u1+u2)>>2)+6)
|vtab[(((3*v1+v2)>>2)+6)]];
}

py1 += coded_picture_width;

if (j!=(coded_picture_height-4))
    py2 += coded_picture_width;
else
    py2 -= coded_picture_width;

pu1 -= chrom_width;
pv1 -= chrom_width;
pu2 -= chrom_width;
pv2 -= chrom_width;

dst += coded_picture_width;
dst2 += coded_picture_width;

/* line j + 2, j + 3 */
for (i=0; i<coded_picture_width; i+=4)
{
    y1 = *py1++;
    y2 = *py2++;
    u1 = *pu1++ >> 1;
    v1 = *pv1++ >> 1;
    u2 = *pu2++ >> 1;
    v2 = *pv2++ >> 1;
    *dst++ = pixel[ytab[(((3*y1+y2)>>2)+3)|utab[(((u1+u2)>>1)+3)
|vtab[(((v1+v2)>>1)+3)]];
    *dst2++ = pixel[ytab[((y1+3*y2)>>2)+15]|utab[(((u1+3*u2)>>2)+15)
|vtab[(((v1+3*v2)>>2)+15)]];

    y1 = *py1++;
    y2 = *py2++;
    *dst++ = pixel[ytab[(((3*y1+y2)>>2)+11)|utab[(((u1+u2)>>1)+11)
|vtab[(((v1+v2)>>1)+11)]];
    *dst2++ = pixel[ytab[((y1+3*y2)>>2)+7]|utab[(((u1+3*u2)>>2)+7)
|vtab[(((v1+3*v2)>>2)+7)]];

    y1 = *py1++;

```

```

    y2 = *py2++;
    u1 = *pu1++ >> 1;
    v1 = *pv1++ >> 1;
    u2 = *pu2++ >> 1;
    v2 = *pv2++ >> 1;
    *dst++ = pixel[ytab[((3*y1+y2)>>2)+1]|utab[((u1+u2)>>1)+1]
               |vtab[((v1+v2)>>1)+1]];
    *dst2++ = pixel[ytab[((y1+3*y2)>>2)+13]|utab[((u1+3*u2)>>2)+13]
               |vtab[((v1+3*v2)>>2)+13]];

    y1 = *py1++;
    y2 = *py2++;
    *dst++ = pixel[ytab[((3*y1+y2)>>2)+9]|utab[((u1+u2)>>1)+9]
               |vtab[((v1+v2)>>1)+9]];
    *dst2++ = pixel[ytab[((y1+3*y2)>>2)+5]|utab[((u1+3*u2)>>2)+5]
               |vtab[((v1+3*v2)>>2)+5]];
}

py1 += coded_picture_width;
py2 += coded_picture_width;
pu1 += chrom_width;
pv1 += chrom_width;
if (j!=(coded_picture_height-8))
{
    pu2 += chrom_width;
    pv2 += chrom_width;
}
else
{
    pu2 -= chrom_width;
    pv2 -= chrom_width;
}
dst += coded_picture_width;
dst2 += coded_picture_width;
}
}

static void ditherbot420(src,dst)
unsigned char *src[];
unsigned char *dst;
{
    int i,j;
    int y1,u1,v1,y2,u2,v2;
    unsigned char *py1,*pu1,*pv1,*py2,*pu2,*pv2,*dst2;

    py2 = py1 = src[0] + coded_picture_width;
    pu2 = pu1 = src[1] + chrom_width;
    pv2 = pv1 = src[2] + chrom_width;

    dst2 = dst;

    for (j=0; j<coded_picture_height; j+=4)
    {
        /* line j + 0, j + 1 */
        for (i=0; i<coded_picture_width; i+=4)
        {
            y1 = *py1++;
            y2 = *py2++;
            u1 = *pu1++ >> 1;

```

```

v1 = *pv1++ >> 1;
u2 = *pu2++ >> 1;
v2 = *pv2++ >> 1;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+15]|utab[((3*u1+u2)>>2)+15]
          |vtab[((3*v1+v2)>>2)+15]];
*dst2++ = pixel[ytab[(y1+3*y2)>>2]|utab[(u1+u2)>>1]
          |vtab[(v1+v2)>>1]]];

y1 = *py1++;
y2 = *py2++;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+7]|utab[((3*u1+u2)>>2)+7]
          |vtab[((3*v1+v2)>>2)+7]];
*dst2++ = pixel[ytab[(y1+3*y2)>>2)+8]|utab[(u1+u2)>>1)+8]
          |vtab[(v1+v2)>>1)+8]];

y1 = *py1++;
y2 = *py2++;
u1 = *pu1++ >> 1;
v1 = *pv1++ >> 1;
u2 = *pu2++ >> 1;
v2 = *pv2++ >> 1;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+13]|utab[((3*u1+u2)>>2)+13]
          |vtab[((3*v1+v2)>>2)+13]];
*dst2++ = pixel[ytab[(y1+3*y2)>>2)+2]|utab[(u1+u2)>>1)+2]
          |vtab[(v1+v2)>>1)+2]];

y1 = *py1++;
y2 = *py2++;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+5]|utab[((3*u1+u2)>>2)+5]
          |vtab[((3*v1+v2)>>2)+5]];
*dst2++ = pixel[ytab[(y1+3*y2)>>2)+10]|utab[(u1+u2)>>1)+10]
          |vtab[(v1+v2)>>1)+10]];
}

if (j!=0)
    py1 += coded_picture_width;
else
    py1 -= coded_picture_width;

py2 += coded_picture_width;

pu1 -= chrom_width;
pv1 -= chrom_width;
pu2 -= chrom_width;
pv2 -= chrom_width;

if (j!=0)
    dst += coded_picture_width;

dst2 += coded_picture_width;

/* line j + 2, j + 3 */
for (i=0; i<coded_picture_width; i+=4)
{
    y1 = *py1++;
    y2 = *py2++;
    u1 = *pu1++ >> 1;
    v1 = *pv1++ >> 1;
    u2 = *pu2++ >> 1;

```

```

v2 = *pv2++ >> 1;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+12]|utab[((u1+3*u2)>>2)+12]
          |vtab[((v1+3*v2)>>2)+12]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+3]|utab[u2+3]
          |vtab[v2+3]];

y1 = *py1++;
y2 = *py2++;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+4]|utab[((u1+3*u2)>>2)+4]
          |vtab[((v1+3*v2)>>2)+4]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+11]|utab[u2+11]
          |vtab[v2+11]];

y1 = *py1++;
y2 = *py2++;
u1 = *pu1++ >> 1;
v1 = *pv1++ >> 1;
u2 = *pu2++ >> 1;
v2 = *pv2++ >> 1;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+14]|utab[((u1+3*u2)>>2)+14]
          |vtab[((v1+3*v2)>>2)+14]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+1]|utab[u2+1]
          |vtab[v2+1]];

y1 = *py1++;
y2 = *py2++;
*dst++ = pixel[ytab[((3*y1+y2)>>2)+6]|utab[((u1+3*u2)>>2)+6]
          |vtab[((v1+3*v2)>>2)+6]];
*dst2++ = pixel[ytab[((y1+3*y2)>>2)+9]|utab[u2+9]
          |vtab[v2+9]];
}

py1 += coded_picture_width;
py2 += coded_picture_width;

if (j!=0)
{
    pu1 += chrom_width;
    pv1 += chrom_width;
}
else
{
    pu1 -= chrom_width;
    pv1 -= chrom_width;
}

pu2 += chrom_width;
pv2 += chrom_width;

dst += coded_picture_width;
dst2 += coded_picture_width;
}

py2 -= (coded_picture_width<<1);
pu2 -= (chrom_width<<1);
pv2 -= (chrom_width<<1);

/* dither last line */
for (i=0; i<coded_picture_width; i+=4)

```

```

{
    y1 = *py1++;
    y2 = *py2++;
    u1 = *pu1++ >> 1;
    v1 = *pv1++ >> 1;
    u2 = *pu2++ >> 1;
    v2 = *pv2++ >> 1;
    *dst++ = pixel[ytab[((3*y1+y2)>>2)+15]|utab[((3*u1+u2)>>2)+15]
                |vtab[((3*v1+v2)>>2)+15]];

    y1 = *py1++;
    y2 = *py2++;
    *dst++ = pixel[ytab[((3*y1+y2)>>2)+7]|utab[((3*u1+u2)>>2)+7]
                |vtab[((3*v1+v2)>>2)+7]];

    y1 = *py1++;
    y2 = *py2++;
    u1 = *pu1++ >> 1;
    v1 = *pv1++ >> 1;
    u2 = *pu2++ >> 1;
    v2 = *pv2++ >> 1;
    *dst++ = pixel[ytab[((3*y1+y2)>>2)+13]|utab[((3*u1+u2)>>2)+13]
                |vtab[((3*v1+v2)>>2)+13]];

    y1 = *py1++;
    y2 = *py2++;
    *dst++ = pixel[ytab[((3*y1+y2)>>2)+5]|utab[((3*u1+u2)>>2)+5]
                |vtab[((3*v1+v2)>>2)+5]];
}

}
#endif

```

getbits.c

```

/* getbits.c, bit level routines */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.  
 \*  
 \*/

```
#include <stdlib.h>
```

```
#include "config.h"
```

```
#include "global.h"
```

```
/* to mask the n least significant bits of an integer */
```

```
static unsigned int msk[33] =
{
  0x00000000,0x00000001,0x00000003,0x00000007,
  0x0000000f,0x0000001f,0x0000003f,0x0000007f,
  0x000000ff,0x000001ff,0x000003ff,0x000007ff,
  0x00000fff,0x00001fff,0x00003fff,0x00007fff,
  0x0000ffff,0x0001ffff,0x0003ffff,0x0007ffff,
  0x000fffff,0x001fffff,0x003fffff,0x007fffff,
  0x00ffffff,0x01ffffff,0x03ffffff,0x07ffffff,
  0x0fffffff,0x1fffffff,0x3fffffff,0x7fffffff,
  0xffffffff
};
```

```
/* initialize buffer, call once before first getbits or showbits */
```

```
void initbits()
{
  ld->incnt = 0;
  ld->rdptr = ld->rdbfr + 2048;
  ld->bitcnt = 0;
}
```

```
void fillbfr()
{
  int l;

  ld->inbfr[0] = ld->inbfr[8];
  ld->inbfr[1] = ld->inbfr[9];
  ld->inbfr[2] = ld->inbfr[10];
  ld->inbfr[3] = ld->inbfr[11];

  if (ld->rdptr >= ld->rdbfr + 2048)
  {
    l = read(ld->infile, ld->rdbfr, 2048);
    ld->rdptr = ld->rdbfr;
    if (l < 2048)
    {
      if (l < 0)
        l = 0;

      while (l & 3)
        ld->rdbfr[l++] = 0;
    }
  }
}
```



```

    while (l<2048)
    {
        ld->rdbfr[l++] = SEQ_END_CODE>>24;
        ld->rdbfr[l++] = SEQ_END_CODE>>16;
        ld->rdbfr[l++] = SEQ_END_CODE>>8;
        ld->rdbfr[l++] = SEQ_END_CODE&0xff;
    }
}

for (l=0; l<8; l++)
    ld->inbfr[l+4] = ld->rdptr[l];

ld->rdptr+= 8;
ld->incnt+= 64;
}

/* return next n bits (right adjusted) without advancing */

unsigned int showbits(n)
int n;
{
    unsigned char *v;
    unsigned int b;
    int c;

    if (ld->incnt<n)
        fillbfr();

    v = ld->inbfr + ((96 - ld->incnt)>>3);
    b = (v[0]<<24) | (v[1]<<16) | (v[2]<<8) | v[3];
    c = ((ld->incnt-1) & 7) + 25;
    return (b>>(c-n)) & msk[n];
}

/* return next bit (could be made faster than getbits(1)) */

unsigned int getbits1()
{
    return getbits(1);
}

/* advance by n bits */

void flushbits(n)
int n;
{
    ld->bitcnt+= n;
    ld->incnt-= n;
    if (ld->incnt < 0)
        fillbfr();
}

/* return next n bits (right adjusted) */

```

```

unsigned int getbits(n)
int n;
{
    unsigned int l;

    l = showbits(n);
    flushbits(n);

    return l;
}

```

getblk.c

```

/* getblk.c, DCT block decoding */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>

#include "config.h"
#include "global.h"

/* defined in getvlc.h */
typedef struct {
    char run, level, len;
} DCTtab;

extern DCTtab DCTtabfirst[],DCTtabnext[],DCTtab0[],DCTtab1[];
extern DCTtab DCTtab2[],DCTtab3[],DCTtab4[],DCTtab5[],DCTtab6[];
extern DCTtab DCTtab0a[],DCTtab1a[];

```

```

/* decode one intra coded MPEG-1 block */

void getintrablock(comp,dc_dct_pred)
int comp;
int dc_dct_pred[];
{
    int val, i, j, sign;
    unsigned int code;
    int tval;
    DCTtab *tab;
    short *bp;

    bp = ld->block[comp];
    /* decode DC coefficients */
    if (comp<4)
        bp[0] = (dc_dct_pred[0]+=getDClum()) << 3;
    else if (comp==4)
        bp[0] = (dc_dct_pred[1]+=getDCchrom()) << 3;
    else
        bp[0] = (dc_dct_pred[2]+=getDCchrom()) << 3;

    if (fault) return;

    /* decode AC coefficients */
    for (i=1; ; i++)
    {
        code = showbits(16);
        if (code>=16384)
            tab = &DCTtabnext[(code>>12)-4];
        else if (code>=1024)
            tab = &DCTtab0[(code>>8)-4];
        else if (code>=512)
            tab = &DCTtab1[(code>>6)-8];
        else if (code>=256)
            tab = &DCTtab2[(code>>4)-16];
        else if (code>=128)
            tab = &DCTtab3[(code>>3)-16];
        else if (code>=64)
            tab = &DCTtab4[(code>>2)-16];
        else if (code>=32)
            tab = &DCTtab5[(code>>1)-16];
        else if (code>=16)
            tab = &DCTtab6[code-16];
        else
        {
            if (!quiet)
                fprintf(stderr,"invalid Huffman code in getintrablock()\n");
            fault = 1;
            return;
        }

        flushbits(tab->len);

        if (tab->run==64) /* end_of_block */
            return;

        if (tab->run==65) /* escape */

```

```

{
    i+= getbits(6);

    val = getbits(8);
    if (val==0)
        val = getbits(8);
    else if (val==128)
        val = getbits(8) - 256;
    else if (val>128)
        val -= 256;

    if (sign = (val<0))
        val = -val;
}
else
{
    i+= tab->run;
    val = tab->level;
    sign = getbits(1);
}

if (i>=64)
{
    if (!quiet)
        fprintf(stderr,"DCT coeff index (i) out of bounds (intra)\n");
    fault = 1;
    return;
}

j = zig_zag_scan[i];
val = (val*ld->quant_scale*ld->intra_quantizer_matrix[j]) >> 3;

/* mismatch control ('oddification') */
if (val!=0) /* should always be true, but it's not guaranteed */
    val = (val-1) | 1; /* equivalent to: if ((val&1)==0) val = val - 1; */

/* saturation */
if (!sign)
    bp[j] = (val>2047) ? 2047 : val; /* positive */
else
    bp[j] = (val>2048) ? -2048 : -val; /* negative */
}
}

/* decode one non-intra coded MPEG-1 block */

void getinterblock(comp)
int comp;
{
    int val, i, j, sign;
    unsigned int code;
    int tval;
    DCTtab *tab;
    short *bp;

    bp = ld->block[comp];

    /* decode AC coefficients */

```

```

for (i=0; ; i++)
{
    code = showbits(16);
    if (code>=16384)
    {
        if (i==0)
            tab = &DCTtabfirst[(code>>12)-4];
        else
            tab = &DCTtabnext[(code>>12)-4];
    }
    else if (code>=1024)
        tab = &DCTtab0[(code>>8)-4];
    else if (code>=512)
        tab = &DCTtab1[(code>>6)-8];
    else if (code>=256)
        tab = &DCTtab2[(code>>4)-16];
    else if (code>=128)
        tab = &DCTtab3[(code>>3)-16];
    else if (code>=64)
        tab = &DCTtab4[(code>>2)-16];
    else if (code>=32)
        tab = &DCTtab5[(code>>1)-16];
    else if (code>=16)
        tab = &DCTtab6[code-16];
    else
    {
        if (!quiet)
            fprintf(stderr,"invalid Huffman code in getinterblock()\n");
        fault = 1;
        return;
    }

    flushbits(tab->len);

    if (tab->run==64) /* end_of_block */
        return;

    if (tab->run==65) /* escape */
    {
        i+= getbits(6);

        val = getbits(8);
        if (val==0)
            val = getbits(8);
        else if (val==128)
            val = getbits(8) - 256;
        else if (val>128)
            val -= 256;

        if (sign = (val<0))
            val = -val;
    }
    else
    {
        i+= tab->run;
        val = tab->level;
        sign = getbits(1);
    }
}

```

```

    if (i>=64)
    {
        if (!quiet)
            fprintf(stderr, "DCT coeff index (i) out of bounds (inter)\n");
        fault = 1;
        return;
    }

    j = zig_zag_scan[i];
    val = (((val<<1)+1)*ld->quant_scale*ld->non_intra_quantizer_matrix[j]) >> 4;

    /* mismatch control ('oddification') */
    if (val!=0) /* should always be true, but it's not guaranteed */
        val = (val-1) | 1; /* equivalent to: if ((val&1)==0) val = val - 1; */

    /* saturation */
    if (!sign)
        bp[j] = (val>2047) ? 2047 : val; /* positive */
    else
        bp[j] = (val>2048) ? -2048 : -val; /* negative */
    }
}

/* decode one intra coded MPEG-2 block */

void getmpg2inblock(comp,dc_dct_pred)
int comp;
int dc_dct_pred[];
{
    int val, i, j, sign, nc, cc, run;
    unsigned int code;
    DCTtab *tab;
    short *bp;
    int *qmat;
    struct layer_data *ld1;

    /* with data partitioning, data always goes to base layer */
    ld1 = (ld->scalable_mode==SC_DP) ? &base : ld;
    bp = ld1->block[comp];

    if (base.scalable_mode==SC_DP)
        if (base.pri_brk<64)
            ld = &enhan;
        else
            ld = &base;

    cc = (comp<4) ? 0 : (comp&1)+1;

    qmat = (comp<4 || chroma_format==CHROMA420)
        ? ld1->intra_quantizer_matrix
        : ld1->chroma_intra_quantizer_matrix;

    /* decode DC coefficients */
    if (cc==0)
        val = (dc_dct_pred[0]+= getDClum());
    else if (cc==1)
        val = (dc_dct_pred[1]+= getDCchrom());
    else

```

```

    val = (dc_dct_pred[2] += getDCchrom());

    if (fault) return;

    bp[0] = val << (3-dc_prec);

    nc=0;

    if (trace)
        printf("DCT(%d)i:",comp);

    /* decode AC coefficients */
    for (i=1; ; i++)
    {
        code = showbits(16);
        if (code>=16384 && !intravlc)
            tab = &DCTtabnext[(code>>12)-4];
        else if (code>=1024)
        {
            if (intravlc)
                tab = &DCTtab0a[(code>>8)-4];
            else
                tab = &DCTtab0[(code>>8)-4];
        }
        else if (code>=512)
        {
            if (intravlc)
                tab = &DCTtab1a[(code>>6)-8];
            else
                tab = &DCTtab1[(code>>6)-8];
        }
        else if (code>=256)
            tab = &DCTtab2[(code>>4)-16];
        else if (code>=128)
            tab = &DCTtab3[(code>>3)-16];
        else if (code>=64)
            tab = &DCTtab4[(code>>2)-16];
        else if (code>=32)
            tab = &DCTtab5[(code>>1)-16];
        else if (code>=16)
            tab = &DCTtab6[code-16];
        else
        {
            if (!quiet)
                fprintf(stderr,"invalid Huffman code in getmpg2intrablock()\n");
            fault = 1;
            return;
        }

        flushbits(tab->len);

        if (trace)
        {
            printf(" ");
            printbits(code,16,tab->len);
        }

        if (tab->run==64) /* end_of_block */
        {

```

```

    if (trace)
        printf("): EOB\n");
    return;
}

if (tab->run==65) /* escape */
{
    if (trace)
    {
        putchar(' ');
        printbits(showbits(6),6,6);
    }

    i+= run = getbits(6);

    if (trace)
    {
        putchar(' ');
        printbits(showbits(12),12,12);
    }

    val = getbits(12);
    if ((val&2047)==0)
    {
        if (!quiet)
            fprintf(stderr,"invalid signed_level (escape) in getmpg2intrablock()\n");
        fault = 1;
        return;
    }
    if (sign = (val>=2048))
        val = 4096 - val;
}
else
{
    i+= run = tab->run;
    val = tab->level;
    sign = getbits(1);

    if (trace)
        printf("%d",sign);
}

if (i>=64)
{
    if (!quiet)
        fprintf(stderr,"DCT coeff index (i) out of bounds (intra2)\n");
    fault = 1;
    return;
}

if (trace)
    printf("): %d/%d",run,sign ? -val : val);

j = (ld1->altscan ? alternate_scan : zig_zag_scan)[i];
val = (val * ld1->quant_scale * qmat[j]) >> 4;
bp[j] = sign ? -val : val;
nc++;

if (base.scalable_mode==SC_DP && nc==base.pri_brk-63)

```



```

    ld = &enhan;
}
}

/* decode one non-intra coded MPEG-2 block */

void getmpg2interblock(comp)
int comp;
{
    int val, i, j, sign, nc, run;
    unsigned int code;
    DCTtab *tab;
    short *bp;
    int *qmat;
    struct layer_data *ld1;

    /* with data partitioning, data always goes to base layer */
    ld1 = (ld->scalable_mode==SC_DP) ? &base : ld;
    bp = ld1->block[comp];

    if (base.scalable_mode==SC_DP)
        if (base.pri_brk<64)
            ld = &enhan;
        else
            ld = &base;

    qmat = (comp<4 || chroma_format==CHROMA420)
        ? ld1->non_intra_quantizer_matrix
        : ld1->chroma_non_intra_quantizer_matrix;

    nc = 0;

    if (trace)
        printf("DCT(%d)n:",comp);

    /* decode AC coefficients */
    for (i=0; ; i++)
    {
        code = showbits(16);
        if (code>=16384)
        {
            if (i==0)
                tab = &DCTtabfirst[(code>>12)-4];
            else
                tab = &DCTtabnext[(code>>12)-4];
        }
        else if (code>=1024)
            tab = &DCTtab0[(code>>8)-4];
        else if (code>=512)
            tab = &DCTtab1[(code>>6)-8];
        else if (code>=256)
            tab = &DCTtab2[(code>>4)-16];
        else if (code>=128)
            tab = &DCTtab3[(code>>3)-16];
        else if (code>=64)
            tab = &DCTtab4[(code>>2)-16];
        else if (code>=32)
            tab = &DCTtab5[(code>>1)-16];

```

```

else if (code>=16)
    tab = &DCTtab6[code-16];
else
{
    if (!quiet)
        fprintf(stderr,"invalid Huffman code in getmpg2interblock()\n");
    fault = 1;
    return;
}

flushbits(tab->len);

if (trace)
{
    printf(" ");
    printbits(code,16,tab->len);
}

if (tab->run==64) /* end_of_block */
{
    if (trace)
        printf("): EOB\n");
    return;
}

if (tab->run==65) /* escape */
{
    if (trace)
    {
        putchar(' ');
        printbits(showbits(6),6,6);
    }

    i+= run = getbits(6);

    if (trace)
    {
        putchar(' ');
        printbits(showbits(12),12,12);
    }

    val = getbits(12);
    if ((val&2047)==0)
    {
        if (!quiet)
            fprintf(stderr,"invalid signed_level (escape) in getmpg2intrablock()\n");
        fault = 1;
        return;
    }
    if (sign = (val>=2048))
        val = 4096 - val;
}
else
{
    i+= run = tab->run;
    val = tab->level;
    sign = getbits(1);

    if (trace)

```

```

    printf("%d",sign);
}

if (i>=64)
{
    if (!quiet)
        fprintf(stderr,"DCT coeff index (i) out of bounds (inter2)\n");
    fault = 1;
    return;
}

if (trace)
    printf("): %d/%d",run,sign?-val:val);

j = (ld1->altscan ? alternate_scan : zig_zag_scan)[i];
val = (((val<<1)+1) * ld1->quant_scale * qmat[j]) >> 5;
bp[j] = sign ? -val : val;
nc++;

if (base.scalable_mode==SC_DP && nc==base.pri_brk-63)
    ld = &enhan;
}
}

```

## gethdr.c

```

/* gethdr.c, header decoding                                     */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>

```

```

#include "config.h"
#include "global.h"

/* private prototypes */
static int gethdr _ANSI_ARGS_((void));
static void getseqhdr _ANSI_ARGS_((void));
static void getgophdr _ANSI_ARGS_((void));
static void getpicturehdr _ANSI_ARGS_((void));
static void ext_user_data _ANSI_ARGS_((void));
static void sequence_extension _ANSI_ARGS_((void));
static void sequence_display_extension _ANSI_ARGS_((void));
static void quant_matrix_extension _ANSI_ARGS_((void));
static void sequence_scalable_extension _ANSI_ARGS_((void));
static void picture_display_extension _ANSI_ARGS_((void));
static void picture_coding_extension _ANSI_ARGS_((void));
static void picture_spatial_scalable_extension _ANSI_ARGS_((void));
static void picture_temporal_scalable_extension _ANSI_ARGS_((void));
static void ext_bit_info _ANSI_ARGS_((void));

/* decode headers from all input streams */

int getheader()
{
    int ret;

    ld = &base;
    ret = gethdr();
    if (twostreams)
    {
        ld = &enhan;
        if (gethdr() != ret && !quiet)
            fprintf(stderr, "streams out of sync\n");
        ld = &base;
    }

    return ret;
}

/*
 * decode headers from one input stream
 * until an End of Sequence or picture start code
 * is found
 */

static int gethdr()
{
    unsigned int code;

    for (;;)
    {
        /* look for startcode */
        startcode();
        code = getbits(32);
        switch (code)
        {
            case SEQ_START_CODE:
                getseqhdr();

```

```

        break;
    case GOP_START_CODE:
        getgophdr();
        break;
    case PICTURE_START_CODE:
        getpicturehdr();
        return 1;
        break;
    case SEQ_END_CODE:
        return 0;
        break;
    default:
        if (!quiet)
            fprintf(stderr, "Unexpected startcode %08x (ignored)\n", code);
        break;
    }
}
}

/* align to start of next startcode */

void startcode()
{
    /* byte align */
    flushbits(ld->incnt&7);
    while (showbits(24)!=11)
        flushbits(8);
}

/* decode sequence header */

static void getseqhdr()
{
    int i;
    int constrained_parameters_flag;
    int load_intra_quantizer_matrix, load_non_intra_quantizer_matrix;
    int pos;

    pos = ld->bitcnt;
    horizontal_size = getbits(12);
    vertical_size = getbits(12);
    aspect_ratio = getbits(4);
    picture_rate = getbits(4);
    bit_rate_value = getbits(18);
    flushbits(1); /* marker bit (=1) */
    vbv_buffer_size = getbits(10);
    constrained_parameters_flag = getbits(1);

    if (load_intra_quantizer_matrix = getbits(1))
    {
        for (i=0; i<64; i++)
            ld->intra_quantizer_matrix[zig_zag_scan[i]] = getbits(8);
    }
    else
    {
        for (i=0; i<64; i++)
            ld->intra_quantizer_matrix[i] = default_intra_quantizer_matrix[i];
    }
}

```

```

    }

    if (load_non_intra_quantizer_matrix == getbits(1))
    {
        for (i=0; i<64; i++)
            ld->non_intra_quantizer_matrix[zig_zag_scan[i]] = getbits(8);
    }
    else
    {
        for (i=0; i<64; i++)
            ld->non_intra_quantizer_matrix[i] = 16;
    }

    /* copy luminance to chrominance matrices */
    for (i=0; i<64; i++)
    {
        ld->chroma_intra_quantizer_matrix[i] =
            ld->intra_quantizer_matrix[i];

        ld->chroma_non_intra_quantizer_matrix[i] =
            ld->non_intra_quantizer_matrix[i];
    }

    if (verbose>0)
    {
        printf("sequence header (byte %d)\n", (pos>>3)-4);
        if (verbose>1)
        {
            printf(" horizontal_size=%d\n", horizontal_size);
            printf(" vertical_size=%d\n", vertical_size);
            printf(" aspect_ratio=%d\n", aspect_ratio);
            printf(" picture_rate=%d\n", picture_rate);
            printf(" bit_rate_value=%d\n", bit_rate_value);
            printf(" vbv_buffer_size=%d\n", vbv_buffer_size);
            printf(" constrained_parameters_flag=%d\n", constrained_parameters_flag);
            printf(" load_intra_quantizer_matrix=%d\n", load_intra_quantizer_matrix);
            printf(" load_non_intra_quantizer_matrix=%d\n", load_non_intra_quantizer_matrix);
        }
    }

    ext_user_data();
}

/* decode group of pictures header */

static void getgophdr()
{
    int drop_flag, hour, minute, sec, frame, closed_gop, broken_link;
    int pos;

    pos = ld->bitcnt;
    drop_flag = getbits(1);
    hour = getbits(5);
    minute = getbits(6);
    flushbits(1);
    sec = getbits(6);
    frame = getbits(6);
    closed_gop = getbits(1);

```

```

broken_link = getbits(1);

if (verbose>0)
{
    printf("group of pictures (byte %d)\n",(pos>>3)-4);
    if (verbose>1)
    {
        printf(" drop_flag=%d\n",drop_flag);
        printf(" timecode %d:%02d:%02d\n",hour,minute,sec,frame);
        printf(" closed_gop=%d\n",closed_gop);
        printf(" broken_link=%d\n",broken_link);
    }
}

ext_user_data();
}

/* decode picture header */

static void getpicturehdr()
{
    int pos;

    ld->pict_scal = 0; /* unless overwritten by pict. spat. scal. ext. */

    pos = ld->bitcnt;
    temp_ref = getbits(10);
    pict_type = getbits(3);
    vbv_delay = getbits(16);
    if (pict_type==P_TYPE || pict_type==B_TYPE)
    {
        full_forw = getbits(1);
        forw_r_size = getbits(3) - 1;
    }
    if (pict_type==B_TYPE)
    {
        full_back = getbits(1);
        back_r_size = getbits(3) - 1;
    }

    if (verbose>0)
    {
        printf("picture header (byte %d)\n",(pos>>3)-4);
        if (verbose>1)
        {
            printf(" temp_ref=%d\n",temp_ref);
            printf(" pict_type=%d\n",pict_type);
            printf(" vbv_delay=%d\n",vbv_delay);
            if (pict_type==P_TYPE || pict_type==B_TYPE)
            {
                printf(" full_forw=%d\n",full_forw);
                printf(" forw_r_size=%d\n",forw_r_size);
            }
            if (pict_type==B_TYPE)
            {
                printf(" full_back=%d\n",full_back);
                printf(" back_r_size=%d\n",back_r_size);
            }
        }
    }
}

```

```

    }
}

ext_bit_info();
ext_user_data();
}

/* decode slice header */

int getslicehdr()
{
    int slice_vertical_position_extension;
    int qs;
    int pos;

    pos = ld->bitcnt;
    slice_vertical_position_extension =
        (ld->mpeg2 && vertical_size>2800) ? getbits(3) : 0;

    if (ld->scalable_mode==SC_DP)
        ld->pri_brk = getbits(7);

    qs = getbits(5);
    ld->quant_scale =
        ld->mpeg2 ? (ld->qscale_type ? non_linear_mquant_table[qs] : qs<<1) : qs;

    if (getbits(1))
    {
        ld->intra_slice = getbits(1);
        flushbits(7);
        ext_bit_info();
    }
    else
        ld->intra_slice = 0;

    if (verbose>2)
    {
        printf("slice header (byte %d)\n", (pos>>3)-4);
        if (verbose>3)
        {
            if (ld->mpeg2 && vertical_size>2800)
                printf(" slice_vertical_position_extension=%d\n", slice_vertical_position_extension);
            if (ld->scalable_mode==SC_DP)
                printf(" priority_breakpoint=%d\n", ld->pri_brk);
            printf(" quantizer_scale_code=%d\n", qs);
        }
    }
    return slice_vertical_position_extension;
}

/* decode extension and user data */

static void ext_user_data()
{
    int code, ext_ID;

    startcode();

```



```

while ((code = showbits(32))==EXT_START_CODE || code==USER_START_CODE)
{
    if (code==EXT_START_CODE)
    {
        flushbits(32);
        ext_ID = getbits(4);
        switch (ext_ID)
        {
            case SEQ_ID:
                sequence_extension();
                break;
            case DISP_ID:
                sequence_display_extension();
                break;
            case QUANT_ID:
                quant_matrix_extension();
                break;
            case SEQSCAL_ID:
                sequence_scalable_extension();
                break;
            case PANSCAN_ID:
                picture_display_extension();
                break;
            case CODING_ID:
                picture_coding_extension();
                break;
            case SPATSCAL_ID:
                picture_spatial_scalable_extension();
                break;
            case TEMPSCAL_ID:
                picture_temporal_scalable_extension();
                break;
            default:
                fprintf(stderr,"reserved extension start code ID %d\n",ext_ID);
                break;
        }
        startcode();
    }
    else
    {
        if (verbose)
            printf("user data\n");
        flushbits(32);
        startcode();
    }
}
}

```

```

/* decode sequence extension */

```

```

static void sequence_extension()
{
    int prof_lev;
    int horizontal_size_extension, vertical_size_extension;
    int bit_rate_extension, vbv_buffer_size_extension;
    int frame_rate_extension_n, frame_rate_extension_d;
    int pos;

```

```

pos = ld->bitcnt;
ld->mpeg2 = 1;
ld->scalable_mode = SC_NONE; /* unless overwritten by seq. scal. ext. */
prof_lev = getbits(8);
prog_seq = getbits(1);
chroma_format = getbits(2);
horizontal_size_extension = getbits(2);
vertical_size_extension = getbits(2);
bit_rate_extension = getbits(12);
flushbits(1);
vbv_buffer_size_extension = getbits(8);
low_delay = getbits(1);
frame_rate_extension_n = getbits(2);
frame_rate_extension_d = getbits(5);

profile = prof_lev >> 4; /* Profile is upper nibble */
level = prof_lev & 15; /* Level is lower nibble */
horizontal_size = (horizontal_size_extension << 12) | (horizontal_size & 0x0fff);
vertical_size = (vertical_size_extension << 12) | (vertical_size & 0x0fff);
bit_rate = (float) ((bit_rate_extension << 18) + bit_rate_value) * 400.0;

if (verbose > 0)
{
    printf("sequence extension (byte %d)\n", (pos >> 3) - 4);
    if (verbose > 1)
    {
        printf(" profile_and_level_indication=%d\n", prof_lev);
        if (prof_lev < 128)
        {
            printf("  profile=%d, level=%d\n", profile, level);
        }
        printf(" progressive_sequence=%d\n", prog_seq);
        printf(" chroma_format=%d\n", chroma_format);
        printf(" horizontal_size_extension=%d\n", horizontal_size_extension);
        printf(" vertical_size_extension=%d\n", vertical_size_extension);
        printf(" bit_rate_extension=%d\n", bit_rate_extension);
        printf(" vbv_buffer_size_extension=%d\n", vbv_buffer_size_extension);
        printf(" low_delay=%d\n", low_delay);
        printf(" frame_rate_extension_n=%d\n", frame_rate_extension_n);
        printf(" frame_rate_extension_d=%d\n", frame_rate_extension_d);
    }
}
}

```

```
/* decode sequence display extension */
```

```

static void sequence_display_extension()
{
    int colour_description;
    int pos;

    pos = ld->bitcnt;
    video_format = getbits(3);
    colour_description = getbits(1);

    if (colour_description)
    {
        colour_primaries = getbits(8);
    }
}

```

```

    transfer_characteristics = getbits(8);
    matrix_coefficients = getbits(8);
}

display_horizontal_size = getbits(14);
flushbits(1);
display_vertical_size = getbits(14);

if (verbose>0)
{
    printf("sequence display extension (byte %d)\n",(pos>>3)-4);
    if (verbose>1)
    {
        printf(" video_format=%d\n",video_format);
        printf(" colour_description=%d\n",colour_description);
        if (colour_description)
        {
            printf(" colour_primaries=%d\n",colour_primaries);
            printf(" transfer_characteristics=%d\n",transfer_characteristics);
            printf(" matrix_coefficients=%d\n",matrix_coefficients);
        }
        printf(" display_horizontal_size=%d\n",display_horizontal_size);
        printf(" display_vertical_size=%d\n",display_vertical_size);
    }
}

/* decode quant matrix extension */

static void quant_matrix_extension()
{
    int i;
    int load_intra_quantiser_matrix, load_non_intra_quantiser_matrix;
    int load_chroma_intra_quantiser_matrix;
    int load_chroma_non_intra_quantiser_matrix;
    int pos;

    pos = ld->bitcnt;

    if (load_intra_quantiser_matrix = getbits(1))
    {
        for (i=0; i<64; i++)
        {
            ld->chroma_intra_quantizer_matrix[zig_zag_scan[i]]
            = ld->intra_quantizer_matrix[zig_zag_scan[i]]
            = getbits(8);
        }
    }

    if (load_non_intra_quantiser_matrix = getbits(1))
    {
        for (i=0; i<64; i++)
        {
            ld->chroma_non_intra_quantizer_matrix[zig_zag_scan[i]]
            = ld->non_intra_quantizer_matrix[zig_zag_scan[i]]
            = getbits(8);
        }
    }
}

```

```

if (load_chroma_intra_quantiser_matrix = getbits(1))
{
    for (i=0; i<64; i++)
        ld->chroma_intra_quantizer_matrix[zig_zag_scan[i]] = getbits(8);
}

if (load_chroma_non_intra_quantiser_matrix = getbits(1))
{
    for (i=0; i<64; i++)
        ld->chroma_non_intra_quantizer_matrix[zig_zag_scan[i]] = getbits(8);
}

if (verbose>0)
{
    printf("quant matrix extension (byte %d)\n", (pos>>3)-4);
    printf(" load_intra_quantiser_matrix=%d\n",
        load_intra_quantiser_matrix);
    printf(" load_non_intra_quantiser_matrix=%d\n",
        load_non_intra_quantiser_matrix);
    printf(" load_chroma_intra_quantiser_matrix=%d\n",
        load_chroma_intra_quantiser_matrix);
    printf(" load_chroma_non_intra_quantiser_matrix=%d\n",
        load_chroma_non_intra_quantiser_matrix);
}
}

/* decode sequence scalable extension */

static void sequence_scalable_extension()
{
    int layer_id;
    int pos;

    pos = ld->bitcnt;
    ld->scalable_mode = getbits(2) + 1; /* add 1 to make SC_DP != SC_NONE */
    layer_id = getbits(4);

    if (ld->scalable_mode==SC_SPAT)
    {
        llw = getbits(14); /* lower_layer_prediction_horizontal_size */
        flushbits(1);
        llh = getbits(14); /* lower_layer_prediction_vertical_size */
        hm = getbits(5);
        hn = getbits(5);
        vm = getbits(5);
        vn = getbits(5);
    }

    if (ld->scalable_mode==SC_TEMP)
        error("temporal scalability not implemented\n");

    if (verbose>0)
    {
        printf("sequence scalable extension (byte %d)\n", (pos>>3)-4);
        if (verbose>1)
        {
            printf(" scalable_mode=%d\n", ld->scalable_mode-1);
        }
    }
}

```

```

    printf(" layer_id=%d\n",layer_id);
    if (ld->scalable_mode==SC_SPAT)
    {
        printf(" lower_layer_prediction_horizontal_size=%d\n",llw);
        printf(" lower_layer_prediction_vertical_size=%d\n",llh);
        printf(" horizontal_subsampling_factor_m=%d\n",hm);
        printf(" horizontal_subsampling_factor_n=%d\n",hn);
        printf(" vertical_subsampling_factor_m=%d\n",vm);
        printf(" vertical_subsampling_factor_n=%d\n",vn);
    }
}
}
}

```

/\* decode picture display extension \*/

```

static void picture_display_extension()
{
    int i,n;
    short frame_centre_horizontal_offset[3];
    short frame_centre_vertical_offset[3];
    int pos;

    pos = ld->bitcnt;

    if (prog_seq || pict_struct!=FRAME_PICTURE)
        n = 1;
    else
        n = repeatfirst ? 3 : 2;

    for (i=0; i<n; i++)
    {
        frame_centre_horizontal_offset[i] = (short)getbits(16);
        flushbits(1);
        frame_centre_vertical_offset[i] = (short)getbits(16);
        flushbits(1);
    }

    if (verbose>0)
    {
        printf("picture display extension (byte %d)\n",(pos>>3)-4);
        if (verbose>1)
        {
            for (i=0; i<n; i++)
            {
                printf(" frame_centre_horizontal_offset[%d]=%d\n",i,
                    frame_centre_horizontal_offset[i]);
                printf(" frame_centre_vertical_offset[%d]=%d\n",i,
                    frame_centre_vertical_offset[i]);
            }
        }
    }
}

```

/\* decode picture coding extension \*/

```

static void picture_coding_extension()

```

```

{
    int chroma_420_type, composite_display_flag;
    int v_axis, field_sequence, sub_carrier, burst_amplitude, sub_carrier_phase;
    int pos;

    pos = ld->bitcnt;

    h_forw_r_size = getbits(4) - 1;
    v_forw_r_size = getbits(4) - 1;
    h_back_r_size = getbits(4) - 1;
    v_back_r_size = getbits(4) - 1;
    dc_prec = getbits(2);
    pict_struct = getbits(2);
    topfirst = getbits(1);
    frame_pred_dct = getbits(1);
    conceal_mv = getbits(1);
    ld->qscale_type = getbits(1);
    intravlc = getbits(1);
    ld->altscan = getbits(1);
    repeatfirst = getbits(1);
    chroma_420_type = getbits(1);
    prog_frame = getbits(1);
    composite_display_flag = getbits(1);
    if (composite_display_flag)
    {
        v_axis = getbits(1);
        field_sequence = getbits(3);
        sub_carrier = getbits(1);
        burst_amplitude = getbits(7);
        sub_carrier_phase = getbits(8);
    }

    if (verbose>0)
    {
        printf("picture coding extension (byte %d)\n", (pos>>3)-4);
        if (verbose>1)
        {
            printf(" forward_horizontal_f_code=%d\n", h_forw_r_size+1);
            printf(" forward_vertical_f_code=%d\n", v_forw_r_size+1);
            printf(" backward_horizontal_f_code=%d\n", h_back_r_size+1);
            printf(" backward_vertical_f_code=%d\n", v_back_r_size+1);
            printf(" intra_dc_precision=%d\n", dc_prec);
            printf(" picture_structure=%d\n", pict_struct);
            printf(" top_field_first=%d\n", topfirst);
            printf(" frame_pred_frame_dct=%d\n", frame_pred_dct);
            printf(" concealment_motion_vectors=%d\n", conceal_mv);
            printf(" q_scale_type=%d\n", ld->qscale_type);
            printf(" intra_vlc_format=%d\n", intravlc);
            printf(" alternate_scan=%d\n", ld->altscan);
            printf(" repeat_first_field=%d\n", repeatfirst);
            printf(" chroma_420_type=%d\n", chroma_420_type);
            printf(" progressive_frame=%d\n", prog_frame);
            printf(" composite_display_flag=%d\n", composite_display_flag);
            if (composite_display_flag)
            {
                printf(" v_axis=%d\n", v_axis);
                printf(" field_sequence=%d\n", field_sequence);
                printf(" sub_carrier=%d\n", sub_carrier);
                printf(" burst_amplitude=%d\n", burst_amplitude);
            }
        }
    }
}

```

```

        printf("  sub_carrier_phase=%d\n",sub_carrier_phase);
    }
}
}

/* decode picture spatial scalable extension */

static void picture_spatial_scalable_extension()
{
    int pos;

    pos = ld->bitcnt;

    ld->pict_scal = 1; /* use spatial scalability in this picture */

    lltempref = getbits(10);
    flushbits(1);
    llx0 = getbits(15);
    if (llx0>=16384)
        llx0-= 32768;
    flushbits(1);
    lly0 = getbits(15);
    if (lly0>=16384)
        lly0-= 32768;
    stwc_table_index = getbits(2);
    llprog_frame = getbits(1);
    llfieldsel = getbits(1);
    if (verbose>0)
    {
        printf("picture spatial scalable extension (byte %d)\n",(pos>>3)-4);
        if (verbose>1)
        {
            printf("  lower_layer_temporal_reference=%d\n",lltempref);
            printf("  lower_layer_horizontal_offset=%d\n",llx0);
            printf("  lower_layer_vertical_offset=%d\n",lly0);
            printf("  spatial_temporal_weight_code_table_index=%d\n",
                stwc_table_index);
            printf("  lower_layer_progressive_frame=%d\n",llprog_frame);
            printf("  lower_layer_deinterlaced_field_select=%d\n",llfieldsel);
        }
    }
}

/* decode picture temporal scalable extension
 *
 * not implemented
 */

static void picture_temporal_scalable_extension()
{
    error("temporal scalability not supported\n");
}

/* decode extra bit information */

```

```
static void ext_bit_info()
{
    while (getbits1())
        flushbits(8);
}
```

getpic.c

```
/* getpic.c, picture decoding */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>

#include "config.h"
#include "global.h"

/* private prototypes*/
static void getMBs _ANSI_ARGS__((int framenum));
static void macroblock_modes _ANSI_ARGS__((int *pmb_type, int *pstwtype,
    int *pstwclass, int *pmotion_type, int *pmv_count, int *pmv_format, int *pdmv,
    int *pmvscale, int *pdct_type));
static void clearblock _ANSI_ARGS__((int comp));
static void sumblock _ANSI_ARGS__((int comp));
static void saturate _ANSI_ARGS__((short *bp));
static void addblock _ANSI_ARGS__((int comp, int bx, int by,
    int dct_type, int addflag));

/* decode one frame or field picture */

void getpicture(framenum)
int framenum;
```



```

{
    int i;
    unsigned char *tmp;

    if (pict_struct==FRAME_PICTURE && secondfield)
    {
        /* recover from illegal number of field pictures */
        printf("odd number of field pictures\n");
        secondfield = 0;
    }

    for (i=0; i<3; i++)
    {
        if (pict_type==B_TYPE)
        {
            newframe[i] = auxframe[i];
        }
        else
        {
            if (!secondfield)
            {
                tmp = oldrefframe[i];
                oldrefframe[i] = refframe[i];
                refframe[i] = tmp;
            }

            newframe[i] = refframe[i];
        }

        if (pict_struct==BOTTOM_FIELD)
            newframe[i] += (i==0) ? coded_picture_width : chrom_width;
    }

    if (base.pict_scal && !secondfield)
        getspatref();

    getMBs(framenum);

    if (framenum!=0)
    {
        if (pict_struct==FRAME_PICTURE || secondfield)
        {
            if (pict_type==B_TYPE)
                storeframe(auxframe,framenum-1);
            else
                storeframe(oldrefframe,framenum-1);
        }
#ifdef DISPLAY
        else if (outtype==T_X11)
        {
            display_second_field();
        }
#endif
    }

    if (pict_struct!=FRAME_PICTURE)
        secondfield = !secondfield;
}

```

```

/* store last frame */

void putlast(framenum)
int framenum;
{
    if (secondfield)
        printf("last frame incomplete, not stored\n");
    else
        storeframe(refframe,framenum-1);
}

/* decode all macroblocks of the current picture */

static void getMBs(framenum)
int framenum;
{
    int comp;
    int MBA, MBAmax, MBAinc, mb_type, cbp, motion_type, dct_type;
    int slice_vert_pos_ext;
    int bx, by;
    unsigned int code;
    int dc_dct_pred[3];
    int mv_count, mv_format, mvscale;
    int PMV[2][2][2], mv_field_sel[2][2];
    int dmv, dmvector[2];
    int qs;
    int stwtype, stwclass;
    int SNRMBA, SNRMBAinc, SNRmb_type, SNRcbp, SNRdct_type, dummy; /* SNR scal. */

    /* number of macroblocks per picture */
    MBAmax = mb_width*mb_height;

    if (pict_struct!=FRAME_PICTURE)
        MBAmax>>=1; /* field picture has half as many macroblocks as frame */

    MBA = 0; /* macroblock address */
    MBAinc = 0;

    if (twostreams && enhan.scalable_mode==SC_SNR)
    {
        SNRMBA=0;
        SNRMBAinc=0;
    }

    fault=0;

    for (;;)
    {
        if (trace)
            printf("frame %d, MB %d\n",framenum,MBA);

#ifdef DISPLAY
        if (!prog_seq && pict_struct==FRAME_PICTURE && MBA==(MBAmax>>1) &&
            framenum!=0 && outtype==T_X11)
            display_second_field();
#endif
    }
}

```

```

ld = &base;
if (MBAinc==0)
{
    if (base.scalable_mode==SC_DP && base.pri_brk==1)
        ld = &enhan;

    if (!showbits(23) || fault) /* startcode or fault */
    {
resync: /* if fault: resynchronize to next startcode */
        fault = 0;
        ld = &base;

        if (MBA>=MBAmax)
            return; /* all macroblocks decoded */

        startcode();
        code = showbits(32);

        if (code<SLICE_MIN_START || code>SLICE_MAX_START)
        {
            /* only slice headers are allowed in picture_data */
            if (!quiet)
                printf("Premature end of picture\n");
            return;
        }

        flushbits(32);

        /* decode slice header (may change quant_scale) */
        slice_vert_pos_ext = getslicehdr();

        if (base.scalable_mode==SC_DP)
        {
            ld = &enhan;
            startcode();
            code = showbits(32);

            if (code<SLICE_MIN_START || code>SLICE_MAX_START)
            {
                /* only slice headers are allowed in picture_data */
                if (!quiet)
                    printf("Premature end of picture\n");
                return;
            }

            flushbits(32);

            /* decode slice header (may change quant_scale) */
            slice_vert_pos_ext = getslicehdr();

            if (base.pri_brk!=1)
                ld = &base;
        }

        /* decode macroblock address increment */
        MBAinc = getMBA();

        if (fault) goto resync;
    }
}

```

```

/* set current location */
MBA = ((slice_vert_pos_ext<<7) + (code&255) - 1)*mb_width + MBAinc - 1;
MBAinc = 1; /* first macroblock in slice: not skipped */

/* reset all DC coefficient and motion vector predictors */
dc_dct_pred[0]=dc_dct_pred[1]=dc_dct_pred[2]=0;
PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;
}
else /* neither startcode nor fault */
{
    if (MBA>=MBAmax)
    {
        if (!quiet)
            printf("Too many macroblocks in picture\n");
        return;
    }

    if (base.scalable_mode==SC_DP && base.pri_brk==1)
        ld = &enhan;

    /* decode macroblock address increment */
    MBAinc = getMBA();

    if (fault) goto resync;
}

if (MBA>=MBAmax)
{
    /* MBAinc points beyond picture dimensions */
    if (!quiet)
        printf("Too many macroblocks in picture\n");
    return;
}

if (MBAinc==1) /* not skipped */
{
    if (base.scalable_mode==SC_DP)
    {
        if (base.pri_brk<=2)
            ld = &enhan;
        else
            ld = &base;
    }

    macroblock_modes(&mb_type, &stwttype, &stwclass,
        &motion_type, &mv_count, &mv_format, &dmv, &mvscale,
        &dct_type);

    if (fault) goto resync;

    if (mb_type & MB_QUANT)
    {
        qs = getbits(5);

        if (trace)
        {
            printf("quantiser_scale_code (");

```

```

    printbits(qs,5,5);
    printf("): %d\n",qs);
}

if (ld->mpeg2)
    ld->quant_scale =
        ld->qscale_type ? non_linear_mquant_table[qs] : (qs << 1);
else
    ld->quant_scale = qs;

if (base.scalable_mode==SC_DP)
    /* make sure base.quant_scale is valid */
    base.quant_scale = ld->quant_scale;
}

/* motion vectors */

/* decode forward motion vectors */
if ((mb_type & MB_FORWARD) || ((mb_type & MB_INTRA) && conceal_mv))
{
    if (ld->mpeg2)
        motion_vectors(PMV,dmvector,mv_field_sel,
            0,mv_count,mv_format,h_forw_r_size,v_forw_r_size,dmv,mvscale);
    else
        motion_vector(PMV[0][0],dmvector,
            forw_r_size,forw_r_size,0,0,full_forw);
}

if (fault) goto resync;

/* decode backward motion vectors */
if (mb_type & MB_BACKWARD)
{
    if (ld->mpeg2)
        motion_vectors(PMV,dmvector,mv_field_sel,
            1,mv_count,mv_format,h_back_r_size,v_back_r_size,0,mvscale);
    else
        motion_vector(PMV[0][1],dmvector,
            back_r_size,back_r_size,0,0,full_back);
}

if (fault) goto resync;

if ((mb_type & MB_INTRA) && conceal_mv)
    flushbits(1); /* remove marker_bit */

if (base.scalable_mode==SC_DP && base.pri_brk==3)
    ld = &enhan;

/* macroblock_pattern */
if (mb_type & MB_PATTERN)
{
    cbp = getCBP();
    if (chroma_format==CHROMA422)
    {
        cbp = (cbp<<2) | getbits(2); /* coded_block_pattern_1 */

        if (trace)
        {

```

```

        printf("coded_block_pattern_1: ");
        printbits(cbp,2,2);
        printf(" (%d)\n",cbp&3);
    }
}
else if (chroma_format==CHROMA444)
{
    cbp = (cbp<<6) | getbits(6); /* coded_block_pattern_2 */

    if (trace)
    {
        printf("coded_block_pattern_2: ");
        printbits(cbp,6,6);
        printf(" (%d)\n",cbp&63);
    }
}
}
else
    cbp = (mb_type & MB_INTRA) ? (1<<blk_cnt)-1 : 0;

if (fault) goto resync;

/* decode blocks */
for (comp=0; comp<blk_cnt; comp++)
{
    if (base.scalable_mode==SC_DP)
        ld = &base;

    clearblock(comp);

    if (cbp & (1<<(blk_cnt-1-comp)))
    {
        if (mb_type & MB_INTRA)
        {
            if (ld->mpeg2)
                getmpg2intrablock(comp,dc_dct_pred);
            else
                getintrablock(comp,dc_dct_pred);
        }
        else
        {
            if (ld->mpeg2)
                getmpg2interblock(comp);
            else
                getinterblock(comp);
        }

        if (fault) goto resync;
    }
}

/* reset intra_dc predictors */
if (!(mb_type & MB_INTRA))
    dc_dct_pred[0]=dc_dct_pred[1]=dc_dct_pred[2]=0;

/* reset motion vector predictors */
if ((mb_type & MB_INTRA) && !conceal_mv)
{
    /* intra mb without concealment motion vectors */

```

```

    PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
    PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;
}

if ((pict_type==P_TYPE) && !(mb_type & (MB_FORWARD|MB_INTRA)))
{
    /* non-intra mb without forward mv in a P picture */
    PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;

    /* derive motion_type */
    if (pict_struct==FRAME_PICTURE)
        motion_type = MC_FRAME;
    else
    {
        motion_type = MC_FIELD;
        /* predict from field of same parity */
        mv_field_sel[0][0] = (pict_struct==BOTTOM_FIELD);
    }
}

if (stwclass==4)
{
    /* purely spatially predicted macroblock */
    PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;
    PMV[0][1][0]=PMV[0][1][1]=PMV[1][1][0]=PMV[1][1][1]=0;
}
else /* MBAinc!=1: skipped macroblock */
{
    if (base.scalable_mode==SC_DP)
        ld = &base;

    for (comp=0; comp<blk_cnt; comp++)
        clearblock(comp);

    /* reset intra_dc predictors */
    dc_dct_pred[0]=dc_dct_pred[1]=dc_dct_pred[2]=0;

    /* reset motion vector predictors */
    if (pict_type==P_TYPE)
        PMV[0][0][0]=PMV[0][0][1]=PMV[1][0][0]=PMV[1][0][1]=0;

    /* derive motion_type */
    if (pict_struct==FRAME_PICTURE)
        motion_type = MC_FRAME;
    else
    {
        motion_type = MC_FIELD;
        /* predict from field of same parity */
        mv_field_sel[0][0]=mv_field_sel[0][1] = (pict_struct==BOTTOM_FIELD);
    }

    /* skipped I are spatial-only predicted, */
    /* skipped P and B are temporal-only predicted */
    stwtype = (pict_type==I_TYPE) ? 8 : 0;

    /* clear MB_INTRA */
    mb_type&= ~MB_INTRA;
}

```

```

if (twostreams && enhan.scalable_mode==SC_SNR)
{
    ld = &enhan;
    if (SNRMBAinc==0)
    {
        if (!showbits(23)) /* startcode */
        {
            startcode();
            code = showbits(32);

            if (code<SLICE_MIN_START || code>SLICE_MAX_START)
            {
                /* only slice headers are allowed in picture_data */
                if (!quiet)
                    printf("Premature end of picture\n");
                return;
            }

            flushbits(32);

            /* decode slice header (may change quant_scale) */
            slice_vert_pos_ext = getslicehdr();

            /* decode macroblock address increment */
            SNRMBAinc = getMBA();

            /* set current location */
            SNRMBA =
                ((slice_vert_pos_ext<<7) + (code&255) - 1)*mb_width + SNRMBAinc - 1;

            SNRMBAinc = 1; /* first macroblock in slice: not skipped */
        }
        else /* not startcode */
        {
            if (SNRMBA>=MBAmax)
            {
                if (!quiet)
                    printf("Too many macroblocks in picture\n");
                return;
            }

            /* decode macroblock address increment */
            SNRMBAinc = getMBA();
        }
    }

    if (SNRMBA!=MBA)
    {
        /* streams out of sync */
        if (!quiet)
            printf("Can't synchronize streams\n");
        return;
    }

    if (SNRMBAinc==1) /* not skipped */
    {
        macroblock_modes(&SNRmb_type, &dummy, &dummy,
            &dummy, &dummy, &dummy, &dummy, &dummy,

```



```

    &SNRdct_type);

if (SNRmb_type & MB_PATTERN)
    dct_type = SNRdct_type;

if (SNRmb_type & MB_QUANT)
{
    qs = getbits(5);
    ld->quant_scale =
        ld->qscale_type ? non_linear_mquant_table[qs] : qs<<1;
}

/* macroblock_pattern */
if (SNRmb_type & MB_PATTERN)
{
    SNRcbp = getCBP();

    if (chroma_format==CHROMA422)
        SNRcbp = (SNRcbp<<2) | getbits(2); /* coded_block_pattern_1 */
    else if (chroma_format==CHROMA444)
        SNRcbp = (SNRcbp<<6) | getbits(6); /* coded_block_pattern_2 */
    }
    else
        SNRcbp = 0;

/* decode blocks */
for (comp=0; comp<blk_cnt; comp++)
{
    clearblock(comp);

    if (SNRcbp & (1<<(blk_cnt-1-comp)))
        getmpg2interblock(comp);
    }
}
else /* SNRMB_Ainc!=1: skipped macroblock */
{
    for (comp=0; comp<blk_cnt; comp++)
        clearblock(comp);
    }

ld = &base;
}

/* pixel coordinates of top left corner of current macroblock */
bx = 16*(MBA%mb_width);
by = 16*(MBA/mb_width);

/* motion compensation */
if (!(mb_type & MB_INTRA))
    reconstruct(bx,by,mb_type,motion_type,PMV,mv_field_sel,dmvector,
        stwtype);

if (base.scalable_mode==SC_DP)
    ld = &base;

/* copy or add block data into picture */
for (comp=0; comp<blk_cnt; comp++)
{
    if (twostreams && enhan.scalable_mode==SC_SNR)

```

```

    sumblock(comp); /* add SNR enhancement layer data to base layer */

    /* MPEG-2 saturation and mismatch control */
    if ((twostreams && enhan.scalable_mode==SC_SNR) || ld->mpeg2)
        saturate(ld->block[comp]);

    /* inverse DCT */
    if (refidct)
        idctref(ld->block[comp]);
    else
        idct(ld->block[comp]);

    addblock(comp,bx,by,dct_type,(mb_type & MB_INTRA)==0);
}

/* advance to next macroblock */
MBA++;
MBAinc--;

if (twostreams && enhan.scalable_mode==SC_SNR)
{
    SNRMBA++;
    SNRMBAinc--;
}
}

static void macroblock_modes(pmb_type,pstwtype,pstwclass,
    pmotion_type,pmv_count,pmv_format,pdmv,pmvscale,pdct_type)
int *pmb_type, *pstwtype, *pstwclass;
int *pmotion_type, *pmv_count, *pmv_format, *pdmv, *pmvscale;
int *pdct_type;
{
    int mb_type;
    int stwtype, stwcode, stwclass;
    int motion_type, mv_count, mv_format, dmvm, mvscale;
    int dct_type;
    static unsigned char stwc_table[3][4]
        = { {6,3,7,4}, {2,1,5,4}, {2,5,7,4} };
    static unsigned char stwclass_table[9]
        = {0, 1, 2, 1, 1, 2, 3, 3, 4};

    /* get macroblock_type */
    mb_type = getMBtype();

    if (fault) return;

    /* get spatial_temporal_weight_code */
    if (mb_type & MB_WEIGHT)
    {
        if (stwc_table_index==0)
            stwtype = 4;
        else
        {
            stwcode = getbits(2);
            if (trace)
            {
                printf("spatial_temporal_weight_code (");

```

```

        printbits(stwcode,2,2);
        printf("): %d\n",stwcode);
    }
    stwtype = stwc_table[stwc_table_index-1][stwcode];
}
}
else
    stwtype = (mb_type & MB_CLASS4) ? 8 : 0;

/* derive spatial_temporal_weight_class (Table 7-18) */
stwclass = stwclass_table[stwtype];

/* get frame/field motion type */
if (mb_type & (MB_FORWARD|MB_BACKWARD))
{
    if (pict_struct==FRAME_PICTURE) /* frame_motion_type */
    {
        motion_type = frame_pred_dct ? MC_FRAME : getbits(2);
        if (!frame_pred_dct && trace)
        {
            printf("frame_motion_type (");
            printbits(motion_type,2,2);
            printf("): %s\n",motion_type==MC_FIELD?"Field":
                motion_type==MC_FRAME?"Frame":
                motion_type==MC_DMV?"Dual_Prime":"Invalid");
        }
    }
    else /* field_motion_type */
    {
        motion_type = getbits(2);
        if (trace)
        {
            printf("field_motion_type (");
            printbits(motion_type,2,2);
            printf("): %s\n",motion_type==MC_FIELD?"Field":
                motion_type==MC_16X8?"16x8 MC":
                motion_type==MC_DMV?"Dual_Prime":"Invalid");
        }
    }
}
else if ((mb_type & MB_INTRA) && conceal_mv)
{
    /* concealment motion vectors */
    motion_type = (pict_struct==FRAME_PICTURE) ? MC_FRAME : MC_FIELD;
}

/* derive mv_count, mv_format and dmv, (table 6-17, 6-18) */
if (pict_struct==FRAME_PICTURE)
{
    mv_count = (motion_type==MC_FIELD && stwclass<2) ? 2 : 1;
    mv_format = (motion_type==MC_FRAME) ? MV_FRAME : MV_FIELD;
}
else
{
    mv_count = (motion_type==MC_16X8) ? 2 : 1;
    mv_format = MV_FIELD;
}

dmv = (motion_type==MC_DMV); /* dual prime */

```

```

/* field mv predictions in frame pictures have to be scaled */
mvscale = ((mv_format==MV_FIELD) && (pict_struct==FRAME_PICTURE));

/* get dct_type (frame DCT / field DCT) */
dct_type = (pict_struct==FRAME_PICTURE)
    && (!frame_pred_dct)
    && (mb_type & (MB_PATTERN|MB_INTRA))
    ? getbits(1)
    : 0;

if (trace && (pict_struct==FRAME_PICTURE)
    && (!frame_pred_dct)
    && (mb_type & (MB_PATTERN|MB_INTRA)))
    printf("dct_type (%d): %s\n",dct_type,dct_type?"Field":"Frame");

/* return values */
*pmb_type = mb_type;
*pstwtype = stwtype;
*pstwclass = stwclass;
*pmotion_type = motion_type;
*pmv_count = mv_count;
*pmv_format = mv_format;
*pdmv = dmv;
*pmvscale = mvscale;
*pdct_type = dct_type;
}

/* set block to zero */

static void clearblock(comp)
int comp;
{
    short *bp;
    int i;

    bp = ld->block[comp];

    for (i=0; i<64; i++)
        *bp++ = 0;
}

/* add SNR enhancement layer block data to base layer */

static void sumblock(comp)
int comp;
{
    short *bp1, *bp2;
    int i;

    bp1 = base.block[comp];
    bp2 = enhan.block[comp];

    for (i=0; i<64; i++)
        *bp1++ += *bp2++;
}

```

```

/* limit coefficients to -2048..2047 */

static void saturate(bp)
short *bp;
{
    int i, sum, val;

    sum = 0;

    /* saturation */
    for (i=0; i<64; i++)
    {
        val = bp[i];

        if (val>2047)
            val = 2047;
        else if (val<-2048)
            val = -2048;

        bp[i] = val;
        sum+= val;
    }

    /* mismatch control */
    if ((sum&1)==0)
        bp[63]^= 1;
}

/* move/add 8x8-Block from block[comp] to refframe */

static void addblock(comp,bx,by,dct_type,addflag)
int comp,bx,by,dct_type,addflag;
{
    int cc,i, j, iincr;
    unsigned char *rfp;
    short *bp;

    cc = (comp<4) ? 0 : (comp&1)+1; /* color component index */

    if (cc==0)
    {
        /* luminance */

        if (pict_struct==FRAME_PICTURE)
            if (dct_type)
            {
                /* field DCT coding */
                rfp = newframe[0]
                    + coded_picture_width*(by+((comp&2)>>1)) + bx + ((comp&1)<<3);
                iincr = (coded_picture_width<<1) - 8;
            }
            else
            {
                /* frame DCT coding */
                rfp = newframe[0]
                    + coded_picture_width*(by+((comp&2)<<2)) + bx + ((comp&1)<<3);
                iincr = coded_picture_width - 8;
            }
        }
    }

```

```

    }
else
{
    /* field picture */
    rfp = newframe[0]
        + (coded_picture_width<<1)*(by+((comp&2)<<2)) + bx + ((comp&1)<<3);
    iincr = (coded_picture_width<<1) - 8;
}
}
else
{
    /* chrominance */

    /* scale coordinates */
    if (chroma_format!=CHROMA444)
        bx >>= 1;
    if (chroma_format==CHROMA420)
        by >>= 1;
    if (pict_struct==FRAME_PICTURE)
    {
        if (dct_type && (chroma_format!=CHROMA420))
        {
            /* field DCT coding */
            rfp = newframe[cc]
                + chrom_width*(by+((comp&2)>>1)) + bx + (comp&8);
            iincr = (chrom_width<<1) - 8;
        }
        else
        {
            /* frame DCT coding */
            rfp = newframe[cc]
                + chrom_width*(by+((comp&2)<<2)) + bx + (comp&8);
            iincr = chrom_width - 8;
        }
    }
    else
    {
        /* field picture */
        rfp = newframe[cc]
            + (chrom_width<<1)*(by+((comp&2)<<2)) + bx + (comp&8);
        iincr = (chrom_width<<1) - 8;
    }
}

bp = ld->block[comp];

if (addflag)
{
    for (i=0; i<8; i++)
    {
        for (j=0; j<8; j++)
        {
            *rfp = clp[*bp++ + *rfp];
            rfp++;
        }

        rfp+= iincr;
    }
}

```

```

else
{
    for (i=0; i<8; i++)
    {
        for (j=0; j<8; j++)
            *rfp++ = clp[*bp++ + 128];

        rfp+= iincr;
    }
}
}

```

getvlc.c

```

/* getvlc.c, variable length decoding */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>

#include "config.h"
#include "global.h"
#include "getvlc.h"

/* private prototypes */
static int getIMBtype _ANSI_ARGS__((void));
static int getPMBtype _ANSI_ARGS__((void));
static int getBMBtype _ANSI_ARGS__((void));
static int getDMBtype _ANSI_ARGS__((void));
static int getspIMBtype _ANSI_ARGS__((void));
static int getspPMBtype _ANSI_ARGS__((void));
static int getspBMBtype _ANSI_ARGS__((void));

```

```

static int getSNRMBtype _ANSI_ARGS_((void));

int getMBtype()
{
    int mb_type;

    if (ld->scalable_mode==SC_SNR)
        mb_type = getSNRMBtype();
    else
    {
        switch (pict_type)
        {
            case I_TYPE:
                mb_type = ld->pict_scal ? getspIMBtype() : getIMBtype();
                break;
            case P_TYPE:
                mb_type = ld->pict_scal ? getspPMBtype() : getPMBtype();
                break;
            case B_TYPE:
                mb_type = ld->pict_scal ? getspBMBtype() : getBMBtype();
                break;
            case D_TYPE:
                /* MPEG-1 only, not implemented */
                mb_type = getDMBtype();
                break;
        }
    }

    return mb_type;
}

static int getIMBtype()
{
    if (trace)
        printf("mb_type(I) ");

    if (getbits1())
    {
        if (trace)
            printf("(1): Intra (1)\n");
        return 1;
    }

    if (!getbits1())
    {
        if (!quiet)
            fprintf(stderr, "Invalid macroblock_type code\n");
        fault = 1;
    }

    if (trace)
        printf("(01): Intra, Quant (17)\n");

    return 17;
}

static char *MBdescr[]={
    "",          "Intra",    "No MC, Coded",    "",
    "Bwd, Not Coded",    "",    "Bwd, Coded",    ""
};

```



```

    "Fwd, Not Coded", "", "Fwd, Coded", "",
    "Interp, Not Coded", "", "Interp, Coded", "",
    "", "Intra, Quant", "No MC, Coded, Quant", "",
    "", "", "Bwd, Coded, Quant", "",
    "", "", "Fwd, Coded, Quant", "",
    "", "", "Interp, Coded, Quant", ""
};

static int getPMBtype()
{
    int code;

    if (trace)
        printf("mb_type(P) (");

    if ((code = showbits(6)) >= 8)
    {
        code >>= 3;
        flushbits(PMBtab0[code].len);
        if (trace)
        {
            printbits(code, 3, PMBtab0[code].len);
            printf("): %s (%d)\n", MBdescr[PMBtab0[code].val], PMBtab0[code].val);
        }
        return PMBtab0[code].val;
    }

    if (code == 0)
    {
        if (!quiet)
            fprintf(stderr, "Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    flushbits(PMBtab1[code].len);

    if (trace)
    {
        printbits(code, 6, PMBtab1[code].len);
        printf("): %s (%d)\n", MBdescr[PMBtab1[code].val], PMBtab1[code].val);
    }

    return PMBtab1[code].val;
}

static int getBMBtype()
{
    int code;

    if (trace)
        printf("mb_type(B) (");

    if ((code = showbits(6)) >= 8)
    {
        code >>= 2;
        flushbits(BMBtab0[code].len);

        if (trace)

```

```

    {
        printbits(code,4,BMBtab0[code].len);
        printf("): %s (%d)\n",MBdescr[BMBtab0[code].val],BMBtab0[code].val);
    }

    return BMBtab0[code].val;
}

if (code==0)
{
    if (!quiet)
        fprintf(stderr,"Invalid macroblock_type code\n");
    fault = 1;
    return 0;
}

flushbits(BMBtab1[code].len);

if (trace)
{
    printbits(code,6,BMBtab1[code].len);
    printf("): %s (%d)\n",MBdescr[BMBtab1[code].val],BMBtab1[code].val);
}

return BMBtab1[code].val;
}

static int getDMBtype()
{
    if (!getbits1())
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault=1;
    }

    return 1;
}

/* macroblock_type for pictures with spatial scalability */
static int getspIMBtype()
{
    int code;

    if (trace)
        printf("mb_type(I,spat) (");

    code = showbits(4);

    if (code==0)
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    if (trace)
    {

```

```

    printbits(code,4,spIMBtab[code].len);
    printf("): %02x\n",spIMBtab[code].val);
}

flushbits(spIMBtab[code].len);
return spIMBtab[code].val;
}

static int getspPMBtype()
{
    int code;

    if (trace)
        printf("mb_type(P,spat) (");

    code = showbits(7);

    if (code<2)
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    if (code>=16)
    {
        code >>= 3;
        flushbits(spPMBtab0[code].len);

        if (trace)
        {
            printbits(code,4,spPMBtab0[code].len);
            printf("): %02x\n",spPMBtab0[code].val);
        }

        return spPMBtab0[code].val;
    }

    flushbits(spPMBtab1[code].len);

    if (trace)
    {
        printbits(code,7,spPMBtab1[code].len);
        printf("): %02x\n",spPMBtab1[code].val);
    }

    return spPMBtab1[code].val;
}

static int getspBMBtype()
{
    int code;
    VLCTab *p;

    if (trace)
        printf("mb_type(B,spat) (");

    code = showbits(9);

```

```

if (code>=64)
    p = &spBMBtab0[(code>>5)-2];
else if (code>=16)
    p = &spBMBtab1[(code>>2)-4];
else if (code>=8)
    p = &spBMBtab2[code-8];
else
{
    if (!quiet)
        fprintf(stderr,"Invalid macroblock_type code\n");
    fault = 1;
    return 0;
}

flushbits(p->len);

if (trace)
{
    printbits(code,9,p->len);
    printf("): %02x\n",p->val);
}

return p->val;
}

static int getSNRMBtype()
{
    int code;

    code = showbits(3);

    if (code==0)
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    flushbits(SNRMBtab[code].len);
    return SNRMBtab[code].val;
}

int getMV()
{
    int code;

    if (trace)
        printf("motion_code (");

    if (getbits1())
    {
        if (trace)
            printf("0): 0\n");
        return 0;
    }

    if ((code = showbits(9))>=64)

```

```

{
    code >>= 6;
    flushbits(MVtab0[code].len);

    if (trace)
    {
        printbits(code,3,MVtab0[code].len);
        printf("%d): %d\n",
            showbits(1),showbits(1)?-MVtab0[code].val:MVtab0[code].val);
    }

    return getbits(1)?-MVtab0[code].val:MVtab0[code].val;
}

if (code>=24)
{
    code >>= 3;
    flushbits(MVtab1[code].len);

    if (trace)
    {
        printbits(code,6,MVtab1[code].len);
        printf("%d): %d\n",
            showbits(1),showbits(1)?-MVtab1[code].val:MVtab1[code].val);
    }

    return getbits(1)?-MVtab1[code].val:MVtab1[code].val;
}

if ((code-=12)<0)
{
    if (!quiet)
        fprintf(stderr,"Invalid motion_vector code\n");
    fault=1;
    return 0;
}

flushbits(MVtab2[code].len);

if (trace)
{
    printbits(code+12,9,MVtab2[code].len);
    printf("%d): %d\n",
        showbits(1),showbits(1)?-MVtab2[code].val:MVtab2[code].val);
}

return getbits(1) ? -MVtab2[code].val : MVtab2[code].val;
}

/* get differential motion vector (for dual prime prediction) */
int getDMV()
{
    if (trace)
        printf("dmvector (");

    if (getbits(1))
    {
        if (trace)
            printf(showbits(1) ? "11): -1\n" : "10): 1\n");
    }

```

```

    return getbits(1) ? -1 : 1;
}
else
{
    if (trace)
        printf("0): 0\n");
    return 0;
}
}

int getCBP()
{
    int code;

    if (trace)
        printf("coded_block_pattern_420 (");

    if ((code = showbits(9))>=128)
    {
        code >>= 4;
        flushbits(CBPtab0[code].len);

        if (trace)
        {
            printbits(code,5,CBPtab0[code].len);
            printf("): ");
            printbits(CBPtab0[code].val,6,6);
            printf(" (%d)\n",CBPtab0[code].val);
        }

        return CBPtab0[code].val;
    }

    if (code>=8)
    {
        code >>= 1;
        flushbits(CBPtab1[code].len);

        if (trace)
        {
            printbits(code,8,CBPtab1[code].len);
            printf("): ");
            printbits(CBPtab1[code].val,6,6);
            printf(" (%d)\n",CBPtab1[code].val);
        }

        return CBPtab1[code].val;
    }

    if (code<1)
    {
        if (!quiet)
            fprintf(stderr,"Invalid coded_block_pattern code\n");
        fault = 1;
        return 0;
    }

    flushbits(CBPtab2[code].len);

```

```

    if (trace)
    {
        printbits(code,9,CBPtab2[code].len);
        printf("): ");
        printbits(CBPtab2[code].val,6,6);
        printf(" (%d)\n",CBPtab2[code].val);
    }

    return CBPtab2[code].val;
}

int getMBA()
{
    int code, val;

    if (trace)
        printf("macroblock_address_increment (");

    val = 0;

    while ((code = showbits(11))<24)
    {
        if (code!=15) /* if not macroblock_stuffing */
        {
            if (code==8) /* if macroblock_escape */
            {
                if (trace)
                    printf("00000001000 ");

                val+= 33;
            }
            else
            {
                if (!quiet)
                    fprintf(stderr,"Invalid macroblock_address_increment code\n");

                fault = 1;
                return 1;
            }
        }
        else /* macroblock stuffing */
        {
            if (trace)
                printf("00000001111 ");
        }

        flushbits(11);
    }

    if (code>=1024)
    {
        flushbits(1);
        if (trace)
            printf("1): %d\n",val+1);
        return val + 1;
    }

    if (code>=128)
    {

```

```

    code >>= 6;
    flushbits(MBAtab1[code].len);

    if (trace)
    {
        printbits(code,5,MBAtab1[code].len);
        printf("): %d\n",val+MBAtab1[code].val);
    }

    return val + MBAtab1[code].val;
}

code-= 24;
flushbits(MBAtab2[code].len);

if (trace)
{
    printbits(code+24,11,MBAtab2[code].len);
    printf("): %d\n",val+MBAtab2[code].val);
}

return val + MBAtab2[code].val;
}

int getDClum()
{
    int code, size, val;

    if (trace)
        printf("dct_dc_size_luminance: (");

    /* decode length */
    code = showbits(5);

    if (code<31)
    {
        size = DClumtab0[code].val;
        flushbits(DClumtab0[code].len);

        if (trace)
        {
            printbits(code,5,DClumtab0[code].len);
            printf("): %d",size);
        }
    }
    else
    {
        code = showbits(9) - 0x1f0;
        size = DClumtab1[code].val;
        flushbits(DClumtab1[code].len);

        if (trace)
        {
            printbits(code+0x1f0,9,DClumtab1[code].len);
            printf("): %d",size);
        }
    }

    if (trace)

```



```

    printf(", dct_dc_differential (");

    if (size==0)
        val = 0;
    else
    {
        val = getbits(size);
        if (trace)
            printbits(val,size,size);
        if ((val & (1<<(size-1)))==0)
            val-= (1<<size) - 1;
    }

    if (trace)
        printf("): %d\n",val);

    return val;
}

int getDCchrom()
{
    int code, size, val;

    if (trace)
        printf("dct_dc_size_chrominance: (");

    /* decode length */
    code = showbits(5);

    if (code<31)
    {
        size = DCchromtab0[code].val;
        flushbits(DCchromtab0[code].len);

        if (trace)
        {
            printbits(code,5,DCchromtab0[code].len);
            printf("): %d",size);
        }
    }
    else
    {
        code = showbits(10) - 0x3e0;
        size = DCchromtab1[code].val;
        flushbits(DCchromtab1[code].len);

        if (trace)
        {
            printbits(code+0x3e0,10,DCchromtab1[code].len);
            printf("): %d",size);
        }
    }

    if (trace)
        printf(", dct_dc_differential (");

    if (size==0)
        val = 0;
    else

```

```

{
    val = getbits(size);
    if (trace)
        printbits(val,size,size);
    if ((val & (1<<(size-1)))==0)
        val-= (1<<size) - 1;
}

if (trace)
    printf("): %d\n",val);

return val;
}

```

getvlc.h

```

/* getvlc.c, variable length decoding */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>

#include "config.h"
#include "global.h"
#include "getvlc.h"

/* private prototypes */
static int getIMBtype _ANSI_ARGS__((void));
static int getPMBtype _ANSI_ARGS__((void));
static int getBMBtype _ANSI_ARGS__((void));
static int getDMBtype _ANSI_ARGS__((void));
static int getspIMBtype _ANSI_ARGS__((void));

```

```

static int getspPMBtype _ANSI_ARGS_((void));
static int getspBMBtype _ANSI_ARGS_((void));
static int getSNRMBtype _ANSI_ARGS_((void));

int getMBtype()
{
    int mb_type;

    if (ld->scalable_mode==SC_SNR)
        mb_type = getSNRMBtype();
    else
    {
        switch (pict_type)
        {
            case I_TYPE:
                mb_type = ld->pict_scal ? getspIMBtype() : getIMBtype();
                break;
            case P_TYPE:
                mb_type = ld->pict_scal ? getspPMBtype() : getPMBtype();
                break;
            case B_TYPE:
                mb_type = ld->pict_scal ? getspBMBtype() : getBMBtype();
                break;
            case D_TYPE:
                /* MPEG-1 only, not implemented */
                mb_type = getDMBtype();
                break;
        }
    }

    return mb_type;
}

static int getIMBtype()
{
    if (trace)
        printf("mb_type(I) ");

    if (getbits1())
    {
        if (trace)
            printf("(1): Intra (1)\n");
        return 1;
    }

    if (!getbits1())
    {
        if (!quiet)
            fprintf(stderr, "Invalid macroblock_type code\n");
        fault = 1;
    }

    if (trace)
        printf("(01): Intra, Quant (17)\n");

    return 17;
}

static char *MBdescr[]={

```

```

    "",          "Intra",      "No MC, Coded",      "",
    "Bwd, Not Coded", "",      "Bwd, Coded",      "",
    "Fwd, Not Coded", "",      "Fwd, Coded",      "",
    "Interp, Not Coded", "",      "Interp, Coded",      "",
    "",          "Intra, Quant", "No MC, Coded, Quant", "",
    "",          "",          "Bwd, Coded, Quant", "",
    "",          "",          "Fwd, Coded, Quant", "",
    "",          "",          "Interp, Coded, Quant", ""
};

static int getPMBtype()
{
    int code;

    if (trace)
        printf("mb_type(P) (");

    if ((code = showbits(6))>=8)
    {
        code >>= 3;
        flushbits(PMBtab0[code].len);
        if (trace)
        {
            printbits(code,3,PMBtab0[code].len);
            printf("): %s (%d)\n",MBdescr[PMBtab0[code].val],PMBtab0[code].val);
        }
        return PMBtab0[code].val;
    }

    if (code==0)
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    flushbits(PMBtab1[code].len);

    if (trace)
    {
        printbits(code,6,PMBtab1[code].len);
        printf("): %s (%d)\n",MBdescr[PMBtab1[code].val],PMBtab1[code].val);
    }

    return PMBtab1[code].val;
}

static int getBMBtype()
{
    int code;

    if (trace)
        printf("mb_type(B) (");

    if ((code = showbits(6))>=8)
    {
        code >>= 2;
        flushbits(BMBtab0[code].len);

```

```

    if (trace)
    {
        printbits(code,4,BMBtab0[code].len);
        printf("): %s (%d)\n",MBdescr[BMBtab0[code].val],BMBtab0[code].val);
    }

    return BMBtab0[code].val;
}

if (code==0)
{
    if (!quiet)
        fprintf(stderr,"Invalid macroblock_type code\n");
    fault = 1;
    return 0;
}

flushbits(BMBtab1[code].len);

if (trace)
{
    printbits(code,6,BMBtab1[code].len);
    printf("): %s (%d)\n",MBdescr[BMBtab1[code].val],BMBtab1[code].val);
}

return BMBtab1[code].val;
}

static int getDMBtype()
{
    if (!getbits1())
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault=1;
    }

    return 1;
}

/* macroblock_type for pictures with spatial scalability */
static int getspIMBtype()
{
    int code;

    if (trace)
        printf("mb_type(I,spat) (");

    code = showbits(4);

    if (code==0)
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }
}

```

```

    if (trace)
    {
        printbits(code,4,spIMBtab[code].len);
        printf("): %02x\n",spIMBtab[code].val);
    }

    flushbits(spIMBtab[code].len);
    return spIMBtab[code].val;
}

static int getspPMBtype()
{
    int code;

    if (trace)
        printf("mb_type(P,spat) (");

    code = showbits(7);

    if (code<2)
    {
        if (!quiet)
            fprintf(stderr,"Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    if (code>=16)
    {
        code >>= 3;
        flushbits(spPMBtab0[code].len);

        if (trace)
        {
            printbits(code,4,spPMBtab0[code].len);
            printf("): %02x\n",spPMBtab0[code].val);
        }

        return spPMBtab0[code].val;
    }

    flushbits(spPMBtab1[code].len);

    if (trace)
    {
        printbits(code,7,spPMBtab1[code].len);
        printf("): %02x\n",spPMBtab1[code].val);
    }

    return spPMBtab1[code].val;
}

static int getspBMBtype()
{
    int code;
    VLCTab *p;

    if (trace)
        printf("mb_type(B,spat) (");

```

```

code = showbits(9);

if (code >= 64)
    p = &spBMBtab0[(code >> 5) - 2];
else if (code >= 16)
    p = &spBMBtab1[(code >> 2) - 4];
else if (code >= 8)
    p = &spBMBtab2[code - 8];
else
{
    if (!quiet)
        fprintf(stderr, "Invalid macroblock_type code\n");
    fault = 1;
    return 0;
}

flushbits(p->len);

if (trace)
{
    printbits(code, 9, p->len);
    printf("): %02x\n", p->val);
}

return p->val;
}

static int getSNRMBtype()
{
    int code;

    code = showbits(3);

    if (code == 0)
    {
        if (!quiet)
            fprintf(stderr, "Invalid macroblock_type code\n");
        fault = 1;
        return 0;
    }

    flushbits(SNRMBtab[code].len);
    return SNRMBtab[code].val;
}

int getMV()
{
    int code;

    if (trace)
        printf("motion_code (");

    if (getbits1())
    {
        if (trace)
            printf("0): 0\n");
        return 0;
    }
}

```

```

if ((code = showbits(9))>=64)
{
    code >>= 6;
    flushbits(MVtab0[code].len);

    if (trace)
    {
        printbits(code,3,MVtab0[code].len);
        printf("%d): %d\n",
            showbits(1),showbits(1)?-MVtab0[code].val:MVtab0[code].val);
    }

    return getbits(1)?-MVtab0[code].val:MVtab0[code].val;
}

if (code>=24)
{
    code >>= 3;
    flushbits(MVtab1[code].len);

    if (trace)
    {
        printbits(code,6,MVtab1[code].len);
        printf("%d): %d\n",
            showbits(1),showbits(1)?-MVtab1[code].val:MVtab1[code].val);
    }

    return getbits(1)?-MVtab1[code].val:MVtab1[code].val;
}

if ((code-=12)<0)
{
    if (!quiet)
        fprintf(stderr,"Invalid motion_vector code\n");
    fault=1;
    return 0;
}

flushbits(MVtab2[code].len);

if (trace)
{
    printbits(code+12,9,MVtab2[code].len);
    printf("%d): %d\n",
        showbits(1),showbits(1)?-MVtab2[code].val:MVtab2[code].val);
}

return getbits(1) ? -MVtab2[code].val : MVtab2[code].val;
}

/* get differential motion vector (for dual prime prediction) */
int getDMV()
{
    if (trace)
        printf("dmvector (");

    if (getbits(1))
    {

```



```

    if (trace)
        printf(showbits(1) ? "11): -1\n" : "10): 1\n");
    return getbits(1) ? -1 : 1;
}
else
{
    if (trace)
        printf("0): 0\n");
    return 0;
}
}

int getCBP()
{
    int code;

    if (trace)
        printf("coded_block_pattern_420 (");

    if ((code = showbits(9)) >= 128)
    {
        code >>= 4;
        flushbits(CBPtab0[code].len);

        if (trace)
        {
            printbits(code, 5, CBPtab0[code].len);
            printf("): ");
            printbits(CBPtab0[code].val, 6, 6);
            printf(" (%d)\n", CBPtab0[code].val);
        }

        return CBPtab0[code].val;
    }

    if (code >= 8)
    {
        code >>= 1;
        flushbits(CBPtab1[code].len);

        if (trace)
        {
            printbits(code, 8, CBPtab1[code].len);
            printf("): ");
            printbits(CBPtab1[code].val, 6, 6);
            printf(" (%d)\n", CBPtab1[code].val);
        }

        return CBPtab1[code].val;
    }

    if (code < 1)
    {
        if (!quiet)
            fprintf(stderr, "Invalid coded_block_pattern code\n");
        fault = 1;
        return 0;
    }
}

```

```

flushbits(CBPtrab2[code].len);

if (trace)
{
    printbits(code,9,CBPtrab2[code].len);
    printf("): ");
    printbits(CBPtrab2[code].val,6,6);
    printf(" (%d)\n",CBPtrab2[code].val);
}

return CBPtrab2[code].val;
}

int getMBA()
{
    int code, val;

    if (trace)
        printf("macroblock_address_increment (");

    val = 0;

    while ((code = showbits(11))<24)
    {
        if (code!=15) /* if not macroblock_stuffing */
        {
            if (code==8) /* if macroblock_escape */
            {
                if (trace)
                    printf("00000001000 ");

                val+= 33;
            }
            else
            {
                if (!quiet)
                    fprintf(stderr,"Invalid macroblock_address_increment code\n");

                fault = 1;
                return 1;
            }
        }
        else /* macroblock stuffing */
        {
            if (trace)
                printf("00000001111 ");
        }

        flushbits(11);
    }

    if (code>=1024)
    {
        flushbits(1);
        if (trace)
            printf("1): %d\n",val+1);
        return val + 1;
    }
}

```

```

if (code>=128)
{
    code >>= 6;
    flushbits(MBAtab1[code].len);

    if (trace)
    {
        printbits(code,5,MBAtab1[code].len);
        printf("): %d\n",val+MBAtab1[code].val);
    }

    return val + MBAtab1[code].val;
}

code-= 24;
flushbits(MBAtab2[code].len);

if (trace)
{
    printbits(code+24,11,MBAtab2[code].len);
    printf("): %d\n",val+MBAtab2[code].val);
}

return val + MBAtab2[code].val;
}

int getDClum()
{
    int code, size, val;

    if (trace)
        printf("dct_dc_size_luminance: (");

    /* decode length */
    code = showbits(5);

    if (code<31)
    {
        size = DClumtab0[code].val;
        flushbits(DClumtab0[code].len);

        if (trace)
        {
            printbits(code,5,DClumtab0[code].len);
            printf("): %d",size);
        }
    }
    else
    {
        code = showbits(9) - 0x1f0;
        size = DClumtab1[code].val;
        flushbits(DClumtab1[code].len);

        if (trace)
        {
            printbits(code+0x1f0,9,DClumtab1[code].len);
            printf("): %d",size);
        }
    }
}

```

```

    if (trace)
        printf(" dct_dc_differential (");

    if (size==0)
        val = 0;
    else
    {
        val = getbits(size);
        if (trace)
            printbits(val,size,size);
        if ((val & (1<<(size-1)))==0)
            val-= (1<<size) - 1;
    }

    if (trace)
        printf("): %d\n",val);

    return val;
}

int getDCchrom()
{
    int code, size, val;

    if (trace)
        printf("dct_dc_size_chrominance: (");

    /* decode length */
    code = showbits(5);

    if (code<31)
    {
        size = DCchromtab0[code].val;
        flushbits(DCchromtab0[code].len);

        if (trace)
        {
            printbits(code,5,DCchromtab0[code].len);
            printf("): %d",size);
        }
    }
    else
    {
        code = showbits(10) - 0x3e0;
        size = DCchromtab1[code].val;
        flushbits(DCchromtab1[code].len);

        if (trace)
        {
            printbits(code+0x3e0,10,DCchromtab1[code].len);
            printf("): %d",size);
        }
    }

    if (trace)
        printf(" dct_dc_differential (");

    if (size==0)

```

```

    val = 0;
else
{
    val = getbits(size);
    if (trace)
        printbits(val,size,size);
    if ((val & (1<<(size-1)))==0)
        val-= (1<<size) - 1;
}

if (trace)
    printf("): %d\n",val);

return val;
}

```

global.h

```

/* global.h, global variables                                     */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include "mpeg2dec.h"

/* choose between declaration (GLOBAL undefined)
 * and definition (GLOBAL defined)
 * GLOBAL is defined in exactly one file (mpeg2dec.c)
 */

#ifndef GLOBAL
#define EXTERN extern
#else

```

```

#define EXTERN
#endif

/* prototypes of global functions */

/* comply.c */
void sequence_layer_checks _ANSI_ARGS_((void));
void picture_layer_checks _ANSI_ARGS_((void));

/* getbits.c */
void initbits _ANSI_ARGS_((void));
void fillbfr _ANSI_ARGS_((void));
unsigned int showbits _ANSI_ARGS_((int n));
unsigned int getbits1 _ANSI_ARGS_((void));
void flushbits _ANSI_ARGS_((int n));
unsigned int getbits _ANSI_ARGS_((int n));

/* getblk.c */
void getintrablock _ANSI_ARGS_((int comp, int dc_dct_pred[]);
void getinterblock _ANSI_ARGS_((int comp));
void getmpg2intrablock _ANSI_ARGS_((int comp, int dc_dct_pred[]);
void getmpg2interblock _ANSI_ARGS_((int comp));

/* gethdr.c */
int getheader _ANSI_ARGS_((void));
void startcode _ANSI_ARGS_((void));
int getslicehdr _ANSI_ARGS_((void));

/* getpic.c */
void getpicture _ANSI_ARGS_((int framenum));
void putlast _ANSI_ARGS_((int framenum));

/* getvlc.c */
int getMBtype _ANSI_ARGS_((void));
int getMV _ANSI_ARGS_((void));
int getDMV _ANSI_ARGS_((void));
int getCBP _ANSI_ARGS_((void));
int getMBA _ANSI_ARGS_((void));
int getDClum _ANSI_ARGS_((void));
int getDCchrom _ANSI_ARGS_((void));

/* idct.c */
void idct _ANSI_ARGS_((short *block));
void init_idct _ANSI_ARGS_((void));

/* idctref.c */
void init_idctref _ANSI_ARGS_((void));
void idctref _ANSI_ARGS_((short *block));

/* motion.c */
void motion_vectors _ANSI_ARGS_((int PMV[2][2][2], int dmvector[2],
    int mv_field_sel[2][2], int s, int mv_count, int mv_format,
    int h_r_size, int v_r_size, int dmvector, int mvscale));
void motion_vector _ANSI_ARGS_((int *PMV, int *dmvector,
    int h_r_size, int v_r_size, int dmvector, int mvscale, int full_pel_vector));
void calc_DMV _ANSI_ARGS_((int DMV[][2], int *dmvector, int mvx, int mvy));

/* mpeg2dec.c */
void error _ANSI_ARGS_((char *text));

```

```

void warning _ANSI_ARGS_((char *text));
void printbits _ANSI_ARGS_((int code, int bits, int len));

/* recon.c */
void reconstruct _ANSI_ARGS_((int bx, int by, int mb_type, int motion_type,
    int PMV[2][2][2], int mv_field_sel[2][2], int dmvector[2], int stwtype));

/* spatasc.c */
void getspatref _ANSI_ARGS_((void));

/* store.c */
void storeframe _ANSI_ARGS_((unsigned char *src[], int frame));

#ifdef DISPLAY
/* display.c */
void init_display _ANSI_ARGS_((char *name));
void exit_display _ANSI_ARGS_((void));
void display_second_field _ANSI_ARGS_((void));
void dither _ANSI_ARGS_((unsigned char *src[]));
void init_dither _ANSI_ARGS_((void));
#endif

/* global variables */

EXTERN char version[]
#ifdef GLOBAL
    ="mpeg2decode V1.1a, 94/07/04"
#endif
;

EXTERN char author[]
#ifdef GLOBAL
    ="(C) 1994, MPEG Software Simulation Group"
#endif
;

/* zig-zag scan */
EXTERN unsigned char zig_zag_scan[64]
#ifdef GLOBAL
=
{
    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
}
#endif
;

/* alternate scan */
EXTERN unsigned char alternate_scan[64]
#ifdef GLOBAL
=
{
    0,8,16,24,1,9,2,10,17,25,32,40,48,56,57,49,
    41,33,26,18,3,11,4,12,19,27,34,42,50,58,35,43,
    51,59,20,28,5,13,6,14,21,29,36,44,52,60,37,45,
    53,61,22,30,7,15,23,31,38,46,54,62,39,47,55,63
}
}

```

```

#endif
;

/* default intra quantization matrix */
EXTERN unsigned char default_intra_quantizer_matrix[64]
#ifdef GLOBAL
=
{
    8, 16, 19, 22, 26, 27, 29, 34,
    16, 16, 22, 24, 27, 29, 34, 37,
    19, 22, 26, 27, 29, 34, 34, 38,
    22, 22, 26, 27, 29, 34, 37, 40,
    22, 26, 27, 29, 32, 35, 40, 48,
    26, 27, 29, 32, 35, 40, 48, 58,
    26, 27, 29, 34, 38, 46, 56, 69,
    27, 29, 35, 38, 46, 56, 69, 83
}
#endif
;

/* non-linear quantization coefficient table */
EXTERN unsigned char non_linear_mquant_table[32]
#ifdef GLOBAL
=
{
    0, 1, 2, 3, 4, 5, 6, 7,
    8, 10, 12, 14, 16, 18, 20, 22,
    24, 28, 32, 36, 40, 44, 48, 52,
    56, 64, 72, 80, 88, 96, 104, 112
}
#endif
;

/* color space conversion coefficients
 *
 * entries are {crv,cbu,cgu,cgv}
 *
 * crv=(255/224)*65536*(1-cr)/0.5
 * cbu=(255/224)*65536*(1-cb)/0.5
 * cgu=(255/224)*65536*(cb/cg)*(1-cb)/0.5
 * cgv=(255/224)*65536*(cr/cg)*(1-cr)/0.5
 *
 * where Y=cr*R+cg*G+cb*B (cr+cg+cb=1)
 */

EXTERN int convmat[8][4]
#ifdef GLOBAL
=
{
    { 117504, 138453, 13954, 34903 }, /* no sequence_display_extension */
    { 117504, 138453, 13954, 34903 }, /* ITU-R Rec. 709 (1990) */
    { 104597, 132201, 25675, 53279 }, /* unspecified */
    { 104597, 132201, 25675, 53279 }, /* reserved */
    { 104448, 132798, 24759, 53109 }, /* FCC */
    { 104597, 132201, 25675, 53279 }, /* ITU-R Rec. 624-4 System B, G */
    { 104597, 132201, 25675, 53279 }, /* SMPTE 170M */
    { 117579, 136230, 16907, 35559 } /* SMPTE 240M (1987) */
}
#endif

```



```

;

EXTERN int quiet;
EXTERN int trace;
EXTERN char errortext[256];
EXTERN unsigned char *refframe[3],*oldrefframe[3],*auxframe[3],*newframe[3];
EXTERN unsigned char *clp;
EXTERN int profile, level;
EXTERN int horizontal_size,vertical_size,mb_width,mb_height;
EXTERN int coded_picture_width, coded_picture_height;
EXTERN int chroma_format,chrom_width,chrom_height,blk_cnt;
EXTERN int pict_type;
EXTERN int forw_r_size,back_r_size;
EXTERN int full_forw,full_back;
EXTERN int fault;
EXTERN int verbose;
EXTERN int prog_seq;
EXTERN int h_forw_r_size,v_forw_r_size,h_back_r_size,v_back_r_size;
EXTERN int dc_prec,pict_struct,topfirst,frame_pred_dct,conceal_mv;
EXTERN int intravlc,repeatfirst,prog_frame;
EXTERN int secondfield;
EXTERN int stwc_table_index,llw,llh,hm,hn,vm,vn;
EXTERN int ltempref,llx0,lly0,llprog_frame,llfieldsel;
EXTERN unsigned char *llframe0[3],*llframe1[3];
EXTERN short *lltmp;
EXTERN char *linputname;
EXTERN int twostreams,sflag;
EXTERN int refidct;
EXTERN int matrix_coefficients;
/* output */
EXTERN int framestoreflag;
EXTERN char *outputname;
EXTERN int outtype, hiQdither;
#define T_YUV 0
#define T_SIF 1
#define T_TGA 2
#define T_PPM 3
#define T_X11 4
#define T_X11HIQ 5

/* layer specific variables (needed for SNR and DP scalability) */
EXTERN struct layer_data {
/* bit input */
int infile;
unsigned char rdbfr[2048];
unsigned char *rdptr;
unsigned char inbfr[16];
int incnt;
int bitcnt;
/* sequence header */
int intra_quantizer_matrix[64],non_intra_quantizer_matrix[64];
int chroma_intra_quantizer_matrix[64],chroma_non_intra_quantizer_matrix[64];
int mpeg2;
/* sequence scalable extension */
int scalable_mode;
/* picture coding extension */
int qscale_type,altscan;
/* picture spatial scalable extension */
int pict_scal;

```

```

/* slice/macroblock */
int pri_brk;
int quant_scale;
int intra_slice;
short block[12][64];
} base, enhan, *ld;

/* some of these (currently unused) variables are layer dependent! */
EXTERN int low_delay;
EXTERN int video_format; /* initialize to -1 for compliance test */
EXTERN int last_temp_ref; /* compliance test variable */
EXTERN int temp_ref, vbv_delay;
EXTERN int picture_rate, aspect_ratio, vbv_buffer_size;
EXTERN int last_pict_type; /* compliance test variable */
EXTERN int colour_primaries;
EXTERN int transfer_characteristics;
EXTERN int display_horizontal_size, display_vertical_size;
EXTERN int bit_rate_value;
EXTERN float bit_rate;

```

idct.c

```

/* idct.c, inverse fast discrete cosine transform */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

/*****
 * inverse two dimensional DCT, Chen-Wang algorithm */
 * (cf. IEEE ASSP-32, pp. 803-816, Aug. 1984) */
 * 32-bit integer arithmetic (8 bit coefficients) */
 * 11 mults, 29 adds per DCT */

```

```

/*                      sE, 18.8.91      */
/*****
/* coefficients extended to 12 bit for IEEE1180-1990      */
/* compliance                      sE, 2.1.94      */
/*****

/* this code assumes >> to be a two's-complement arithmetic */
/* right shift: (-2)>>1 == -1 , (-3)>>1 == -2      */

#include "config.h"

#define W1 2841 /* 2048*sqrt(2)*cos(1*pi/16) */
#define W2 2676 /* 2048*sqrt(2)*cos(2*pi/16) */
#define W3 2408 /* 2048*sqrt(2)*cos(3*pi/16) */
#define W5 1609 /* 2048*sqrt(2)*cos(5*pi/16) */
#define W6 1108 /* 2048*sqrt(2)*cos(6*pi/16) */
#define W7 565  /* 2048*sqrt(2)*cos(7*pi/16) */

/* global declarations */
void init_idct _ANSI_ARGS_((void));
void idct _ANSI_ARGS_((short *block));

/* private data */
static short iclip[1024]; /* clipping table */
static short *iclp;

/* private prototypes */
static void idctrow _ANSI_ARGS_((short *blk));
static void idctcol _ANSI_ARGS_((short *blk));

/* row (horizontal) IDCT
*
*      7          pi      1
* dst[k] = sum c[l] * src[l] * cos( -- * ( k + - ) * 1 )
*      l=0          8      2
*
* where: c[0] = 128
*      c[1..7] = 128*sqrt(2)
*/

static void idctrow(blk)
short *blk;
{
    int x0, x1, x2, x3, x4, x5, x6, x7, x8;

    /* shortcut */
    if (!(x1 = blk[4]<<11) | (x2 = blk[6]) | (x3 = blk[2]) |
        (x4 = blk[1]) | (x5 = blk[7]) | (x6 = blk[5]) | (x7 = blk[3]))
    {
        blk[0]=blk[1]=blk[2]=blk[3]=blk[4]=blk[5]=blk[6]=blk[7]=blk[0]<<3;
        return;
    }

    x0 = (blk[0]<<11) + 128; /* for proper rounding in the fourth stage */

    /* first stage */
    x8 = W7*(x4+x5);
    x4 = x8 + (W1-W7)*x4;
    x5 = x8 - (W1+W7)*x5;

```

```

x8 = W3*(x6+x7);
x6 = x8 - (W3-W5)*x6;
x7 = x8 - (W3+W5)*x7;

```

```

/* second stage */

```

```

x8 = x0 + x1;
x0 -= x1;
x1 = W6*(x3+x2);
x2 = x1 - (W2+W6)*x2;
x3 = x1 + (W2-W6)*x3;
x1 = x4 + x6;
x4 -= x6;
x6 = x5 + x7;
x5 -= x7;

```

```

/* third stage */

```

```

x7 = x8 + x3;
x8 -= x3;
x3 = x0 + x2;
x0 -= x2;
x2 = (181*(x4+x5)+128)>>8;
x4 = (181*(x4-x5)+128)>>8;

```

```

/* fourth stage */

```

```

blk[0] = (x7+x1)>>8;
blk[1] = (x3+x2)>>8;
blk[2] = (x0+x4)>>8;
blk[3] = (x8+x6)>>8;
blk[4] = (x8-x6)>>8;
blk[5] = (x0-x4)>>8;
blk[6] = (x3-x2)>>8;
blk[7] = (x7-x1)>>8;
}

```

```

/* column (vertical) IDCT

```

```

*
*          7          pi          1
* dst[8*k] = sum c[l] * src[8*l] * cos( -- * ( k + - ) * l )
*          l=0          8          2
*
* where: c[0] = 1/1024
*          c[1..7] = (1/1024)*sqrt(2)
*/

```

```

static void idctcol(blk)

```

```

short *blk;

```

```

{
    int x0, x1, x2, x3, x4, x5, x6, x7, x8;

```

```

/* shortcut */

```

```

if (!(x1 = (blk[8*4]<<8)) | (x2 = blk[8*6]) | (x3 = blk[8*2]) |
    (x4 = blk[8*1]) | (x5 = blk[8*7]) | (x6 = blk[8*5]) | (x7 = blk[8*3]))
{
    blk[8*0]=blk[8*1]=blk[8*2]=blk[8*3]=blk[8*4]=blk[8*5]=blk[8*6]=blk[8*7]=
    iclp[(blk[8*0]+32)>>6];
    return;
}

```

```

x0 = (blk[8*0]<<8) + 8192;

```

```

/* first stage */
x8 = W7*(x4+x5) + 4;
x4 = (x8+(W1-W7)*x4)>>3;
x5 = (x8-(W1+W7)*x5)>>3;
x8 = W3*(x6+x7) + 4;
x6 = (x8-(W3-W5)*x6)>>3;
x7 = (x8-(W3+W5)*x7)>>3;

/* second stage */
x8 = x0 + x1;
x0 -= x1;
x1 = W6*(x3+x2) + 4;
x2 = (x1-(W2+W6)*x2)>>3;
x3 = (x1+(W2-W6)*x3)>>3;
x1 = x4 + x6;
x4 -= x6;
x6 = x5 + x7;
x5 -= x7;

/* third stage */
x7 = x8 + x3;
x8 -= x3;
x3 = x0 + x2;
x0 -= x2;
x2 = (181*(x4+x5)+128)>>8;
x4 = (181*(x4-x5)+128)>>8;

/* fourth stage */
blk[8*0] = iclp[(x7+x1)>>14];
blk[8*1] = iclp[(x3+x2)>>14];
blk[8*2] = iclp[(x0+x4)>>14];
blk[8*3] = iclp[(x8+x6)>>14];
blk[8*4] = iclp[(x8-x6)>>14];
blk[8*5] = iclp[(x0-x4)>>14];
blk[8*6] = iclp[(x3-x2)>>14];
blk[8*7] = iclp[(x7-x1)>>14];
}

/* two dimensional inverse discrete cosine transform */
void idct(block)
short *block;
{
    int i;

    for (i=0; i<8; i++)
        idctrow(block+8*i);

    for (i=0; i<8; i++)
        idctcol(block+i);
}

void init_idct()
{
    int i;

    iclp = iclip+512;
    for (i= -512; i<512; i++)
        iclp[i] = (i<-256) ? -256 : ((i>255) ? 255 : i);
}

```

idctf.c

/\* idctf.c, inverse fast discrete cosine transform, floating point version \*/

/\* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims  
 \* any and all warranties, whether express, implied, or statutory, 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.

\*

\* The MPEG Software Simulation Group 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.

\*

\*/

#include <math.h>

#include "config.h"

#define W1 1.38703984532 /\* sqrt(2)\*cos(1\*pi/16) \*/

#define W2 1.30656296487 /\* sqrt(2)\*cos(2\*pi/16) \*/

#define W3 1.17587560242 /\* sqrt(2)\*cos(3\*pi/16) \*/

#define W4 0.707106781187 /\* cos(4\*pi/16) \*/

#define W5 0.785694958387 /\* sqrt(2)\*cos(5\*pi/16) \*/

#define W6 0.541196100153 /\* sqrt(2)\*cos(6\*pi/16) \*/

#define W7 0.275899379291 /\* sqrt(2)\*cos(7\*pi/16) \*/

/\* global declarations \*/

void init\_idct \_ANSI\_ARGS\_\_((void));

void idct \_ANSI\_ARGS\_\_((short \*block));

/\* private data \*/

static short iclip[1024]; /\* clipping table \*/

static short \*iclp;

static double temp[64];

/\* private prototypes \*/

static void idctrow \_ANSI\_ARGS\_\_((short \*src, double \*dst));

```
static void idtctcol _ANSI_ARGS_((double *src, short *dst));
```

```
/* row (horizontal) IDCT
```

```
*/
```

```

*          7          pi          1
* dst[k] = sum c[l] * src[l] * cos( -- * ( k + - ) * 1 )
*          l=0          8          2

```

```
*/
```

```
/* where: c[0] = 128
```

```

*          c[1..7] = 128*sqrt(2)
*/

```

```
*/
```

```
static void idctrow(src,dst)
```

```
short *src;
```

```
double *dst;
```

```
{
```

```
double x0, x1, x2, x3, x4, x5, x6, x7, x8;
```

```
x0 = src[0];
```

```
x1 = src[4];
```

```
x2 = src[6];
```

```
x3 = src[2];
```

```
x4 = src[1];
```

```
x5 = src[7];
```

```
x6 = src[5];
```

```
x7 = src[3];
```

```
/* first stage */
```

```
x8 = W7*(x4+x5);
```

```
x4 = x8 + (W1-W7)*x4;
```

```
x5 = x8 - (W1+W7)*x5;
```

```
x8 = W3*(x6+x7);
```

```
x6 = x8 - (W3-W5)*x6;
```

```
x7 = x8 - (W3+W5)*x7;
```

```
/* second stage */
```

```
x8 = x0 + x1;
```

```
x0 -= x1;
```

```
x1 = W6*(x3+x2);
```

```
x2 = x1 - (W2+W6)*x2;
```

```
x3 = x1 + (W2-W6)*x3;
```

```
x1 = x4 + x6;
```

```
x4 -= x6;
```

```
x6 = x5 + x7;
```

```
x5 -= x7;
```

```
/* third stage */
```

```
x7 = x8 + x3;
```

```
x8 -= x3;
```

```
x3 = x0 + x2;
```

```
x0 -= x2;
```

```
x2 = W4*(x4+x5);
```

```
x4 = W4*(x4-x5);
```

```
/* fourth stage */
```

```
dst[0] = x7 + x1;
```

```
dst[1] = x3 + x2;
```

```
dst[2] = x0 + x4;
```

```
dst[3] = x8 + x6;
```

```

dst[4] = x8 - x6;
dst[5] = x0 - x4;
dst[6] = x3 - x2;
dst[7] = x7 - x1;
}

/* column (vertical) IDCT
*
*      7      pi      1
* dst[8*k] = sum c[l] * src[8*l] * cos( -- * ( k + - ) * 1 )
*      l=0      8      2
*
* where: c[0] = 1/1024
*      c[1..7] = (1/1024)*sqrt(2)
*/
static void idctcol(src,dst)
double *src;
short *dst;
{
    double x0, x1, x2, x3, x4, x5, x6, x7, x8;

    x0 = src[8*0];
    x1 = src[8*4];
    x2 = src[8*6];
    x3 = src[8*2];
    x4 = src[8*1];
    x5 = src[8*7];
    x6 = src[8*5];
    x7 = src[8*3];

    /* first stage */
    x8 = W7*(x4+x5);
    x4 = x8 + (W1-W7)*x4;
    x5 = x8 - (W1+W7)*x5;
    x8 = W3*(x6+x7);
    x6 = x8 - (W3-W5)*x6;
    x7 = x8 - (W3+W5)*x7;

    /* second stage */
    x8 = x0 + x1;
    x0 -= x1;
    x1 = W6*(x3+x2);
    x2 = x1 - (W2+W6)*x2;
    x3 = x1 + (W2-W6)*x3;
    x1 = x4 + x6;
    x4 -= x6;
    x6 = x5 + x7;
    x5 -= x7;

    /* third stage */
    x7 = x8 + x3;
    x8 -= x3;
    x3 = x0 + x2;
    x0 -= x2;
    x2 = W4*(x4+x5);
    x4 = W4*(x4-x5);

    /* fourth stage */
    dst[8*0] = iclp[(int)floor(0.125*(x7+x1)+0.5)];

```



```

dst[8*1] = iclp[(int)floor(0.125*(x3+x2)+0.5)];
dst[8*2] = iclp[(int)floor(0.125*(x0+x4)+0.5)];
dst[8*3] = iclp[(int)floor(0.125*(x8+x6)+0.5)];
dst[8*4] = iclp[(int)floor(0.125*(x8-x6)+0.5)];
dst[8*5] = iclp[(int)floor(0.125*(x0-x4)+0.5)];
dst[8*6] = iclp[(int)floor(0.125*(x3-x2)+0.5)];
dst[8*7] = iclp[(int)floor(0.125*(x7-x1)+0.5)];
}

/* two dimensional inverse discrete cosine transform */
void idct(block)
short *block;
{
    int i;

    for (i=0; i<8; i++)
        idctrow(block+8*i,temp+8*i);

    for (i=0; i<8; i++)
        idctcol(temp+i,block+i);
}

void init_idct()
{
    int i;

    iclp = iclip+512;
    for (i= -512; i<512; i++)
        iclp[i] = (i<-256) ? -256 : ((i>255) ? 255 : i);
}

idctref.c

/* idctref.c, Inverse Discrete Fourier Transform, double precision */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
*
*/

```

```

/* Perform IEEE 1180 reference (64-bit floating point, separable 8x1
* direct matrix multiply) Inverse Discrete Cosine Transform
*/

```

```

/* Here we use math.h to generate constants. Compiler results may
vary a little */

```

```

#include <math.h>

```

```

#include "config.h"

```

```

#ifndef PI
# ifdef M_PI
#  define PI M_PI
# else
#  define PI 3.14159265358979323846
# endif
#endif

```

```

/* global declarations */
void init_idctref _ANSI_ARGS__((void));
void idctref _ANSI_ARGS__((short *block));

```

```

/* private data */

```

```

/* cosine transform matrix for 8x1 IDCT */
static double c[8][8];

```

```

/* initialize DCT coefficient matrix */

```

```

void init_idctref()
{
    int freq, time;
    double scale;

    for (freq=0; freq < 8; freq++)
    {
        scale = (freq == 0) ? sqrt(0.125) : 0.5;
        for (time=0; time<8; time++)
            c[freq][time] = scale*cos((PI/8.0)*freq*(time + 0.5));
    }
}

```

```

/* perform IDCT matrix multiply for 8x8 coefficient block */

```

```

void idctref(block)
short *block;
{
    int i, j, k, v;
    double partial_product;
    double tmp[64];

    for (i=0; i<8; i++)

```

```

    for (j=0; j<8; j++)
    {
        partial_product = 0.0;

        for (k=0; k<8; k++)
            partial_product+= c[k][j]*block[8*i+k];

        tmp[8*i+j] = partial_product;
    }

/* Transpose operation is integrated into address mapping by switching
   loop order of i and j */

for (j=0; j<8; j++)
for (i=0; i<8; i++)
{
    partial_product = 0.0;

    for (k=0; k<8; k++)
        partial_product+= c[k][i]*tmp[8*k+j];

    v = floor(partial_product+0.5);
    block[8*i+j] = (v<-256) ? -256 : ((v>255) ? 255 : v);
}
}

makefile

# Makefile for mpeg2decode

# Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
# any and all warranties, whether express, implied, or statutory, 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.
#
# The MPEG Software Simulation Group 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.
#
#

```

```
# uncomment the following two lines if you want to include X11 support
```

```
#USE_DISP = -DDISPLAY
```

```
#LIBS = -lX11
```

```
# uncomment the following two lines if you want to use shared memory
```

```
# (faster display if server and client run on the same machine)
```

```
#USE_SHMEM = -DSH_MEM
```

```
#LIBS = -Xext -lX11
```

```
# if your X11 include files / libraries are in a non standard location:
```

```
# set INCLUDEDIR to -I followed by the appropriate include file path and
```

```
# set LIBRARYDIR to -L followed by the appropriate library path and
```

```
#INCLUDEDIR = -I/usr/openwin/include
```

```
#LIBRARYDIR = -L/usr/openwin/lib
```

```
# select one of the following CC CFLAGS settings
```

```
#
```

```
# GNU gcc
```

```
#
```

```
CC = gcc
```

```
CFLAGS = -O2 $(USE_DISP) $(USE_SHMEM) $(INCLUDEDIR)
```

```
#
```

```
# SPARCworks acc
```

```
#
```

```
#CC = acc
```

```
#CFLAGS = -fast $(USE_DISP) $(USE_SHMEM) $(INCLUDEDIR)
```

```
#
```

```
# SUN cc
```

```
#
```

```
#CC = cc
```

```
#CFLAGS = -O3 -DNON_ANSI_COMPILER $(USE_DISP) $(USE_SHMEM) $(INCLUDEDIR)
```

```
# the following are user contributed configurations
```

```
# DEC Alpha cc
```

```
#CC = cc
```

```
#CFLAGS = -O -Olimit 1000
```

```
# HP715
```

```
#CC = cc
```

```
#CFLAGS = -Aa -D_HPUX_SOURCE
```

```
# SGI Irix 5.2
```

```
#CC = cc
```

```
#CFLAGS = -O2 -sopt
```

```
OBJ = mpeg2dec.o getpic.o motion.o getvlc.o gethdr.o getblk.o getbits.o store.o recon.o spatasc.o idct.o  
idctref.o display.o
```

```
all: mpeg2decode
```

```
pc: mpeg2dec.exe
```

clean:

```
rm -f *.o *% core mpeg2decode
```

```
mpeg2dec.exe: mpeg2decode
               coff2exe mpeg2dec
```

```
mpeg2decode: $(OBJ)
              $(CC) $(CFLAGS) $(LIBRARYDIR) -o mpeg2decode $(OBJ) -lm $(LIBS)
```

```
display.o : display.c config.h global.h mpeg2dec.h
getbits.o : getbits.c config.h global.h mpeg2dec.h
getblk.o : getblk.c config.h global.h mpeg2dec.h
gethdr.o : gethdr.c config.h global.h mpeg2dec.h
getpic.o : getpic.c config.h global.h mpeg2dec.h
getvlc.o : getvlc.c config.h global.h mpeg2dec.h getvlc.h
idct.o : idct.c config.h
idctref.o : idctref.c config.h
motion.o : motion.c config.h global.h mpeg2dec.h
mpeg2dec.o : mpeg2dec.c config.h global.h mpeg2dec.h
recon.o : recon.c config.h global.h mpeg2dec.h
spatscal.o : spatscal.c config.h global.h mpeg2dec.h
store.o : store.c config.h global.h mpeg2dec.h
```

motion.c

```
/* motion.c, motion vector decoding
```

```
*/
```

```
/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
* any and all warranties, whether express, implied, or statutory, 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.
```

```
*
```

```
* The MPEG Software Simulation Group 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.
```

```
*
```

```
*/
```

```
#include <stdio.h>
```

```
#include "config.h"
```

```

#include "global.h"

/* private prototypes */
static void calcMV_ANSI_ARGS_((int *pred, int r_size, int motion_code,
    int motion_r, int full_pel_vector));

void motion_vectors(PMV,dmvector,
    mv_field_sel,s,mv_count,mv_format,h_r_size,v_r_size,dmv,mvscale)
int PMV[2][2][2];
int dmvector[2];
int mv_field_sel[2][2];
int s, mv_count, mv_format, h_r_size, v_r_size, dmvector, mvscale;
{
    if (mv_count==1)
    {
        if (mv_format==MV_FIELD && !dmvector)
        {
            mv_field_sel[1][s] = mv_field_sel[0][s] = getbits(1);
            if (trace)
            {
                printf("motion_vertical_field_select[][%d] (%d): %d\n",s,
                    mv_field_sel[0][s],mv_field_sel[0][s]);
            }
        }
    }

    motion_vector(PMV[0][s],dmvector,h_r_size,v_r_size,dmv,mvscale,0);

    /* update other motion vector predictors */
    PMV[1][s][0] = PMV[0][s][0];
    PMV[1][s][1] = PMV[0][s][1];
}
else
{
    mv_field_sel[0][s] = getbits(1);
    if (trace)
    {
        printf("motion_vertical_field_select[0][%d] (%d): %d\n",s,
            mv_field_sel[0][s],mv_field_sel[0][s]);
    }
    motion_vector(PMV[0][s],dmvector,h_r_size,v_r_size,dmv,mvscale,0);

    mv_field_sel[1][s] = getbits(1);
    if (trace)
    {
        printf("motion_vertical_field_select[1][%d] (%d): %d\n",s,
            mv_field_sel[1][s],mv_field_sel[1][s]);
    }
    motion_vector(PMV[1][s],dmvector,h_r_size,v_r_size,dmv,mvscale,0);
}
}

/* get and decode motion vector and differential motion vector */
void motion_vector(PMV,dmvector,
    h_r_size,v_r_size,dmv,mvscale,full_pel_vector)
int *PMV;
int *dmvector;
int h_r_size;
int v_r_size;
int dmvector; /* MPEG-2 only: get differential motion vectors */

```

```

int mvscale; /* MPEG-2 only: field vector in frame pic */
int full_pel_vector; /* MPEG-1 only */
{
    int motion_code,motion_r;

    motion_code = getMV();
    motion_r = (h_r_size!=0 && motion_code!=0) ? getbits(h_r_size) : 0;

    if (trace)
    {
        if (h_r_size!=0 && motion_code!=0)
        {
            printf("motion_residual (");
            printbits(motion_r,h_r_size,h_r_size);
            printf("): %d\n",motion_r);
        }
    }

    calcMV(&PMV[0],h_r_size,motion_code,motion_r,full_pel_vector);

    if (dmv)
        dmvector[0] = getDMV();

    motion_code = getMV();
    motion_r = (v_r_size!=0 && motion_code!=0) ? getbits(v_r_size) : 0;

    if (trace)
    {
        if (v_r_size!=0 && motion_code!=0)
        {
            printf("motion_residual (");
            printbits(motion_r,v_r_size,v_r_size);
            printf("): %d\n",motion_r);
        }
    }

    if (mvscale)
        PMV[1] >>= 1; /* DIV 2 */

    calcMV(&PMV[1],v_r_size,motion_code,motion_r,full_pel_vector);

    if (mvscale)
        PMV[1] <<= 1;

    if (dmv)
        dmvector[1] = getDMV();

    if (trace)
        printf("PMV = %d,%d\n",PMV[0],PMV[1]);
}

/* calculate motion vector component */
static void calcMV(pred,r_size,motion_code,motion_r,full_pel_vector)
int *pred;
int r_size, motion_code, motion_r, full_pel_vector;
{
    int lim, vec;

    lim = 16<<r_size;

```

```

vec = full_pel_vector ? (*pred >> 1) : (*pred);

if (motion_code>0)
{
    vec+= ((motion_code-1)<<r_size) + motion_r + 1;
    if (vec>=lim)
        vec-= lim + lim;
}
else if (motion_code<0)
{
    vec-= ((-motion_code-1)<<r_size) + motion_r + 1;
    if (vec<=-lim)
        vec+= lim + lim;
}
*pred = full_pel_vector ? (vec<<1) : vec;
}

void calc_DMV(DMV,dmvector,mvx,mvy)
int DMV[][2];
int *dmvector; /* differential motion vector */
int mvx, mvy; /* decoded mv components (always in field format) */
{
    if (pict_struct==FRAME_PICTURE)
    {
        if (topfirst)
        {
            /* vector for prediction of top field from bottom field */
            DMV[0][0] = ((mvx + (mvx>0))>>1) + dmvector[0];
            DMV[0][1] = ((mvy + (mvy>0))>>1) + dmvector[1] - 1;

            /* vector for prediction of bottom field from top field */
            DMV[1][0] = ((3*mvx+(mvx>0))>>1) + dmvector[0];
            DMV[1][1] = ((3*mvy+(mvy>0))>>1) + dmvector[1] + 1;
        }
        else
        {
            /* vector for prediction of top field from bottom field */
            DMV[0][0] = ((3*mvx+(mvx>0))>>1) + dmvector[0];
            DMV[0][1] = ((3*mvy+(mvy>0))>>1) + dmvector[1] - 1;

            /* vector for prediction of bottom field from top field */
            DMV[1][0] = ((mvx + (mvx>0))>>1) + dmvector[0];
            DMV[1][1] = ((mvy + (mvy>0))>>1) + dmvector[1] + 1;
        }
    }
    else
    {
        /* vector for prediction from field of opposite 'parity' */
        DMV[0][0] = ((mvx+(mvx>0))>>1) + dmvector[0];
        DMV[0][1] = ((mvy+(mvy>0))>>1) + dmvector[1];

        /* correct for vertical field shift */
        if (pict_struct==TOP_FIELD)
            DMV[0][1]--;
        else
            DMV[0][1]++;
    }
}

```



mpeg2dec.c

```

/* mpeg2dec.c, main(), initialization, option processing */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>

#define GLOBAL
#include "config.h"
#include "global.h"

/* private prototypes */
static void initdecoder _ANSI_ARGS_((void));
static void options _ANSI_ARGS_((int *argcp, char **argvp[]));
static int getval _ANSI_ARGS_((char *argv[]));

int main(argc,argv)
int argc;
char *argv[];
{
    int first, framenum;

    options(&argc,&argv);

    /* pointer to name of output files */
#ifdef DISPLAY
    if (outtype==T_X11)

```

```

    outputname = "";
else
#endif
    outputname = argv[argc-1];

ld = &base; /* select base layer context */

/* open MPEG input file(s) */
if ((base.infile=open(argv[1],O_RDONLY|O_BINARY))<0)
{
    sprintf(errortext,"Input file %s not found\n",argv[1]);
    error(errortext);
}

initbits();

if (argc==4)
{
    ld = &enhan; /* select enhancement layer context */

    if ((enhan.infile = open(argv[2],O_RDONLY|O_BINARY))<0)
    {
        sprintf(errortext,"Input file %s not found\n",argv[2]);
        error(errortext);
    }

    twostreams = 1;
    initbits();
    ld = &base;
}

first = 1;
framenum = 0;

while (getheader())
{
    if (first)
    {
        initdecoder();
        first = 0;
    }

    getpicture(framenum);

    if (!secondfield)
        framenum++;
}

if (framenum!=0)
{
    /* put last frame */
    putlast(framenum);
}

close(base.infile);

if (twostreams)
    close(enhan.infile);

```

```

#ifdef DISPLAY
    if (outtype==T_X11)
        exit_display();
#endif

    return 0;
}

static void initdecoder()
{
    int i, cc, size;
    static int blk_cnt_tab[3] = {6,8,12};

    /* check scalability mode of enhancement layer */
    if (twostreams && enhan.scalable_mode!=SC_SNR &&
        !(base.scalable_mode==SC_DP && base.scalable_mode==SC_DP))
        error("unsupported scalability mode\n");

    /* clip table */
    if (!(clp=(unsigned char *)malloc(1024)))
        error("malloc failed\n");

    clp += 384;

    for (i=-384; i<640; i++)
        clp[i] = (i<0) ? 0 : ((i>255) ? 255 : i);

    /* force MPEG-1 parameters */
    if (!base.mpeg2)
    {
        prog_seq = 1;
        prog_frame = 1;
        pict_struct = FRAME_PICTURE;
        frame_pred_dct = 1;
        chroma_format = CHROMA420;
        matrix_coefficients = 5;
    }

    /* round to nearest multiple of coded macroblocks */
    mb_width = (horizontal_size+15)/16;
    mb_height = (base.mpeg2 && !prog_seq) ? 2*((vertical_size+31)/32)
        : (vertical_size+15)/16;
    coded_picture_width = 16*mb_width;
    coded_picture_height = 16*mb_height;

    chrom_width = (chroma_format==CHROMA444) ? coded_picture_width
        : coded_picture_width>>1;
    chrom_height = (chroma_format!=CHROMA420) ? coded_picture_height
        : coded_picture_height>>1;
    blk_cnt = blk_cnt_tab[chroma_format-1];

    for (cc=0; cc<3; cc++)
    {
        if (cc==0)
            size = coded_picture_width*coded_picture_height;
        else
            size = chrom_width*chrom_height;

        if (!(refframe[cc] = (unsigned char *)malloc(size)))

```

```

    error("malloc failed\n");

    if (!(oldreframe[cc] = (unsigned char *)malloc(size)))
        error("malloc failed\n");

    if (!(auxframe[cc] = (unsigned char *)malloc(size)))
        error("malloc failed\n");

    if (base.scalable_mode==SC_SPAT)
    {
        /* this assumes lower layer is 4:2:0 */
        if (!(llframe0[cc] = (unsigned char *)malloc((llw*llh)/(cc?4:1))))
            error("malloc failed\n");
        if (!(llframe1[cc] = (unsigned char *)malloc((llw*llh)/(cc?4:1))))
            error("malloc failed\n");
    }
}

if (base.scalable_mode==SC_SPAT)
{
    if (!(lltmp = (short *)malloc(llw*((llh*vn)/vm)*sizeof(short))))
        error("malloc failed\n");
}

/* IDCT */
if (refidct)
    init_idctref();
else
    init_idct();

#ifdef DISPLAY
    if (outtype==T_X11)
    {
        init_display("");
        init_dither();
    }
#endif
}

void error(text)
char *text;
{
    fprintf(stderr,text);
    exit(1);
}

/* trace output */
void printbits(code,bits,len)
int code,bits,len;
{
    int i;
    for (i=0; i<len; i++)
        printf("%d", (code>>(bits-1-i))&1);
}

/* option processing */
static void options(argcp,argvp)
int *argcp;
char **argvp[];

```

```

{
while (*argcp>1 && (*argv)[1][0]!='-')
{
while ((*argv)[1][1])
{
switch (toupper((*argv)[1][1]))
{
case 'V':
verbose = getval(*argv);
break;
case 'O':
outtype = getval(*argv);
#ifdef DISPLAY
if (outtype==T_X11HIQ)
{
hiQdither = 1;
outtype=T_X11;
}
#endif
break;
case 'F':
framestoreflag = 1;
break;
case 'S':
sflag = 1;
break;
case 'R':
refidct = 1;
break;
case 'T':
trace = 1;
break;
case 'Q':
quiet = 1;
break;
default:
fprintf(stderr,"undefined option -%c ignored\n",(*argv)[1][1]);
}

(*argv)[1]++;
}

(*argv)++;
(*argcp)--;
}

if (sflag)
{
/* input file for spatial prediction */
linputname = (*argv)[1];
(*argv)++;
(*argcp)--;
}

#ifdef DISPLAY
if (outtype==T_X11)
{
framestoreflag = 1; /* two avoid calling dither() twice */
(*argcp)++; /* fake outfile parameter */

```

```

    }
#endif

    if (*argcp!=3 && *argcp!=4)
    {
        printf("\n%s, %s\n",version,author);
        printf("Usage:  mpeg2decode {options} input.m2v {upper.m2v} {outfile}\n\
Options: -vn  verbose output (n: level)\n\
        -on  output format (0: YUV, 1: SIF, 2: TGA, 3:PPM, 4:X11, 5:X11 HiQ)\n\
        -f   store interlaced video in frame format\n\
        -q   disable warnings to stderr\n\
        -r   use double precision reference IDCT\n\
        -s   infile  spatial scalable sequence\n\
        -t   enable low level tracing\n");
        exit(0);
    }
}

static int getval(argv)
char *argv[];
{
    int val;

    if (sscanf(argv[1]+2,"%d",&val)!=1)
        return 0;

    while (isdigit(argv[1][2]))
        argv[1]++;

    return val;
}

mpeg2dec.h

/* mpeg2dec.h, MPEG specific defines */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
*
*/

```

```

#define ERROR (-1)

```

```

#define PICTURE_START_CODE 0x100
#define SLICE_MIN_START 0x101
#define SLICE_MAX_START 0x1AF
#define USER_START_CODE 0x1B2
#define SEQ_START_CODE 0x1B3
#define EXT_START_CODE 0x1B5
#define SEQ_END_CODE 0x1B7
#define GOP_START_CODE 0x1B8

```

```

/* scalable_mode */
#define SC_NONE 0
#define SC_DP 1
#define SC_SPAT 2
#define SC_SNR 3
#define SC_TEMP 4

```

```

/* picture coding type */
#define I_TYPE 1
#define P_TYPE 2
#define B_TYPE 3
#define D_TYPE 4

```

```

/* picture structure */
#define TOP_FIELD 1
#define BOTTOM_FIELD 2
#define FRAME_PICTURE 3

```

```

/* macroblock type */
#define MB_INTRA 1
#define MB_PATTERN 2
#define MB_BACKWARD 4
#define MB_FORWARD 8
#define MB_QUANT 16
#define MB_WEIGHT 32
#define MB_CLASS4 64

```

```

/* motion_type */
#define MC_FIELD 1
#define MC_FRAME 2
#define MC_16X8 2
#define MC_DMV 3

```

```

/* mv_format */
#define MV_FIELD 0
#define MV_FRAME 1

```

```

/* chroma_format */
#define CHROMA420 1
#define CHROMA422 2
#define CHROMA444 3

```

```

/* extension start code IDs */

```

```
#define SEQ_ID    1
#define DISP_ID   2
#define QUANT_ID  3
#define SEQSCAL_ID 5
#define PANSCAN_ID 7
#define CODING_ID  8
#define SPATSCAL_ID 9
#define TEMPSCAL_ID 10
```

```
recon.c
```

```
/* recon.c, motion compensation routines
```

```
*/
```

```
/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
```

```
 *
```

```
 * The MPEG Software Simulation Group 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.
```

```
 *
```

```
*/
```

```
#include <stdio.h>
```

```
#include "config.h"
```

```
#include "global.h"
```

```
/* private prototypes */
```

```
static void recon _ANSI_ARGS_((unsigned char *src[], int sfield,
    unsigned char *dst[], int dfield,
    int lx, int lx2, int w, int h, int x, int y, int dx, int dy,
    int addflag));
```

```
static void recon_comp _ANSI_ARGS_((unsigned char *src, unsigned char *dst,
    int lx, int lx2, int w, int h, int x, int y, int dx, int dy, int addflag));
```

```
void reconstruct(bx,by,mb_type,motion_type,PMV,mv_field_sel,dmvector,stwtype)
int bx, by;
int mb_type;
int motion_type;
int PMV[2][2][2], mv_field_sel[2][2], dmvector[2];
```



```

int stwtype;
{
    int currentfield;
    unsigned char **predframe;
    int DMV[2][2];
    int stwtop, stwbot;

    stwtop = stwtype%3; /* 0:temporal, 1:(spat+temp)/2, 2:spatial */
    stwbot = stwtype/3;

    if ((mb_type & MB_FORWARD) || (pict_type==P_TYPE))
    {
        if (pict_struct==FRAME_PICTURE)
        {
            if ((motion_type==MC_FRAME) || !(mb_type & MB_FORWARD))
            {
                /* frame-based prediction */
                if (stwtop<2)
                    recon(oldrefframe,0,newframe,0,
                        coded_picture_width,coded_picture_width<<1,16,8,bx,by,
                        PMV[0][0][0],PMV[0][0][1],stwtop);

                if (stwbot<2)
                    recon(oldrefframe,1,newframe,1,
                        coded_picture_width,coded_picture_width<<1,16,8,bx,by,
                        PMV[0][0][0],PMV[0][0][1],stwbot);
            }
            else if (motion_type==MC_FIELD) /* field-based prediction */
            {
                /* top field prediction */
                if (stwtop<2)
                    recon(oldrefframe,mv_field_sel[0][0],newframe,0,
                        coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
                        PMV[0][0][0],PMV[0][0][1]>>1,stwtop);

                /* bottom field prediction */
                if (stwbot<2)
                    recon(oldrefframe,mv_field_sel[1][0],newframe,1,
                        coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
                        PMV[1][0][0],PMV[1][0][1]>>1,stwbot);
            }
            else if (motion_type==MC_DMV) /* dual prime prediction */
            {
                /* calculate derived motion vectors */
                calc_DMV(DMV,dmvector,PMV[0][0][0],PMV[0][0][1]>>1);

                if (stwtop<2)
                {
                    /* predict top field from top field */
                    recon(oldrefframe,0,newframe,0,
                        coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
                        PMV[0][0][0],PMV[0][0][1]>>1,0);

                    /* predict and add to top field from bottom field */
                    recon(oldrefframe,1,newframe,0,
                        coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
                        DMV[0][0],DMV[0][1],1);
                }
            }
        }
    }
}

```

```

if (stwb0t<2)
{
/* predict bottom field from bottom field */
recon(oldrefframe,1,newframe,1,
coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
PMV[0][0][0],PMV[0][0][1]>>1,0);

/* predict and add to bottom field from top field */
recon(oldrefframe,0,newframe,1,
coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
DMV[1][0],DMV[1][1],1);
}
}
else
/* invalid motion_type */
printf("invalid motion_type\n");
}
else /* TOP_FIELD or BOTTOM_FIELD */
{
/* field picture */
currentfield = (pict_struct==BOTTOM_FIELD);

/* determine which frame to use for prediction */
if ((pict_type==P_TYPE) && secondfield
&& (currentfield!=mv_field_sel[0][0]))
predframe = refframe; /* same frame */
else
predframe = oldrefframe; /* previous frame */

if ((motion_type==MC_FIELD) || !(mb_type & MB_FORWARD))
{
/* field-based prediction */
if (stwt0p<2)
recon(predframe,mv_field_sel[0][0],newframe,0,
coded_picture_width<<1,coded_picture_width<<1,16,16,bx,by,
PMV[0][0][0],PMV[0][0][1],stwt0p);
}
else if (motion_type==MC_16X8)
{
if (stwt0p<2)
{
recon(predframe,mv_field_sel[0][0],newframe,0,
coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by,
PMV[0][0][0],PMV[0][0][1],stwt0p);

/* determine which frame to use for lower half prediction */
if ((pict_type==P_TYPE) && secondfield
&& (currentfield!=mv_field_sel[1][0]))
predframe = refframe; /* same frame */
else
predframe = oldrefframe; /* previous frame */

recon(predframe,mv_field_sel[1][0],newframe,0,
coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by+8,
PMV[1][0][0],PMV[1][0][1],stwt0p);
}
}
}
else if (motion_type==MC_DMV) /* dual prime prediction */
{

```

```

    if (secondfield)
        predframe = refframe; /* same frame */
    else
        predframe = oldrefframe; /* previous frame */

    /* calculate derived motion vectors */
    calc_DMV(DMV,dmvector,PMV[0][0][0],PMV[0][0][1]);

    /* predict from field of same parity */
    recon(oldrefframe,currentfield,newframe,0,
        coded_picture_width<<1,coded_picture_width<<1,16,16,bx,by,
        PMV[0][0][0],PMV[0][0][1],0);

    /* predict from field of opposite parity */
    recon(predframe,!currentfield,newframe,0,
        coded_picture_width<<1,coded_picture_width<<1,16,16,bx,by,
        DMV[0][0],DMV[0][1],1);
}
else
    /* invalid motion_type */
    printf("invalid motion_type\n");
}
stwtop = stwbot = 1;
}

if (mb_type & MB_BACKWARD)
{
    if (pict_struct==FRAME_PICTURE)
    {
        if (motion_type==MC_FRAME)
        {
            /* frame-based prediction */
            if (stwtop<2)
                recon(refframe,0,newframe,0,
                    coded_picture_width,coded_picture_width<<1,16,8,bx,by,
                    PMV[0][1][0],PMV[0][1][1],stwtop);

            if (stwbot<2)
                recon(refframe,1,newframe,1,
                    coded_picture_width,coded_picture_width<<1,16,8,bx,by,
                    PMV[0][1][0],PMV[0][1][1],stwbot);
        }
        else /* field-based prediction */
        {
            /* top field prediction */
            if (stwtop<2)
                recon(refframe,mv_field_sel[0][1],newframe,0,
                    coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
                    PMV[0][1][0],PMV[0][1][1]>>1,stwtop);

            /* bottom field prediction */
            if (stwbot<2)
                recon(refframe,mv_field_sel[1][1],newframe,1,
                    coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by>>1,
                    PMV[1][1][0],PMV[1][1][1]>>1,stwbot);
        }
    }
}
else /* TOP_FIELD or BOTTOM_FIELD */
{

```

```

/* field picture */
if (motion_type==MC_FIELD)
{
    /* field-based prediction */
    recon(refframe,mv_field_sel[0][1],newframe,0,
        coded_picture_width<<1,coded_picture_width<<1,16,16,bx,by,
        PMV[0][1][0],PMV[0][1][1],stwtot);
}
else if (motion_type==MC_16X8)
{
    recon(refframe,mv_field_sel[0][1],newframe,0,
        coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by,
        PMV[0][1][0],PMV[0][1][1],stwtot);

    recon(refframe,mv_field_sel[1][1],newframe,0,
        coded_picture_width<<1,coded_picture_width<<1,16,8,bx,by+8,
        PMV[1][1][0],PMV[1][1][1],stwtot);
}
else
    /* invalid motion_type */
    printf("invalid motion_type\n");
}
}
}

```

```

static void recon(src,sfield,dst,dfield,lx,lx2,w,h,x,y,dx,dy,addflag)
unsigned char *src[]; /* prediction source buffer */
int sfield; /* prediction source field number (0 or 1) */
unsigned char *dst[]; /* prediction destination buffer */
int dfield; /* prediction destination field number (0 or 1) */
int lx,lx2; /* horizontal offsets */
int w,h; /* prediction block/sub-block width, height */
int x,y; /* pixel co-ordinates of top-left sample in current MB */
int dx,dy; /* horizontal, vertical motion vector */
int addflag; /* add prediction error to prediction ? */
{
    /* Y */
    recon_comp(src[0]+(sfield?lx2>>1:0),dst[0]+(dfield?lx2>>1:0),
        lx,lx2,w,h,x,y,dx,dy,addflag);

    if (chroma_format!=CHROMA444)
    {
        lx>>=1; lx2>>=1; w>>=1; x>>=1; dx/=2;
    }

    if (chroma_format==CHROMA420)
    {
        h>>=1; y>>=1; dy/=2;
    }

    /* Cb */
    recon_comp(src[1]+(sfield?lx2>>1:0),dst[1]+(dfield?lx2>>1:0),
        lx,lx2,w,h,x,y,dx,dy,addflag);
    /* Cr */
    recon_comp(src[2]+(sfield?lx2>>1:0),dst[2]+(dfield?lx2>>1:0),
        lx,lx2,w,h,x,y,dx,dy,addflag);
}

```

```
static void recon_comp(src,dst,lx,lx2,w,h,x,y,dx,dy,addflag)
```

```

unsigned char *src;
unsigned char *dst;
int lx,lx2;
int w,h;
int x,y;
int dx,dy;
int addflag;
{
    int xint, xh, yint, yh;
    int i, j, v;
    unsigned char *s, *d;

    /* half pel scaling */
    xint = dx>>1;
    xh = dx & 1;
    yint = dy>>1;
    yh = dy & 1;

    /* origins */
    s = src + lx*(y+yint) + x + xint;
    d = dst + lx*y + x;

    if (!xh && !yh)
        if (addflag)
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                {
                    v = d[i]+s[i];
                    d[i] = (v+(v>=0?1:0))>>1;
                }
                s+= lx2;
                d+= lx2;
            }
        else
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = s[i];
                s+= lx2;
                d+= lx2;
            }
    else if (!xh && yh)
        if (addflag)
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                {
                    v = d[i] + ((unsigned int)(s[i]+s[i+lx]+1))>>1;
                    d[i]=(v+(v>=0?1:0))>>1;
                }
                s+= lx2;
                d+= lx2;
            }
        else
            for (j=0; j<h; j++)
            {
                for (i=0; i<w; i++)
                    d[i] = (unsigned int)(s[i]+s[i+lx]+1)>>1;
            }

```

```

        s+= 1x2;
        d+= 1x2;
    }
else if (xh && !yh)
    if (addflag)
        for (j=0; j<h; j++)
        {
            for (i=0; i<w; i++)
            {
                v = d[i] + ((unsigned int)(s[i]+s[i+1]+1)>>1);
                d[i] = (v+(v>=0?1:0))>>1;
            }
            s+= 1x2;
            d+= 1x2;
        }
else
    for (j=0; j<h; j++)
    {
        for (i=0; i<w; i++)
            d[i] = (unsigned int)(s[i]+s[i+1]+1)>>1;
        s+= 1x2;
        d+= 1x2;
    }
else /* if (xh && yh) */
    if (addflag)
        for (j=0; j<h; j++)
        {
            for (i=0; i<w; i++)
            {
                v = d[i] + ((unsigned int)(s[i]+s[i+1]+s[i+1x]+s[i+1x+1]+2)>>2);
                d[i] = (v+(v>=0?1:0))>>1;
            }
            s+= 1x2;
            d+= 1x2;
        }
else
    for (j=0; j<h; j++)
    {
        for (i=0; i<w; i++)
            d[i] = (unsigned int)(s[i]+s[i+1]+s[i+1x]+s[i+1x+1]+2)>>2;
        s+= 1x2;
        d+= 1x2;
    }
}

```

spatscal.c

```

/* spatscal.c, spatial scalability decoding */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
*
* The MPEG Software Simulation Group 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.
*
*/

```

```

#include <stdio.h>
#include "config.h"
#include "global.h"

```

```

/* private prototypes */
static void spatpred _ANSI_ARGS_((int prog_frame, int llprog_frame,
unsigned char *fld0, unsigned char *fld1, short *tmp, unsigned char *dst,
int llx0, int lly0, int llw, int llh, int horizontal_size, int vertical_size,
int vm, int vn, int hm, int hn, int aperture));
static void deinterlace _ANSI_ARGS_((unsigned char *fld0, unsigned char *fld1,
int j0, int lx, int ly, int aperture));
static void subv _ANSI_ARGS_((unsigned char *s, short *d,
int lx, int lys, int lyd, int m, int n, int j0, int dj));
static void subh _ANSI_ARGS_((short *s, unsigned char *d,
int x0, int lx, int lxs, int lxd, int ly, int m, int n));

```

```

/* get reference frame */
void getspatref()
{
    int i, j, llw2, llh2;
    FILE *fd;
    char fname[80];

    llw2 = llw>>1;
    llh2 = llh>>1;

    sprintf(fname, llinputname, lltempref, 'a');
    strcat(fname, ".Y");
    if (verbose>1)
        printf("reading %s\n", fname);
    fd=fopen(fname, "rb");
    for (j=0; j<llh; j+=2)
        for (i=0; i<llw; i++)
            llframe0[0][llw*j+i]=getc(fd);
    fclose(fd);

    sprintf(fname, llinputname, lltempref, 'b');
    strcat(fname, ".Y");
    if (verbose>1)
        printf("reading %s\n", fname);
}

```

```

fd=fopen(fname,"rb");
for (j=1; j<llh; j+=2)
    for (i=0; i<llw; i++)
        llframe1[0][llw*j+i]=getc(fd);
fclose(fd);

sprintf(fname,llinputname,lltempref,'a');
strcat(fname,".U");
if (verbose>1)
    printf("reading %s\n",fname);
fd=fopen(fname,"rb");
for (j=0; j<llh2; j+=2)
    for (i=0; i<llw2; i++)
        llframe0[1][llw2*j+i]=getc(fd);
fclose(fd);

sprintf(fname,llinputname,lltempref,'b');
strcat(fname,".U");
if (verbose>1)
    printf("reading %s\n",fname);
fd=fopen(fname,"rb");
for (j=1; j<llh2; j+=2)
    for (i=0; i<llw2; i++)
        llframe1[1][llw2*j+i]=getc(fd);
fclose(fd);

sprintf(fname,llinputname,lltempref,'a');
strcat(fname,".V");
if (verbose>1)
    printf("reading %s\n",fname);
fd=fopen(fname,"rb");
for (j=0; j<llh2; j+=2)
    for (i=0; i<llw2; i++)
        llframe0[2][llw2*j+i]=getc(fd);
fclose(fd);

sprintf(fname,llinputname,lltempref,'b');
strcat(fname,".V");
if (verbose>1)
    printf("reading %s\n",fname);
fd=fopen(fname,"rb");
for (j=1; j<llh2; j+=2)
    for (i=0; i<llw2; i++)
        llframe1[2][llw2*j+i]=getc(fd);
fclose(fd);

spatpred(prog_frame,llprog_frame,llframe0[0],llframe1[0],lltmp,newframe[0],
    llx0,lly0,llw,llh,horizontal_size,vertical_size,vm,vn,hm,hn,
    pict_struct!=FRAME_PICTURE); /* this changed from CD to DIS */
spatpred(prog_frame,llprog_frame,llframe0[1],llframe1[1],lltmp,newframe[1],
    llx0/2,lly0/2,llw2,llh2,horizontal_size>>1,vertical_size>>1,vm,vn,hm,hn,1);
spatpred(prog_frame,llprog_frame,llframe0[2],llframe1[2],lltmp,newframe[2],
    llx0/2,lly0/2,llw2,llh2,horizontal_size>>1,vertical_size>>1,vm,vn,hm,hn,1);
}

/* form spatial prediction */
static void spatpred(prog_frame,llprog_frame,
    fld0,fld1,tmp,dst,llx0,lly0,llw,llh,horizontal_size,vertical_size,
    vm,vn,hm,hn,aperture)

```



```

int prog_frame,llprog_frame;
unsigned char *fld0,*fld1;
short *tmp;
unsigned char *dst;
int llx0,lly0,llw,llh,horizontal_size,vertical_size,vm,vn,hm,hn,aperture;
{
    int w, h, x0, llw2, llh2;
#ifdef 0
    if (llprog_frame)
    {
        /* progressive -> progressive / interlaced */
        subv(fld0,tmp,horizontal_size,vertical_size,m,n,0,1);
        subh(tmp,dst,0,horizontal_size,horizontal_size,vertical_size,m,n);
    }
    else if (prog_frame)
    {
        /* interlaced -> progressive */
        if (ll_fldsel)
        {
            deinterlace(fld1,fld0);
            subv(fld1,tmp);
            subh(tmp,dst);
        }
        else
        {
            deinterlace(fld0,fld1);
            subv(fld0,tmp);
            subh(tmp,dst);
        }
    }
    else
    {
#endif

        /* interlaced -> interlaced */
        llw2 = (llw*hn)/hm;
        llh2 = (llh*vn)/vm;
        deinterlace(fld0,fld1,1,llw,llh,aperture);
        deinterlace(fld1,fld0,0,llw,llh,aperture);
        subv(fld0,tmp,llw,llh,llh2,vm,vn,0,2);
        subv(fld1,tmp,llw,llh,llh2,vm,vn,1,2);

        /* vertical limits */
        if (lly0<0)
        {
            tmp-= llw*lly0;
            llh2+= lly0;
            if (llh2<0)
                llh2 = 0;
            h = (vertical_size<llh2) ? vertical_size : llh2;
        }
        else
        {
            dst+= horizontal_size*lly0;
            h= vertical_size - lly0;
            if (h>llh2)
                h = llh2;
        }
}

```

```

/* horizontal limits */
if (llx0<0)
{
    x0 = -llx0;
    llw2+= llx0;
    if (llw2<0)
        llw2 = 0;
    w = (horizontal_size<llw2) ? horizontal_size : llw2;
}
else
{
    dst+= llx0;
    x0 = 0;
    w = horizontal_size - llx0;
    if (w>llw2)
        w = llw2;
}
subh(tmp,dst,x0,w,llw,horizontal_size,h,hm,hn);

#if 0
}
#endif
}

/* deinterlace one field (interpolate opposite parity samples)
*
* deinterlacing is done in-place: if j0=1, fld0 contains the input field in
* its even lines and the odd lines are interpolated by this routine
* if j0=0, the input field is in the odd lines and the even lines are
* interpolated
*
* fld0: field to be deinterlaced
* fld1: other field (referenced by the two field aperture filter)
* j0: 0: interpolate even (top) lines, 1: interpolate odd (bottom) lines
* lx: width of fld0 and fld1
* ly: height of the deinterlaced field (has to be even)
* aperture: 1: use one field aperture filter (two field otherwise)
*/
static void deinterlace(fld0,fld1,j0,lx,ly,aperture)
unsigned char *fld0,*fld1;
int j0,lx,ly; /* ly has to be even */
int aperture;
{
    int i,j,v;
    unsigned char *p0, *p0m1, *p0p1, *p1, *p1m2, *p1p2;

    /* deinterlace one field */
    for (j=j0; j<ly; j+=2)
    {
        p0 = fld0+lx*j;
        p0m1 = (j==0) ? p0+lx : p0-lx;
        p0p1 = (j==ly-1) ? p0-lx : p0+lx;

        if (aperture)
            for (i=0; i<lx; i++)
                p0[i] = (unsigned int)(p0m1[i] + p0p1[i] + 1)>>1;
        else
        {
            p1 = fld1 + lx*j;

```

```

    p1m2 = (j<2) ? p1 : p1-2*lx;
    p1p2 = (j>=ly-2) ? p1 : p1+2*lx;
    for (i=0; i<lx; i++)
    {
        v = 8*(p0m1[i]+p0p1[i]) + 2*p1[i] - p1m2[i] - p1p2[i];
        p0[i] = clp[(v + ((v>=0) ? 8 : 7))>>4];
    }
}
}
}

```

```

/* vertical resampling */
static void subv(s,d,lx,lys,lyd,m,n,j0,dj)
unsigned char *s;
short *d;
int lx, lys, lyd, m, n, j0, dj;
{
    int i, j, c1, c2, jd;
    unsigned char *s1, *s2;
    short *d1;

    for (j=j0; j<lyd; j+=dj)
    {
        d1 = d + lx*j;
        jd = (j*m)/n;
        s1 = s + lx*jd;
        s2 = (jd<lys-1)? s1+lx : s1;
        c2 = (16*((j*m)%n) + (n>>1))/n;
        c1 = 16 - c2;
        for (i=0; i<lx; i++)
            d1[i] = c1*s1[i] + c2*s2[i];
    }
}

```

```

/* horizontal resampling */
static void subh(s,d,x0,lx,lxs,lxd,ly,m,n)
short *s;
unsigned char *d;
int x0, lx, lxs, lxd, ly, m, n;
{
    int i, i1, j, id, c1, c2, v;
    short *s1, *s2;
    unsigned char *d1;

    for (i1=0; i1<lx; i1++)
    {
        d1 = d + i1;
        i = x0 + i1;
        id = (i*m)/n;
        s1 = s+id;
        s2 = (id<lx-1) ? s1+1 : s1;
        c2 = (16*((i*m)%n) + (n>>1))/n;
        c1 = 16 - c2;
        for (j=0; j<ly; j++)
        {
            v = c1*(s1) + c2*(s2);
            *d1 = (v + ((v>=0) ? 128 : 127))>>8;
            d1+= lxd;
            s1+= lxs;
        }
    }
}

```

```

        s2+= 1xs;
    }
}
}

```

store.c

```

/* store.c, picture output routines */

/* Copyright (C) 1994, MPEG Software Simulation Group. 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. The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statutory, 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.
 *
 * The MPEG Software Simulation Group 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.
 */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

#include "config.h"
#include "global.h"

/* private prototypes */
static void store_one _ANSI_ARGS_((char *outname, unsigned char *src[],
    int offset, int incr, int height));
static void store_yuv _ANSI_ARGS_((char *outname, unsigned char *src[],
    int offset, int incr, int height));
static void store_sif _ANSI_ARGS_((char *outname, unsigned char *src[],
    int offset, int incr, int height));
static void store_ppm_tga _ANSI_ARGS_((char *outname, unsigned char *src[],
    int offset, int incr, int height, int tgaflag));
static void store_yuv1 _ANSI_ARGS_((char *name, unsigned char *src,
    int offset, int incr, int width, int height));
static void putbyte _ANSI_ARGS_((int c));
static void putword _ANSI_ARGS_((int w));
static void conv422to444 _ANSI_ARGS_((unsigned char *src, unsigned char *dst));

```

```

static void conv420to422 _ANSI_ARGS__((unsigned char *src, unsigned char *dst));

#define OBFRSIZE 4096
static unsigned char obfr[OBFRSIZE];
static unsigned char *optr;
static int outfile;

/*
 * store a picture as either one frame or two fields
 */
void storeframe(src,frame)
unsigned char *src[];
int frame;
{
    char outname[32];

    if (prog_seq || prog_frame || framestoreflag)
    {
        /* progressive */
        sprintf(outname,outputname,frame,'f');
        store_one(outname,src,0,coded_picture_width,vertical_size);
    }
    else
    {
        /* interlaced */
        sprintf(outname,outputname,frame,'a');
        store_one(outname,src,0,coded_picture_width<<1,vertical_size>>1);

        sprintf(outname,outputname,frame,'b');
        store_one(outname,src,
            coded_picture_width,coded_picture_width<<1,vertical_size>>1);
    }
}

/*
 * store one frame or one field
 */
static void store_one(outname,src,offset,incr,height)
char *outname;
unsigned char *src[];
int offset, incr, height;
{
    switch (outtype)
    {
    case T_YUV:
        store_yuv(outname,src,offset,incr,height);
        break;
    case T_SIF:
        store_sif(outname,src,offset,incr,height);
        break;
    case T_TGA:
        store_ppm_tga(outname,src,offset,incr,height,1);
        break;
    case T_PPM:
        store_ppm_tga(outname,src,offset,incr,height,0);
        break;
#ifdef DISPLAY
    case T_X11:
        dither(src);

```

```

    break;
#endif
    default:
        break;
    }
}

/* separate headerless files for y, u and v */
static void store_yuv(outname,src,offset,incr,height)
char *outname;
unsigned char *src[];
int offset,incr,height;
{
    int hsize;
    char tmpname[32];

    hsize = horizontal_size;

    sprintf(tmpname,"%s.Y",outname);
    store_yuv1(tmpname,src[0],offset,incr,hsize,height);

    if (chroma_format!=CHROMA444)
    {
        offset>>=1; incr>>=1; hsize>>=1;
    }

    if (chroma_format==CHROMA420)
    {
        height>>=1;
    }

    sprintf(tmpname,"%s.U",outname);
    store_yuv1(tmpname,src[1],offset,incr,hsize,height);

    sprintf(tmpname,"%s.V",outname);
    store_yuv1(tmpname,src[2],offset,incr,hsize,height);
}

/* auxiliary routine */
static void store_yuv1(name,src,offset,incr,width,height)
char *name;
unsigned char *src;
int offset,incr,width,height;
{
    int i, j;
    unsigned char *p;

    if (!quiet)
        fprintf(stderr,"saving %s\n",name);

    if ((outfile = open(name,O_CREAT|O_TRUNC|O_WRONLY|O_BINARY,0666))== -1)
    {
        sprintf(errortext,"Couldn't create %s\n",name);
        error(errortext);
    }

    optr=obfr;

    for (i=0; i<height; i++)

```

```

    {
        p = src + offset + incr*i;
        for (j=0; j<width; j++)
            putbyte(*p++);
    }

    if (optr!=obfr)
        write(outfile,obfr,optr-obfr);

    close(outfile);
}

/*
 * store as headerless file in U,Y,V,Y format
 */
static void store_sif (outname,src,offset,incr,height)
char *outname;
unsigned char *src[];
int offset, incr, height;
{
    int i,j;
    unsigned char *py, *pu, *pv;
    static unsigned char *u422, *v422;

    if (chroma_format==CHROMA444)
        error("4:4:4 not supported for SIF format");

    if (chroma_format==CHROMA422)
    {
        u422 = src[1];
        v422 = src[2];
    }
    else
    {
        if (!u422)
        {
            if (!(u422 = (unsigned char *)malloc((coded_picture_width>>1)
                                                *coded_picture_height)))
                error("malloc failed");
            if (!(v422 = (unsigned char *)malloc((coded_picture_width>>1)
                                                *coded_picture_height)))
                error("malloc failed");
        }

        conv420to422(src[1],u422);
        conv420to422(src[2],v422);
    }

    strcat(outname, ".SIF");

    if (!quiet)
        fprintf(stderr,"saving %s\n",outname);

    if ((outfile = open(outname,O_CREAT|O_TRUNC|O_WRONLY|O_BINARY,0666))===-1)
    {
        sprintf(errortext,"Couldn't create %s\n",outname);
        error(errortext);
    }
}

```

```

optr = obfr;

for (i=0; i<height; i++)
{
    py = src[0] + offset + incr*i;
    pu = u422 + (offset>>1) + (incr>>1)*i;
    pv = v422 + (offset>>1) + (incr>>1)*i;

    for (j=0; j<horizontal_size; j+=2)
    {
        putbyte(*pu++);
        putbyte(*py++);
        putbyte(*pv++);
        putbyte(*py++);
    }
}

if (optr!=obfr)
    write(outfile,obfr,optr-obfr);

close(outfile);
}

/*
 * store as PPM (PBMPLUS) or uncompressed Truevision TGA ('Targa') file
 */
static void store_ppm_tga(outname,src,offset,incr,height,tgaflag)
char *outname;
unsigned char *src[];
int offset, incr, height;
int tgaflag;
{
    int i, j;
    int y, u, v, r, g, b;
    int crv, cbu, cgu, cgV;
    unsigned char *py, *pu, *pv;
    static unsigned char tga24[14] = {0,0,2,0,0,0,0, 0,0,0,0,0,24,32};
    char header[32];
    static unsigned char *u422, *v422, *u444, *v444;

    if (chroma_format==CHROMA444)
    {
        u444 = src[1];
        v444 = src[2];
    }
    else
    {
        if (!u444)
        {
            if (chroma_format==CHROMA420)
            {
                if (!(u422 = (unsigned char *)malloc(((coded_picture_width>>1)
                    *coded_picture_height))))
                    error("malloc failed");
                if (!(v422 = (unsigned char *)malloc(((coded_picture_width>>1)
                    *coded_picture_height))))
                    error("malloc failed");
            }
        }
    }
}

```



```

    if (! (u444 = (unsigned char *) malloc(coded_picture_width
                                           * coded_picture_height)))
        error("malloc failed");

    if (! (v444 = (unsigned char *) malloc(coded_picture_width
                                           * coded_picture_height)))
        error("malloc failed");
}

if (chroma_format == CHROMA420)
{
    conv420to422(src[1], u422);
    conv420to422(src[2], v422);
    conv422to444(u422, u444);
    conv422to444(v422, v444);
}
else
{
    conv422to444(src[1], u444);
    conv422to444(src[2], v444);
}
}

strcat(outname, tgaflag ? ".tga" : ".ppm");

if (!quiet)
    fprintf(stderr, "saving %s\n", outname);

if ((outfile = open(outname, O_CREAT|O_TRUNC|O_WRONLY|O_BINARY, 0666)) == -1)
{
    sprintf(errortext, "Couldn't create %s\n", outname);
    error(errortext);
}

optr = obfr;

if (tgaflag)
{
    /* TGA header */
    for (i=0; i<12; i++)
        putbyte(tga24[i]);

    putword(horizontal_size); putword(height);
    putbyte(tga24[12]); putbyte(tga24[13]);
}
else
{
    /* PPM header */
    sprintf(header, "P6\n%d %d\n255\n", horizontal_size, height);

    for (i=0; header[i] != 0; i++)
        putbyte(header[i]);
}

/* matrix coefficients */
crv = convmat[matrix_coefficients][0];
cbu = convmat[matrix_coefficients][1];
cgu = convmat[matrix_coefficients][2];
cgv = convmat[matrix_coefficients][3];

```

```

for (i=0; i<height; i++)
{
    py = src[0] + offset + incr*i;
    pu = u444 + offset + incr*i;
    pv = v444 + offset + incr*i;

    for (j=0; j<horizontal_size; j++)
    {
        u = *pu++ - 128;
        v = *pv++ - 128;
        y = 76309 * (*py++ - 16); /* (255/219)*65536 */
        r = clp[(y + crv*v + 32768)>>16];
        g = clp[(y - cgu*u - cg*v + 32768)>>16];
        b = clp[(y + cbu*u + 32786)>>16];

        if (tgaflag)
        {
            putbyte(b); putbyte(g); putbyte(r);
        }
        else
        {
            putbyte(r); putbyte(g); putbyte(b);
        }
    }
}

if (optr!=obfr)
    write(outfile,obfr,optr-obfr);

close(outfile);
}

static void putbyte(c)
int c;
{
    *optr++ = c;

    if (optr == obfr+OBFRSIZE)
    {
        write(outfile,obfr,OBFRSIZE);
        optr = obfr;
    }
}

static void putword(w)
int w;
{
    putbyte(w); putbyte(w>>8);
}

/* horizontal 1:2 interpolation filter */
static void conv422to444(src,dst)
unsigned char *src,*dst;
{
    int i, i2, w, j, im3, im2, im1, ip1, ip2, ip3;

    w = coded_picture_width>>1;

```

```

if (base.mpeg2)
{
    for (j=0; j<coded_picture_height; j++)
    {
        for (i=0; i<w; i++)
        {
            i2 = i<<1;
            im2 = (i<2) ? 0 : i-2;
            im1 = (i<1) ? 0 : i-1;
            ip1 = (i<w-1) ? i+1 : w-1;
            ip2 = (i<w-2) ? i+2 : w-1;
            ip3 = (i<w-3) ? i+3 : w-1;

            /* FIR filter coefficients (*256): 21 0 -52 0 159 256 159 0 -52 0 21 */
            /* even samples (0 0 256 0 0) */
            dst[i2] = src[i];

            /* odd samples (21 -52 159 159 -52 21) */
            dst[i2+1] = clp[(int)(21*(src[im2]+src[ip3])
                                -52*(src[im1]+src[ip2])
                                +159*(src[i]+src[ip1])+128)>>8];
        }
        src+= w;
        dst+= coded_picture_width;
    }
}
else
{
    for (j=0; j<coded_picture_height; j++)
    {
        for (i=0; i<w; i++)
        {
            i2 = i<<1;
            im3 = (i<3) ? 0 : i-3;
            im2 = (i<2) ? 0 : i-2;
            im1 = (i<1) ? 0 : i-1;
            ip1 = (i<w-1) ? i+1 : w-1;
            ip2 = (i<w-2) ? i+2 : w-1;
            ip3 = (i<w-3) ? i+3 : w-1;

            /* FIR filter coefficients (*256): 5 -21 70 228 -37 11 */
            dst[i2] = clp[(int)( 5*src[im3]
                                -21*src[im2]
                                +70*src[im1]
                                +228*src[i]
                                -37*src[ip1]
                                +11*src[ip2]+128)>>8];

            dst[i2+1] = clp[(int)( 5*src[ip3]
                                -21*src[ip2]
                                +70*src[ip1]
                                +228*src[i]
                                -37*src[im1]
                                +11*src[im2]+128)>>8];
        }
        src+= w;
        dst+= coded_picture_width;
    }
}

```

```

    }
}

/* vertical 1:2 interpolation filter */
static void conv420to422(src,dst)
unsigned char *src,*dst;
{
    int w, h, i, j, j2;
    int jm6, jm5, jm4, jm3, jm2, jm1, jp1, jp2, jp3, jp4, jp5, jp6, jp7;

    w = coded_picture_width>>1;
    h = coded_picture_height>>1;

    if (prog_frame)
    {
        /* intra frame */
        for (i=0; i<w; i++)
        {
            for (j=0; j<h; j++)
            {
                j2 = j<<1;
                jm3 = (j<3) ? 0 : j-3;
                jm2 = (j<2) ? 0 : j-2;
                jm1 = (j<1) ? 0 : j-1;
                jp1 = (j<h-1) ? j+1 : h-1;
                jp2 = (j<h-2) ? j+2 : h-1;
                jp3 = (j<h-3) ? j+3 : h-1;

                /* FIR filter coefficients (*256): 5 -21 70 228 -37 11 */
                /* New FIR filter coefficients (*256): 3 -16 67 227 -32 7 */
                dst[w*j2] = clp[(int)( 3*src[w*jm3]
                    -16*src[w*jm2]
                    +67*src[w*jm1]
                    +227*src[w*j]
                    -32*src[w*jp1]
                    +7*src[w*jp2]+128)>>8];

                dst[w*(j2+1)] = clp[(int)( 3*src[w*jp3]
                    -16*src[w*jp2]
                    +67*src[w*jp1]
                    +227*src[w*j]
                    -32*src[w*jm1]
                    +7*src[w*jm2]+128)>>8];
            }
            src++;
            dst++;
        }
    }
    else
    {
        /* intra field */
        for (i=0; i<w; i++)
        {
            for (j=0; j<h; j+=2)
            {
                j2 = j<<1;

                /* top field */
                jm6 = (j<6) ? 0 : j-6;

```

```

jm4 = (j<4) ? 0 : j-4;
jm2 = (j<2) ? 0 : j-2;
jp2 = (j<h-2) ? j+2 : h-2;
jp4 = (j<h-4) ? j+4 : h-2;
jp6 = (j<h-6) ? j+6 : h-2;

/* Polyphase FIR filter coefficients (*256): 2 -10 35 242 -18 5 */
/* New polyphase FIR filter coefficients (*256): 1 -7 30 248 -21 5 */
dst[w*j2] = clp[(int)( 1*src[w*jm6]
-7*src[w*jm4]
+30*src[w*jm2]
+248*src[w*j]
-21*src[w*jp2]
+5*src[w*jp4]+128)>>8];

/* Polyphase FIR filter coefficients (*256): 11 -38 192 113 -30 8 */
/* New polyphase FIR filter coefficients (*256):7 -35 194 110 -24 4 */
dst[w*(j2+2)] = clp[(int)( 7*src[w*jm4]
-35*src[w*jm2]
+194*src[w*j]
+110*src[w*jp2]
-24*src[w*jp4]
+4*src[w*jp6]+128)>>8];

/* bottom field */
jm5 = (j<5) ? 1 : j-5;
jm3 = (j<3) ? 1 : j-3;
jm1 = (j<1) ? 1 : j-1;
jp1 = (j<h-1) ? j+1 : h-1;
jp3 = (j<h-3) ? j+3 : h-1;
jp5 = (j<h-5) ? j+5 : h-1;
jp7 = (j<h-7) ? j+7 : h-1;

/* Polyphase FIR filter coefficients (*256): 11 -38 192 113 -30 8 */
/* New polyphase FIR filter coefficients (*256):7 -35 194 110 -24 4 */
dst[w*(j2+1)] = clp[(int)( 7*src[w*jp5]
-35*src[w*jp3]
+194*src[w*jp1]
+110*src[w*jm1]
-24*src[w*jm3]
+4*src[w*jm5]+128)>>8];

dst[w*(j2+3)] = clp[(int)( 1*src[w*jp7]
-7*src[w*jp5]
+30*src[w*jp3]
+248*src[w*jp1]
-21*src[w*jm1]
+5*src[w*jm3]+128)>>8];
}
src++;
dst++;
}
}
}

```

# Annex C

## Audio source listing

### C.1 Introduction

**Table C.1 List of audio files**

filename	description
1cb0, 1cb1, 1cb2, 2cb0, 2cb1, 2cb2	Critical band rate tables
1th0, 1th1, 1th2, 2th0, 2th1, 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
dewindow	Decoder subband filter bank window table
enwindow	Encoder subband filter bank window table
common.c	Global variable functions and definitions, read AIFF routine, CRC, and low-level I/O routines
common.h	Global conditional compiler switches and defaults
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
musicin.c	Prompts for encoder user input parameters and performs initialization
musicout.c	Decoder parameter initialization
predisto.c	Encoder pre-distortion routines
psy.c	Psychoacoustic model routines
subs.c	FFT subroutine
tonal.c	Model I for layers I and II

## C.2 Tables

1cb0	1cb2	26
26	25	0 1
0 1	0 1	1 3
1 2	1 2	2 5
2 3	2 4	3 7
3 4	3 5	4 9
4 5	4 7	5 11
5 7	5 9	6 14
6 8	6 11	7 17
7 10	7 13	8 20
8 11	8 15	9 24
9 13	9 18	10 27
10 15	10 21	11 32
11 17	11 24	12 37
12 20	12 28	13 42
13 23	13 32	14 49
14 27	14 37	15 57
15 32	15 44	16 69
16 38	16 51	17 81
17 45	17 61	18 97
18 53	18 73	19 113
19 63	19 87	20 137
20 75	20 101	21 161
21 89	21 121	22 193
22 105	22 145	23 241
23 133	23 181	24 329
24 177	24 241	25 433
25 233	2cb0	2cb2
1cb1	26	25
26	0 1	0 1
0 1	1 3	1 4
1 2	2 5	2 6
2 3	3 8	3 10
3 4	4 10	4 14
4 5	5 13	5 17
5 6	6 16	6 21
6 7	7 19	7 25
7 9	8 22	8 30
8 10	9 26	9 35
9 12	10 30	10 41
10 14	11 34	11 47
11 16	12 40	12 55
12 19	13 46	13 63
13 21	14 53	14 73
14 25	15 63	15 87
15 29	16 75	16 101
16 35	17 89	17 121
17 41	18 105	18 145
18 49	19 125	19 173
19 57	20 149	20 201
20 69	21 177	21 241
21 81	22 209	22 289
22 97	23 265	23 353
23 121	24 353	24 481
24 165	25 465	1th0
25 217	2cb1	107

1 1 0.850 25.87	58 68 19.464 1.93	7 7 6.041 5.00
2 2 1.694 14.85	59 70 19.635 2.11	8 8 6.770 4.45
3 3 2.525 10.72	60 72 19.801 2.28	9 9 7.457 4.00
4 4 3.337 8.50	61 74 19.963 2.45	10 10 8.103 3.61
5 5 4.124 7.10	62 76 20.120 2.63	11 11 8.708 3.26
6 6 4.882 6.11	63 78 20.273 2.82	12 12 9.275 2.93
7 7 5.608 5.37	64 80 20.421 3.03	13 13 9.805 2.63
8 8 6.301 4.79	65 82 20.565 3.25	14 14 10.301 2.32
9 9 6.959 4.32	66 84 20.705 3.49	15 15 10.765 2.02
10 10 7.581 3.92	67 86 20.840 3.74	16 16 11.199 1.71
11 11 8.169 3.57	68 88 20.971 4.02	17 17 11.606 1.38
12 12 8.723 3.25	69 90 21.099 4.32	18 18 11.988 1.04
13 13 9.244 2.95	70 92 21.222 4.64	19 19 12.347 0.67
14 14 9.734 2.67	71 94 21.341 4.98	20 20 12.684 0.29
15 15 10.195 2.39	72 96 21.457 5.35	21 21 13.002 -0.11
16 16 10.629 2.11	73 100 21.676 6.15	22 22 13.302 -0.54
17 17 11.037 1.83	74 104 21.882 7.07	23 23 13.586 -0.97
18 18 11.421 1.53	75 108 22.074 8.10	24 24 13.855 -1.43
19 19 11.783 1.23	76 112 22.253 9.25	25 25 14.111 -1.88
20 20 12.125 0.90	77 116 22.420 10.54	26 26 14.354 -2.34
21 21 12.448 0.56	78 120 22.575 11.97	27 27 14.585 -2.79
22 22 12.753 0.21	79 124 22.721 13.56	28 28 14.807 -3.22
23 23 13.042 -0.17	80 128 22.857 15.30	29 29 15.018 -3.62
24 24 13.317 -0.56	81 132 22.984 17.23	30 30 15.221 -3.98
25 25 13.577 -0.96	82 136 23.102 19.33	31 31 15.415 -4.30
26 26 13.825 -1.37	83 140 23.213 21.64	32 32 15.602 -4.57
27 27 14.062 -1.79	84 144 23.317 24.15	33 33 15.783 -4.77
28 28 14.288 -2.21	85 148 23.414 26.88	34 34 15.956 -4.91
29 29 14.504 -2.63	86 152 23.506 29.84	35 35 16.124 -4.98
30 30 14.711 -3.03	87 156 23.592 33.04	36 36 16.287 -4.97
31 31 14.909 -3.41	88 160 23.673 36.51	37 37 16.445 -4.90
32 32 15.100 -3.77	89 164 23.749 40.24	38 38 16.598 -4.76
33 33 15.283 -4.09	90 168 23.821 44.26	39 39 16.746 -4.55
34 34 15.460 -4.37	91 172 23.888 48.58	40 40 16.891 -4.29
35 35 15.631 -4.60	92 176 23.952 53.21	41 41 17.032 -3.99
36 36 15.795 -4.78	93 180 24.013 58.17	42 42 17.169 -3.64
37 37 15.955 -4.91	94 184 24.070 63.48	43 43 17.303 -3.26
38 38 16.110 -4.97	95 188 24.124 69.13	44 44 17.434 -2.86
39 39 16.260 -4.98	96 192 24.176 69.13	45 45 17.563 -2.45
40 40 16.405 -4.92	97 196 24.225 69.13	46 46 17.688 -2.04
41 41 16.547 -4.81	98 200 24.271 69.13	47 47 17.811 -1.63
42 42 16.685 -4.65	99 204 24.316 69.13	48 48 17.932 -1.24
43 43 16.820 -4.43	100 208 24.358 69.13	49 50 18.166 -0.51
44 44 16.951 -4.17	101 212 24.398 69.13	50 52 18.392 0.12
45 45 17.079 -3.87	102 216 24.436 69.13	51 54 18.611 0.64
46 46 17.204 -3.54	103 220 24.473 69.13	52 56 18.823 1.06
47 47 17.327 -3.19	104 224 24.508 69.13	53 58 19.028 1.39
48 48 17.447 -2.82	105 228 24.541 69.13	54 60 19.226 1.66
49 50 17.680 -2.06	106 232 24.573 69.13	55 62 19.419 1.88
50 52 17.904 -1.33	1th1	56 64 19.606 2.08
51 54 18.121 -0.64	103	57 66 19.788 2.27
52 56 18.331 -0.04	1 1 0.925 24.17	58 68 19.964 2.46
53 58 18.534 0.47	2 2 1.842 13.87	59 70 20.135 2.65
54 60 18.730 0.89	3 3 2.742 10.01	60 72 20.300 2.86
55 62 18.922 1.23	4 4 3.618 7.94	61 74 20.461 3.09
56 64 19.108 1.51	5 5 4.463 6.62	62 76 20.616 3.33
57 66 19.288 1.74	6 6 5.272 5.70	63 78 20.766 3.60



64 80 20.912 3.89	17 17 8.901 3.15	74 104 20.079 2.59
65 82 21.052 4.20	18 18 9.275 2.93	75 108 20.300 2.86
66 84 21.188 4.54	19 19 9.632 2.73	76 112 20.513 3.17
67 86 21.318 4.91	20 20 9.974 2.53	77 116 20.717 3.51
68 88 21.445 5.31	21 21 10.301 2.32	78 120 20.912 3.89
69 90 21.567 5.73	22 22 10.614 2.12	79 124 21.098 4.31
70 92 21.684 6.18	23 23 10.913 1.92	80 128 21.275 4.79
71 94 21.797 6.67	24 24 11.199 1.71	81 132 21.445 5.31
72 96 21.906 7.19	25 25 11.474 1.49	82 136 21.606 5.88
73 100 22.113 8.33	26 26 11.736 1.27	83 140 21.760 6.50
74 104 22.304 9.63	27 27 11.988 1.04	84 144 21.906 7.19
75 108 22.482 11.08	28 28 12.230 0.80	85 148 22.046 7.93
76 112 22.646 12.71	29 29 12.461 0.55	86 152 22.178 8.75
77 116 22.799 14.53	30 30 12.684 0.29	87 156 22.304 9.63
78 120 22.941 16.54	31 31 12.898 0.02	88 160 22.424 10.58
79 124 23.072 18.77	32 32 13.104 -0.25	89 164 22.538 11.60
80 128 23.195 21.23	33 33 13.302 -0.54	90 168 22.646 12.71
81 132 23.309 23.94	34 34 13.493 -0.83	91 172 22.749 13.90
82 136 23.415 26.90	35 35 13.678 -1.12	92 176 22.847 15.18
83 140 23.515 30.14	36 36 13.855 -1.43	93 180 22.941 16.54
84 144 23.607 33.67	37 37 14.027 -1.73	94 184 23.030 18.01
85 148 23.694 37.51	38 38 14.193 -2.04	95 188 23.114 19.57
86 152 23.775 41.67	39 39 14.354 -2.34	96 192 23.195 21.23
87 156 23.852 46.17	40 40 14.509 -2.64	97 196 23.272 23.01
88 160 23.923 51.04	41 41 14.660 -2.93	98 200 23.345 24.90
89 164 23.991 56.29	42 42 14.807 -3.22	99 204 23.415 26.90
90 168 24.054 61.94	43 43 14.949 -3.49	100 208 23.482 29.03
91 172 24.114 68.00	44 44 15.087 -3.74	101 212 23.546 31.28
92 176 24.171 68.00	45 45 15.221 -3.98	102 216 23.607 33.67
93 180 24.224 68.00	46 46 15.351 -4.20	103 220 23.666 36.19
94 184 24.275 68.00	47 47 15.478 -4.40	104 224 23.722 38.86
95 188 24.322 68.00	48 48 15.602 -4.57	105 228 23.775 41.67
96 192 24.368 68.00	49 50 15.841 -4.82	106 232 23.827 44.63
97 196 24.411 68.00	50 52 16.069 -4.96	107 236 23.876 47.76
98 200 24.452 68.00	51 54 16.287 -4.97	108 240 23.923 51.04
99 204 24.491 68.00	52 56 16.496 -4.86	2th0
100 208 24.528 68.00	53 58 16.697 -4.63	131
101 212 24.564 68.00	54 60 16.891 -4.29	1 1 0.425 45.05
102 216 24.597 68.00	55 62 17.078 -3.87	2 2 0.850 25.87
1th2	56 64 17.259 -3.39	3 3 1.273 18.70
109	57 66 17.434 -2.86	4 4 1.694 14.85
1 1 0.617 33.44	58 68 17.605 -2.31	5 5 2.112 12.41
2 2 1.232 19.20	59 70 17.770 -1.77	6 6 2.525 10.72
3 3 1.842 13.87	60 72 17.932 -1.24	7 7 2.934 9.47
4 4 2.445 11.01	61 74 18.089 -0.74	8 8 3.337 8.50
5 5 3.037 9.20	62 76 18.242 -0.29	9 9 3.733 7.73
6 6 3.618 7.94	63 78 18.392 0.12	10 10 4.124 7.10
7 7 4.185 7.00	64 80 18.539 0.48	11 11 4.507 6.56
8 8 4.736 6.28	65 82 18.682 0.79	12 12 4.882 6.11
9 9 5.272 5.70	66 84 18.823 1.06	13 13 5.249 5.72
10 10 5.789 5.21	67 86 18.960 1.29	14 14 5.608 5.37
11 11 6.289 4.80	68 88 19.095 1.49	15 15 5.959 5.07
12 12 6.770 4.45	69 90 19.226 1.66	16 16 6.301 4.79
13 13 7.233 4.14	70 92 19.356 1.81	17 17 6.634 4.55
14 14 7.677 3.86	71 94 19.482 1.95	18 18 6.959 4.32
15 15 8.103 3.61	72 96 19.606 2.08	19 19 7.274 4.11
16 16 8.511 3.37	73 100 19.847 2.33	20 20 7.581 3.92

21 21 7.879 3.74	78 120 18.730 0.89	3 3 1.385 17.47
22 22 8.169 3.57	79 124 18.922 1.23	4 4 1.842 13.87
23 23 8.450 3.40	80 128 19.108 1.51	5 5 2.295 11.60
24 24 8.723 3.25	81 132 19.288 1.74	6 6 2.742 10.01
25 25 8.987 3.10	82 136 19.464 1.93	7 7 3.184 8.84
26 26 9.244 2.95	83 140 19.635 2.11	8 8 3.618 7.94
27 27 9.493 2.81	84 144 19.801 2.28	9 9 4.045 7.22
28 28 9.734 2.67	85 148 19.963 2.45	10 10 4.463 6.62
29 29 9.968 2.53	86 152 20.120 2.63	11 11 4.872 6.12
30 30 10.195 2.39	87 156 20.273 2.82	12 12 5.272 5.70
31 31 10.416 2.25	88 160 20.421 3.03	13 13 5.661 5.33
32 32 10.629 2.11	89 164 20.565 3.25	14 14 6.041 5.00
33 33 10.836 1.97	90 168 20.705 3.49	15 15 6.411 4.71
34 34 11.037 1.83	91 172 20.840 3.74	16 16 6.770 4.45
35 35 11.232 1.68	92 176 20.971 4.02	17 17 7.119 4.21
36 36 11.421 1.53	93 180 21.099 4.32	18 18 7.457 4.00
37 37 11.605 1.38	94 184 21.222 4.64	19 19 7.785 3.79
38 38 11.783 1.23	95 188 21.341 4.98	20 20 8.103 3.61
39 39 11.957 1.07	96 192 21.457 5.35	21 21 8.410 3.43
40 40 12.125 0.90	97 200 21.676 6.15	22 22 8.708 3.26
41 41 12.289 0.74	98 208 21.882 7.07	23 23 8.996 3.09
42 42 12.448 0.56	99 216 22.074 8.10	24 24 9.275 2.93
43 43 12.603 0.39	100 224 22.253 9.25	25 25 9.544 2.78
44 44 12.753 0.21	101 232 22.420 10.54	26 26 9.805 2.63
45 45 12.900 0.02	102 240 22.575 11.97	27 27 10.057 2.47
46 46 13.042 -0.17	103 248 22.721 13.56	28 28 10.301 2.32
47 47 13.181 -0.36	104 256 22.857 15.30	29 29 10.537 2.17
48 48 13.317 -0.56	105 264 22.984 17.23	30 30 10.765 2.02
49 50 13.577 -0.96	106 272 23.102 19.33	31 31 10.986 1.86
50 52 13.825 -1.37	107 280 23.213 21.64	32 32 11.199 1.71
51 54 14.062 -1.79	108 288 23.317 24.15	33 33 11.406 1.55
52 56 14.288 -2.21	109 296 23.414 26.88	34 34 11.606 1.38
53 58 14.504 -2.63	110 304 23.506 29.84	35 35 11.800 1.21
54 60 14.711 -3.03	111 312 23.592 33.04	36 36 11.988 1.04
55 62 14.909 -3.41	112 320 23.673 36.51	37 37 12.170 0.86
56 64 15.100 -3.77	113 328 23.749 40.24	38 38 12.347 0.67
57 66 15.283 -4.09	114 336 23.821 44.26	39 39 12.518 0.49
58 68 15.460 -4.37	115 344 23.888 48.58	40 40 12.684 0.29
59 70 15.631 -4.60	116 352 23.952 53.21	41 41 12.845 0.09
60 72 15.795 -4.78	117 360 24.013 58.17	42 42 13.002 -0.11
61 74 15.955 -4.91	118 368 24.070 63.48	43 43 13.154 -0.32
62 76 16.110 -4.97	119 376 24.124 69.13	44 44 13.302 -0.54
63 78 16.260 -4.98	120 384 24.176 69.13	45 45 13.446 -0.75
64 80 16.405 -4.92	121 392 24.225 69.13	46 46 13.586 -0.97
65 82 16.547 -4.81	122 400 24.271 69.13	47 47 13.723 -1.20
66 84 16.685 -4.65	123 408 24.316 69.13	48 48 13.855 -1.43
67 86 16.820 -4.43	124 416 24.358 69.13	49 50 14.111 -1.88
68 88 16.951 -4.17	125 424 24.398 69.13	50 52 14.354 -2.34
69 90 17.079 -3.87	126 432 24.436 69.13	51 54 14.585 -2.79
70 92 17.204 -3.54	127 440 24.473 69.13	52 56 14.807 -3.22
71 94 17.327 -3.19	128 448 24.508 69.13	53 58 15.018 -3.62
72 96 17.447 -2.82	129 456 24.541 69.13	54 60 15.221 -3.98
73 100 17.680 -2.06	130 464 24.573 69.13	55 62 15.415 -4.30
74 104 17.904 -1.33	2th1	56 64 15.602 -4.57
75 108 18.121 -0.64	127	57 66 15.783 -4.77
76 112 18.331 -0.04	1 1 0.463 42.10	58 68 15.956 -4.91
77 116 18.534 0.47	2 2 0.925 24.17	59 70 16.124 -4.98

60 72 16.287 -4.97	117 360 24.224 68.00	46 46 10.913 1.92
61 74 16.445 -4.90	118 368 24.275 68.00	47 47 11.058 1.81
62 76 16.598 -4.76	119 376 24.322 68.00	48 48 11.199 1.71
63 78 16.746 -4.55	120 384 24.368 68.00	49 50 11.474 1.49
64 80 16.891 -4.29	121 392 24.411 68.00	50 52 11.736 1.27
65 82 17.032 -3.99	122 400 24.452 68.00	51 54 11.988 1.04
66 84 17.169 -3.64	123 408 24.491 68.00	52 56 12.230 0.80
67 86 17.303 -3.26	124 416 24.528 68.00	53 58 12.461 0.55
68 88 17.434 -2.86	125 424 24.564 68.00	54 60 12.684 0.29
69 90 17.563 -2.45	126 432 24.597 68.00	55 62 12.898 0.02
70 92 17.688 -2.04	2th2	56 64 13.104 -0.25
71 94 17.811 -1.63	133	57 66 13.302 -0.54
72 96 17.932 -1.24	1 1 0.309 58.23	58 68 13.493 -0.83
73 100 18.166 -0.51	2 2 0.617 33.44	59 70 13.678 -1.12
74 104 18.392 0.12	3 3 0.925 24.17	60 72 13.855 -1.43
75 108 18.611 0.64	4 4 1.232 19.20	61 74 14.027 -1.73
76 112 18.823 1.06	5 5 1.538 16.05	62 76 14.193 -2.04
77 116 19.028 1.39	6 6 1.842 13.87	63 78 14.354 -2.34
78 120 19.226 1.66	7 7 2.145 12.26	64 80 14.509 -2.64
79 124 19.419 1.88	8 8 2.445 11.01	65 82 14.660 -2.93
80 128 19.606 2.08	9 9 2.742 10.01	66 84 14.807 -3.22
81 132 19.788 2.27	10 10 3.037 9.20	67 86 14.949 -3.49
82 136 19.964 2.46	11 11 3.329 8.52	68 88 15.087 -3.74
83 140 20.135 2.65	12 12 3.618 7.94	69 90 15.221 -3.98
84 144 20.300 2.86	13 13 3.903 7.44	70 92 15.351 -4.20
85 148 20.461 3.09	14 14 4.185 7.00	71 94 15.478 -4.40
86 152 20.616 3.33	15 15 4.463 6.62	72 96 15.602 -4.57
87 156 20.766 3.60	16 16 4.736 6.28	73 100 15.841 -4.82
88 160 20.912 3.89	17 17 5.006 5.97	74 104 16.069 -4.96
89 164 21.052 4.20	18 18 5.272 5.70	75 108 16.287 -4.97
90 168 21.188 4.54	19 19 5.533 5.44	76 112 16.496 -4.86
91 172 21.318 4.91	20 20 5.789 5.21	77 116 16.697 -4.63
92 176 21.445 5.31	21 21 6.041 5.00	78 120 16.891 -4.29
93 180 21.567 5.73	22 22 6.289 4.80	79 124 17.078 -3.87
94 184 21.684 6.18	23 23 6.532 4.62	80 128 17.259 -3.39
95 188 21.797 6.67	24 24 6.770 4.45	81 132 17.434 -2.86
96 192 21.906 7.19	25 25 7.004 4.29	82 136 17.605 -2.31
97 200 22.113 8.33	26 26 7.233 4.14	83 140 17.770 -1.77
98 208 22.304 9.63	27 27 7.457 4.00	84 144 17.932 -1.24
99 216 22.482 11.08	28 28 7.677 3.86	85 148 18.089 -0.74
100 224 22.646 12.71	29 29 7.892 3.73	86 152 18.242 -0.29
101 232 22.799 14.53	30 30 8.103 3.61	87 156 18.392 0.12
102 240 22.941 16.54	31 31 8.309 3.49	88 160 18.539 0.48
103 248 23.072 18.77	32 32 8.511 3.37	89 164 18.682 0.79
104 256 23.195 21.23	33 33 8.708 3.26	90 168 18.823 1.06
105 264 23.309 23.94	34 34 8.901 3.15	91 172 18.960 1.29
106 272 23.415 26.90	35 35 9.090 3.04	92 176 19.095 1.49
107 280 23.515 30.14	36 36 9.275 2.93	93 180 19.226 1.66
108 288 23.607 33.67	37 37 9.456 2.83	94 184 19.356 1.81
109 296 23.694 37.51	38 38 9.632 2.73	95 188 19.482 1.95
110 304 23.775 41.67	39 39 9.805 2.63	96 192 19.606 2.08
111 312 23.852 46.17	40 40 9.974 2.53	97 200 19.847 2.33
112 320 23.923 51.04	41 41 10.139 2.42	98 208 20.079 2.59
113 328 23.991 56.29	42 42 10.301 2.32	99 216 20.300 2.86
114 336 24.054 61.94	43 43 10.459 2.22	100 224 20.513 3.17
115 344 24.114 68.00	44 44 10.614 2.12	101 232 20.717 3.51
116 352 24.171 68.00	45 45 10.765 2.02	102 240 20.912 3.89

103 248 21.098 4.31	39603.86	7273.40
104 256 21.275 4.79	38347.55	7273.40
105 264 21.445 5.31	37131.08	6834.99
106 272 21.606 5.88	36036.08	6834.99
107 280 21.760 6.50	35054.00	6452.64
108 288 21.906 7.19	34098.68	6452.64
109 296 22.046 7.93	33169.40	6105.73
110 304 22.178 8.75	32339.82	6105.73
111 312 22.304 9.63	31530.99	5804.14
112 320 22.424 10.58	30742.39	5804.14
113 328 22.538 11.60	29973.51	5542.91
114 336 22.646 12.71	29291.23	5542.91
115 344 22.749 13.90	28624.48	5330.13
116 352 22.847 15.18	27972.91	5330.13
117 352 22.941 16.54	27336.16	5031.96
118 368 23.030 18.01	26652.48	5031.96
119 376 23.114 19.57	26045.79	5031.96
120 384 23.195 21.23	25452.92	5031.96
121 382 23.272 23.01	24873.54	4872.34
122 400 23.345 24.90	24307.35	4872.34
123 408 23.415 26.90	23754.04	4872.34
124 416 23.482 29.03	23159.95	4872.34
125 424 23.546 31.28	22632.76	4861.13
126 432 23.607 33.67	21514.81	4861.13
127 440 23.666 36.19	21514.81	4861.13
128 448 23.722 38.86	20452.08	4861.13
129 456 23.775 41.67	20452.08	4985.83
130 464 23.827 44.63	19397.13	4985.83
131 472 23.876 47.76	19397.13	4985.83
132 480 23.923 51.04	18354.29	4985.83
absthr_0	18354.29	5257.00
table 0	17327.57	5257.00
10156347392.00	17327.57	5257.00
33708348.00	16320.66	5257.00
3987838.75	16320.66	5685.09
1269806.88	15336.90	5685.09
614805.19	15336.90	5685.09
372166.75	14412.44	5685.09
256884.33	14412.44	6262.35
192636.06	13481.48	6262.35
153016.27	13481.48	6262.35
126980.66	12610.65	6262.35
108576.95	12610.65	6994.19
95003.05	11796.08	6994.19
84671.56	11796.08	6994.19
76513.41	10983.42	6994.19
70103.16	10983.42	7902.01
64824.27	10250.32	7902.01
60358.38	10250.32	7902.01
56720.18	9544.16	7902.01
53424.14	9544.16	8968.87
50668.45	8907.13	8968.87
48276.70	8907.13	8968.87
46103.89	8312.61	8968.87
44232.10	8312.61	10156.35
42534.13	7775.67	10156.35
40995.63	7775.67	10156.35

10156.35	26105.83	45997.86
11474.60	26105.83	45997.86
11474.60	26105.83	45997.86
11474.60	26105.83	51848.66
11474.60	27716.45	51848.66
12874.71	27716.45	51848.66
12874.71	27716.45	51848.66
12874.71	27716.45	51848.66
12874.71	27716.45	51848.66
14280.31	27716.45	51848.66
14280.31	27716.45	51848.66
14280.31	27716.45	59120.43
14280.31	29494.26	59120.43
15694.14	29494.26	59120.43
15694.14	29494.26	59120.43
15694.14	29494.26	59120.43
15694.14	29494.26	59120.43
17050.52	29494.26	59120.43
17050.52	29494.26	59120.43
17050.52	29494.26	68192.65
17050.52	31676.53	68192.65
18312.08	31676.53	68192.65
18312.08	31676.53	68192.65
18312.08	31676.53	68192.65
18312.08	31676.53	68192.65
19486.67	31676.53	68192.65
19486.67	31676.53	68192.65
19486.67	31676.53	79935.11
19486.67	34256.07	79935.11
20546.49	34256.07	79935.11
20546.49	34256.07	79935.11
20546.49	34256.07	79935.11
20546.49	34256.07	79935.11
21514.81	34256.07	79935.11
21514.81	34256.07	79935.11
21514.81	34256.07	94784.55
21514.81	37388.46	94784.55
22373.69	37388.46	94784.55
22373.69	37388.46	94784.55
22373.69	37388.46	94784.55
22373.69	37388.46	94784.55
23159.95	37388.46	94784.55
23159.95	37388.46	94784.55
23159.95	37388.46	114482.10
23159.95	41184.86	114482.10
23918.70	41184.86	114482.10
23918.70	41184.86	114482.10
23918.70	41184.86	114482.10
23918.70	41184.86	114482.10
24645.50	41184.86	114482.10
24645.50	41184.86	114482.10
24645.50	41184.86	140196.64
24645.50	45997.86	140196.64
26105.83	45997.86	140196.64
26105.83	45997.86	140196.64
26105.83	45997.86	140196.64
26105.83	45997.86	140196.64

140196.64	965467.38	35541636.00
140196.64	1382730.50	35541636.00
174476.75	1382730.50	35541636.00
174476.75	1382730.50	35541636.00
174476.75	1382730.50	35541636.00
174476.75	1382730.50	35541636.00
174476.75	1382730.50	35541636.00
174476.75	1382730.50	35541636.00
174476.75	1382730.50	63494696.00
174476.75	2026457.63	63494696.00
220667.11	2026457.63	63494696.00
220667.11	2026457.63	63494696.00
220667.11	2026457.63	63494696.00
220667.11	2026457.63	63494696.00
220667.11	2026457.63	63494696.00
220667.11	2026457.63	117418800.00
220667.11	3053076.00	117418800.00
284929.63	3053076.00	117418800.00
284929.63	3053076.00	117418800.00
284929.63	3053076.00	117418800.00
284929.63	3053076.00	117418800.00
284929.63	3053076.00	117418800.00
284929.63	3053076.00	117418800.00
284929.63	3053076.00	224252560.00
284929.63	4717778.50	224252560.00
374746.47	4717778.50	224252560.00
374746.47	4717778.50	224252560.00
374746.47	4717778.50	224252560.00
374746.47	4717778.50	224252560.00
374746.47	4717778.50	224252560.00
374746.47	4717778.50	224252560.00
374746.47	4717778.50	443340800.00
374746.47	7477175.00	443340800.00
503196.50	7477175.00	443340800.00
503196.50	7477175.00	443340800.00
503196.50	7477175.00	443340800.00
503196.50	7477175.00	443340800.00
503196.50	7477175.00	443340800.00
503196.50	7477175.00	911459584.00
503196.50	12210618.00	911459584.00
688236.44	12210618.00	911459584.00
688236.44	12210618.00	911459584.00
688236.44	12210618.00	911459584.00
688236.44	12210618.00	911459584.00
688236.44	12210618.00	911459584.00
688236.44	12210618.00	911459584.00
688236.44	12210618.00	1935251712.00
688236.44	20499234.00	1935251712.00
965467.38	20499234.00	1935251712.00
965467.38	20499234.00	1935251712.00
965467.38	20499234.00	1935251712.00
965467.38	20499234.00	1935251712.00
965467.38	20499234.00	1935251712.00
965467.38	20499234.00	1935251712.00
965467.38	20499234.00	60776765194240.00

60776765194240.00	32265.44	5043.56
60776765194240.00	31170.05	5043.56
60776765194240.00	30111.86	5232.84
60776765194240.00	29156.65	5232.84
60776765194240.00	28231.74	5504.75
60776765194240.00	27336.16	5504.75
60776765194240.00	26469.00	5844.37
60776765194240.00	25629.35	5844.37
60776765194240.00	24816.33	6262.35
60776765194240.00	24029.10	6262.35
60776765194240.00	23266.85	6756.75
60776765194240.00	22476.96	6756.75
60776765194240.00	21713.89	7323.82
60776765194240.00	20976.72	7323.82
60776765194240.00	20264.58	7975.13
60776765194240.00	19531.59	7975.13
60776765194240.00	18781.82	9500.31
60776765194240.00	18102.46	9500.31
60776765194240.00	17367.51	9500.31
60776765194240.00	16700.81	9500.31
60776765194240.00	16022.77	11239.26
60776765194240.00	15336.90	11239.26
60776765194240.00	14680.39	11239.26
60776765194240.00	14051.98	11239.26
60776765194240.00	13419.54	13174.60
60776765194240.00	12238.76	13174.60
60776765194240.00	12238.76	13174.60
60776765194240.00	11136.21	13174.60
60776765194240.00	11136.21	15126.47
60776765194240.00	10109.68	15126.47
60776765194240.00	10109.68	15126.47
60776765194240.00	9177.78	15126.47
absthr_1	9177.78	17011.30
table 1	8331.78	17011.30
488357088.00	8331.78	17011.30
5898447.00	7598.67	17011.30
1131716.50	7598.67	18738.62
466377.53	6962.06	18738.62
265911.78	6962.06	18738.62
180192.88	6408.22	18738.62
135125.59	6408.22	20264.58
108078.07	5953.02	20264.58
90518.55	5953.02	20264.58
78295.64	5581.33	20264.58
69141.30	5581.33	21614.12
62335.81	5293.44	21614.12
56981.98	5293.44	21614.12
52569.95	5078.52	21614.12
49061.14	5078.52	22789.65
45997.86	4928.76	22789.65
43524.88	4928.76	22789.65
41279.80	4861.13	22789.65
39331.23	4861.13	23808.80
37647.63	4849.95	23808.80
36119.16	4849.95	23808.80
34732.62	4917.42	23808.80
33399.32	4917.42	24816.33

24816.33	62912.59	517294.19
24816.33	62912.59	517294.19
24816.33	62912.59	517294.19
25807.00	62912.59	517294.19
25807.00	62912.59	517294.19
25807.00	62912.59	806747.31
25807.00	77756.66	806747.31
26837.22	77756.66	806747.31
26837.22	77756.66	806747.31
26837.22	77756.66	806747.31
26837.22	77756.66	806747.31
27972.91	77756.66	806747.31
27972.91	77756.66	806747.31
27972.91	77756.66	1308391.13
27972.91	98568.38	1308391.13
29223.86	98568.38	1308391.13
29223.86	98568.38	1308391.13
29223.86	98568.38	1308391.13
29223.86	98568.38	1308391.13
30671.68	98568.38	1308391.13
30671.68	98568.38	1308391.13
30671.68	98568.38	2227088.75
30671.68	128451.02	2227088.75
32265.44	128451.02	2227088.75
32265.44	128451.02	2227088.75
32265.44	128451.02	2227088.75
32265.44	128451.02	2227088.75
34098.68	128451.02	2227088.75
34098.68	128451.02	2227088.75
34098.68	128451.02	3969516.00
34098.68	172877.14	3969516.00
36119.16	172877.14	3969516.00
36119.16	172877.14	3969516.00
36119.16	172877.14	3969516.00
36119.16	172877.14	3969516.00
38524.55	172877.14	3969516.00
38524.55	172877.14	3969516.00
38524.55	172877.14	7442819.50
38524.55	240291.06	7442819.50
41279.80	240291.06	7442819.50
41279.80	240291.06	7442819.50
41279.80	240291.06	7442819.50
41279.80	240291.06	7442819.50
44436.27	240291.06	7442819.50
44436.27	240291.06	7442819.50
44436.27	240291.06	14714232.00
44436.27	346527.44	14714232.00
48054.89	346527.44	14714232.00
48054.89	346527.44	14714232.00
48054.89	346527.44	14714232.00
48054.89	346527.44	14714232.00
52328.41	346527.44	14714232.00
52328.41	346527.44	14714232.00
52328.41	346527.44	30742394.00
52328.41	517294.19	30742394.00
62912.59	517294.19	30742394.00
62912.59	517294.19	30742394.00



[illegible]

60776765194240.00	32339.82	5685.09
60776765194240.00	31098.37	5685.09
60776765194240.00	29973.51	6091.69
60776765194240.00	28955.93	6091.69
60776765194240.00	27972.91	6602.94
60776765194240.00	26961.10	6602.94
60776765194240.00	26045.79	7206.72
60776765194240.00	25161.56	7206.72
60776765194240.00	24307.35	7902.01
60776765194240.00	23428.13	7902.01
60776765194240.00	22632.76	8684.36
60776765194240.00	21814.11	8684.36
60776765194240.00	20976.72	9544.16
60776765194240.00	20171.47	9544.16
60776765194240.00	19397.13	10489.08
60776765194240.00	18609.62	10489.08
60776765194240.00	17813.02	11474.60
60776765194240.00	17089.82	11474.60
60776765194240.00	16320.66	13574.93
60776765194240.00	15586.10	13574.93
60776765194240.00	14884.61	13574.93
60776765194240.00	14182.00	13574.93
60776765194240.00	13481.48	15694.14
60776765194240.00	12845.10	15694.14
60776765194240.00	12210.62	15694.14
60776765194240.00	11580.77	15694.14
60776765194240.00	10983.42	17690.40
60776765194240.00	9902.33	17690.40
60776765194240.00	9902.33	17690.40
60776765194240.00	8907.13	17690.40
60776765194240.00	8907.13	19486.67
60776765194240.00	8030.41	19486.67
60776765194240.00	8030.41	19486.67
60776765194240.00	7273.40	19486.67
absthr_2	7273.40	21025.08
table 2	6633.42	21025.08
247592464.00	6633.42	21025.08
3987838.75	6105.73	21025.08
852584.69	6105.73	22373.69
372166.75	5672.02	22373.69
220667.11	5672.02	22373.69
153016.27	5330.13	22373.69
116879.30	5330.13	23536.27
95003.05	5090.23	23536.27
80489.19	5090.23	23536.27
70103.16	4928.76	23536.27
62479.50	4928.76	24645.50
56720.18	4849.95	24645.50
52087.99	4849.95	24645.50
48276.70	4861.13	24645.50
45158.30	4861.13	25747.65
42534.13	4940.12	25747.65
40247.38	4940.12	25747.65
38347.55	5101.97	25747.65
36537.39	5101.97	26899.09
35054.00	5354.73	26899.09
33630.83	5354.73	26899.09

26899.09	103929.20	2026457.63
28102.02	103929.20	2026457.63
28102.02	103929.20	2026457.63
28102.02	103929.20	2026457.63
28102.02	103929.20	2026457.63
29494.26	103929.20	2026457.63
29494.26	103929.20	2026457.63
29494.26	103929.20	3782140.25
29494.26	140196.64	3782140.25
31098.37	140196.64	3782140.25
31098.37	140196.64	3782140.25
31098.37	140196.64	3782140.25
31098.37	140196.64	3782140.25
32865.30	140196.64	3782140.25
32865.30	140196.64	3782140.25
32865.30	140196.64	7477175.00
32865.30	195766.14	7477175.00
34973.38	195766.14	7477175.00
34973.38	195766.14	7477175.00
34973.38	195766.14	7477175.00
34973.38	195766.14	7477175.00
34973.38	195766.14	7477175.00
37388.46	195766.14	7477175.00
37388.46	195766.14	7477175.00
37388.46	195766.14	15766581.00
37388.46	284929.63	15766581.00
40154.81	284929.63	15766581.00
40154.81	284929.63	15766581.00
40154.81	284929.63	15766581.00
40154.81	284929.63	15766581.00
43424.77	284929.63	15766581.00
43424.77	284929.63	15766581.00
43424.77	284929.63	35541636.00
43424.77	433249.00	35541636.00
47286.54	433249.00	35541636.00
47286.54	433249.00	35541636.00
47286.54	433249.00	35541636.00
47286.54	433249.00	35541636.00
51848.66	433249.00	35541636.00
51848.66	433249.00	35541636.00
51848.66	433249.00	86047336.00
51848.66	688236.44	86047336.00
57113.34	688236.44	86047336.00
57113.34	688236.44	86047336.00
57113.34	688236.44	86047336.00
57113.34	688236.44	86047336.00
63348.68	688236.44	86047336.00
63348.68	688236.44	86047336.00
63348.68	688236.44	224252560.00
63348.68	1150105.38	224252560.00
70914.91	1150105.38	224252560.00
70914.91	1150105.38	224252560.00
70914.91	1150105.38	224252560.00
70914.91	1150105.38	224252560.00
79935.11	1150105.38	224252560.00
79935.11	1150105.38	224252560.00
79935.11	1150105.38	632029568.00
79935.11	2026457.63	632029568.00

[illegible]

60776765194240.00	1 3 15 4 3 4	4 12 2047 11 3 11
60776765194240.00	1 4 31 5 3 5	4 13 4095 12 3 12
60776765194240.00	1 5 63 6 3 6	4 14 8191 13 3 13
60776765194240.00	1 6 127 7 3 7	4 15 65535 16 3 16
60776765194240.00	1 7 255 8 3 8	5 0 0 4 0 0
60776765194240.00	1 8 511 9 3 9	5 1 3 5 1 0
60776765194240.00	1 9 1023 10 3 10	5 2 5 7 1 1
60776765194240.00	1 10 2047 11 3 11	5 3 7 3 3 2
60776765194240.00	1 11 4095 12 3 12	5 4 9 10 1 3
60776765194240.00	1 12 8191 13 3 13	5 5 15 4 3 4
60776765194240.00	1 13 16383 14 3 14	5 6 31 5 3 5
60776765194240.00	1 14 32767 15 3 15	5 7 63 6 3 6
60776765194240.00	1 15 65535 16 3 16	5 8 127 7 3 7
60776765194240.00	2 0 0 4 0 0	5 9 255 8 3 8
60776765194240.00	2 1 3 5 1 0	5 10 511 9 3 9
60776765194240.00	2 2 7 3 3 2	5 11 1023 10 3 10
60776765194240.00	2 3 15 4 3 4	5 12 2047 11 3 11
60776765194240.00	2 4 31 5 3 5	5 13 4095 12 3 12
60776765194240.00	2 5 63 6 3 6	5 14 8191 13 3 13
60776765194240.00	2 6 127 7 3 7	5 15 65535 16 3 16
60776765194240.00	2 7 255 8 3 8	6 0 0 4 0 0
60776765194240.00	2 8 511 9 3 9	6 1 3 5 1 0
60776765194240.00	2 9 1023 10 3 10	6 2 5 7 1 1
60776765194240.00	2 10 2047 11 3 11	6 3 7 3 3 2
60776765194240.00	2 11 4095 12 3 12	6 4 9 10 1 3
60776765194240.00	2 12 8191 13 3 13	6 5 15 4 3 4
60776765194240.00	2 13 16383 14 3 14	6 6 31 5 3 5
60776765194240.00	2 14 32767 15 3 15	6 7 63 6 3 6
60776765194240.00	2 15 65535 16 3 16	6 8 127 7 3 7
60776765194240.00	3 0 0 4 0 0	6 9 255 8 3 8
60776765194240.00	3 1 3 5 1 0	6 10 511 9 3 9
60776765194240.00	3 2 5 7 1 1	6 11 1023 10 3 10
60776765194240.00	3 3 7 3 3 2	6 12 2047 11 3 11
60776765194240.00	3 4 9 10 1 3	6 13 4095 12 3 12
60776765194240.00	3 5 15 4 3 4	6 14 8191 13 3 13
60776765194240.00	3 6 31 5 3 5	6 15 65535 16 3 16
alloc_0	3 7 63 6 3 6	7 0 0 4 0 0
27	3 8 127 7 3 7	7 1 3 5 1 0
0 0 0 4 0 0	3 9 255 8 3 8	7 2 5 7 1 1
0 1 3 5 1 0	3 10 511 9 3 9	7 3 7 3 3 2
0 2 7 3 3 2	3 11 1023 10 3 10	7 4 9 10 1 3
0 3 15 4 3 4	3 12 2047 11 3 11	7 5 15 4 3 4
0 4 31 5 3 5	3 13 4095 12 3 12	7 6 31 5 3 5
0 5 63 6 3 6	3 14 8191 13 3 13	7 7 63 6 3 6
0 6 127 7 3 7	3 15 65535 16 3 16	7 8 127 7 3 7
0 7 255 8 3 8	4 0 0 4 0 0	7 9 255 8 3 8
0 8 511 9 3 9	4 1 3 5 1 0	7 10 511 9 3 9
0 9 1023 10 3 10	4 2 5 7 1 1	7 11 1023 10 3 10
0 10 2047 11 3 11	4 3 7 3 3 2	7 12 2047 11 3 11
0 11 4095 12 3 12	4 4 9 10 1 3	7 13 4095 12 3 12
0 12 8191 13 3 13	4 5 15 4 3 4	7 14 8191 13 3 13
0 13 16383 14 3 14	4 6 31 5 3 5	7 15 65535 16 3 16
0 14 32767 15 3 15	4 7 63 6 3 6	8 0 0 4 0 0
0 15 65535 16 3 16	4 8 127 7 3 7	8 1 3 5 1 0
1 0 0 4 0 0	4 9 255 8 3 8	8 2 5 7 1 1
1 1 3 5 1 0	4 10 511 9 3 9	8 3 7 3 3 2
1 2 7 3 3 2	4 11 1023 10 3 10	8 4 9 10 1 3

8 5 15 4 3 4	12 6 31 5 3 5	19 7 65535 16 3 16
8 6 31 5 3 5	12 7 65535 16 3 16	20 0 0 3 0 0
8 7 63 6 3 6	13 0 0 3 0 0	20 1 3 5 1 0
8 8 127 7 3 7	13 1 3 5 1 0	20 2 5 7 1 1
8 9 255 8 3 8	13 2 5 7 1 1	20 3 7 3 3 2
8 10 511 9 3 9	13 3 7 3 3 2	20 4 9 10 1 3
8 11 1023 10 3 10	13 4 9 10 1 3	20 5 15 4 3 4
8 12 2047 11 3 11	13 5 15 4 3 4	20 6 31 5 3 5
8 13 4095 12 3 12	13 6 31 5 3 5	20 7 65535 16 3 16
8 14 8191 13 3 13	13 7 65535 16 3 16	21 0 0 3 0 0
8 15 65535 16 3 16	14 0 0 3 0 0	21 1 3 5 1 0
9 0 0 4 0 0	14 1 3 5 1 0	21 2 5 7 1 1
9 1 3 5 1 0	14 2 5 7 1 1	21 3 7 3 3 2
9 2 5 7 1 1	14 3 7 3 3 2	21 4 9 10 1 3
9 3 7 3 3 2	14 4 9 10 1 3	21 5 15 4 3 4
9 4 9 10 1 3	14 5 15 4 3 4	21 6 31 5 3 5
9 5 15 4 3 4	14 6 31 5 3 5	21 7 65535 16 3 16
9 6 31 5 3 5	14 7 65535 16 3 16	22 0 0 3 0 0
9 7 63 6 3 6	15 0 0 3 0 0	22 1 3 5 1 0
9 8 127 7 3 7	15 1 3 5 1 0	22 2 5 7 1 1
9 9 255 8 3 8	15 2 5 7 1 1	22 3 7 3 3 2
9 10 511 9 3 9	15 3 7 3 3 2	22 4 9 10 1 3
9 11 1023 10 3 10	15 4 9 10 1 3	22 5 15 4 3 4
9 12 2047 11 3 11	15 5 15 4 3 4	22 6 31 5 3 5
9 13 4095 12 3 12	15 6 31 5 3 5	22 7 65535 16 3 16
9 14 8191 13 3 13	15 7 65535 16 3 16	23 0 0 2 0 0
9 15 65535 16 3 16	16 0 0 3 0 0	23 1 3 5 1 0
10 0 0 4 0 0	16 1 3 5 1 0	23 2 5 7 1 1
10 1 3 5 1 0	16 2 5 7 1 1	23 3 65535 16 3 16
10 2 5 7 1 1	16 3 7 3 3 2	24 0 0 2 0 0
10 3 7 3 3 2	16 4 9 10 1 3	24 1 3 5 1 0
10 4 9 10 1 3	16 5 15 4 3 4	24 2 5 7 1 1
10 5 15 4 3 4	16 6 31 5 3 5	24 3 65535 16 3 16
10 6 31 5 3 5	16 7 65535 16 3 16	25 0 0 2 0 0
10 7 63 6 3 6	17 0 0 3 0 0	25 1 3 5 1 0
10 8 127 7 3 7	17 1 3 5 1 0	25 2 5 7 1 1
10 9 255 8 3 8	17 2 5 7 1 1	25 3 65535 16 3 16
10 10 511 9 3 9	17 3 7 3 3 2	26 0 0 2 0 0
10 11 1023 10 3 10	17 4 9 10 1 3	26 1 3 5 1 0
10 12 2047 11 3 11	17 5 15 4 3 4	26 2 5 7 1 1
10 13 4095 12 3 12	17 6 31 5 3 5	26 3 65535 16 3 16
10 14 8191 13 3 13	17 7 65535 16 3 16	alloc_1
10 15 65535 16 3 16	18 0 0 3 0 0	30
11 0 0 3 0 0	18 1 3 5 1 0	0 0 0 4 0 0
11 1 3 5 1 0	18 2 5 7 1 1	0 1 3 5 1 0
11 2 5 7 1 1	18 3 7 3 3 2	0 2 7 3 3 2
11 3 7 3 3 2	18 4 9 10 1 3	0 3 15 4 3 4
11 4 9 10 1 3	18 5 15 4 3 4	0 4 31 5 3 5
11 5 15 4 3 4	18 6 31 5 3 5	0 5 63 6 3 6
11 6 31 5 3 5	18 7 65535 16 3 16	0 6 127 7 3 7
11 7 65535 16 3 16	19 0 0 3 0 0	0 7 255 8 3 8
12 0 0 3 0 0	19 1 3 5 1 0	0 8 511 9 3 9
12 1 3 5 1 0	19 2 5 7 1 1	0 9 1023 10 3 10
12 2 5 7 1 1	19 3 7 3 3 2	0 10 2047 11 3 11
12 3 7 3 3 2	19 4 9 10 1 3	0 11 4095 12 3 12
12 4 9 10 1 3	19 5 15 4 3 4	0 12 8191 13 3 13
12 5 15 4 3 4	19 6 31 5 3 5	0 13 16383 14 3 14

0 14 32767 15 3 15	4 7 63 6 3 6	8 0 0 4 0 0
0 15 65535 16 3 16	4 8 127 7 3 7	8 1 3 5 1 0
1 0 0 4 0 0	4 9 255 8 3 8	8 2 5 7 1 1
1 1 3 5 1 0	4 10 511 9 3 9	8 3 7 3 3 2
1 2 7 3 3 2	4 11 1023 10 3 10	8 4 9 10 1 3
1 3 15 4 3 4	4 12 2047 11 3 11	8 5 15 4 3 4
1 4 31 5 3 5	4 13 4095 12 3 12	8 6 31 5 3 5
1 5 63 6 3 6	4 14 8191 13 3 13	8 7 63 6 3 6
1 6 127 7 3 7	4 15 65535 16 3 16	8 8 127 7 3 7
1 7 255 8 3 8	5 0 0 4 0 0	8 9 255 8 3 8
1 8 511 9 3 9	5 1 3 5 1 0	8 10 511 9 3 9
1 9 1023 10 3 10	5 2 5 7 1 1	8 11 1023 10 3 10
1 10 2047 11 3 11	5 3 7 3 3 2	8 12 2047 11 3 11
1 11 4095 12 3 12	5 4 9 10 1 3	8 13 4095 12 3 12
1 12 8191 13 3 13	5 5 15 4 3 4	8 14 8191 13 3 13
1 13 16383 14 3 14	5 6 31 5 3 5	8 15 65535 16 3 16
1 14 32767 15 3 15	5 7 63 6 3 6	9 0 0 4 0 0
1 15 65535 16 3 16	5 8 127 7 3 7	9 1 3 5 1 0
2 0 0 4 0 0	5 9 255 8 3 8	9 2 5 7 1 1
2 1 3 5 1 0	5 10 511 9 3 9	9 3 7 3 3 2
2 2 7 3 3 2	5 11 1023 10 3 10	9 4 9 10 1 3
2 3 15 4 3 4	5 12 2047 11 3 11	9 5 15 4 3 4
2 4 31 5 3 5	5 13 4095 12 3 12	9 6 31 5 3 5
2 5 63 6 3 6	5 14 8191 13 3 13	9 7 63 6 3 6
2 6 127 7 3 7	5 15 65535 16 3 16	9 8 127 7 3 7
2 7 255 8 3 8	6 0 0 4 0 0	9 9 255 8 3 8
2 8 511 9 3 9	6 1 3 5 1 0	9 10 511 9 3 9
2 9 1023 10 3 10	6 2 5 7 1 1	9 11 1023 10 3 10
2 10 2047 11 3 11	6 3 7 3 3 2	9 12 2047 11 3 11
2 11 4095 12 3 12	6 4 9 10 1 3	9 13 4095 12 3 12
2 12 8191 13 3 13	6 5 15 4 3 4	9 14 8191 13 3 13
2 13 16383 14 3 14	6 6 31 5 3 5	9 15 65535 16 3 16
2 14 32767 15 3 15	6 7 63 6 3 6	10 0 0 4 0 0
2 15 65535 16 3 16	6 8 127 7 3 7	10 1 3 5 1 0
3 0 0 4 0 0	6 9 255 8 3 8	10 2 5 7 1 1
3 1 3 5 1 0	6 10 511 9 3 9	10 3 7 3 3 2
3 2 5 7 1 1	6 11 1023 10 3 10	10 4 9 10 1 3
3 3 7 3 3 2	6 12 2047 11 3 11	10 5 15 4 3 4
3 4 9 10 1 3	6 13 4095 12 3 12	10 6 31 5 3 5
3 5 15 4 3 4	6 14 8191 13 3 13	10 7 63 6 3 6
3 6 31 5 3 5	6 15 65535 16 3 16	10 8 127 7 3 7
3 7 63 6 3 6	7 0 0 4 0 0	10 9 255 8 3 8
3 8 127 7 3 7	7 1 3 5 1 0	10 10 511 9 3 9
3 9 255 8 3 8	7 2 5 7 1 1	10 11 1023 10 3 10
3 10 511 9 3 9	7 3 7 3 3 2	10 12 2047 11 3 11
3 11 1023 10 3 10	7 4 9 10 1 3	10 13 4095 12 3 12
3 12 2047 11 3 11	7 5 15 4 3 4	10 14 8191 13 3 13
3 13 4095 12 3 12	7 6 31 5 3 5	10 15 65535 16 3 16
3 14 8191 13 3 13	7 7 63 6 3 6	11 0 0 3 0 0
3 15 65535 16 3 16	7 8 127 7 3 7	11 1 3 5 1 0
4 0 0 4 0 0	7 9 255 8 3 8	11 2 5 7 1 1
4 1 3 5 1 0	7 10 511 9 3 9	11 3 7 3 3 2
4 2 5 7 1 1	7 11 1023 10 3 10	11 4 9 10 1 3
4 3 7 3 3 2	7 12 2047 11 3 11	11 5 15 4 3 4
4 4 9 10 1 3	7 13 4095 12 3 12	11 6 31 5 3 5
4 5 15 4 3 4	7 14 8191 13 3 13	11 7 65535 16 3 16
4 6 31 5 3 5	7 15 65535 16 3 16	12 0 0 3 0 0

12 1 3 5 1 0	19 2 5 7 1 1	29 3 65535 16 3 16
12 2 5 7 1 1	19 3 7 3 3 2	alloc_2
12 3 7 3 3 2	19 4 9 10 1 3	8
12 4 9 10 1 3	19 5 15 4 3 4	0 0 0 4 0 0
12 5 15 4 3 4	19 6 31 5 3 5	0 1 3 5 1 0
12 6 31 5 3 5	19 7 65535 16 3 16	0 2 5 7 1 1
12 7 65535 16 3 16	20 0 0 3 0 0	0 3 9 10 1 3
13 0 0 3 0 0	20 1 3 5 1 0	0 4 15 4 3 4
13 1 3 5 1 0	20 2 5 7 1 1	0 5 31 5 3 5
13 2 5 7 1 1	20 3 7 3 3 2	0 6 63 6 3 6
13 3 7 3 3 2	20 4 9 10 1 3	0 7 127 7 3 7
13 4 9 10 1 3	20 5 15 4 3 4	0 8 255 8 3 8
13 5 15 4 3 4	20 6 31 5 3 5	0 9 511 9 3 9
13 6 31 5 3 5	20 7 65535 16 3 16	0 10 1023 10 3 10
13 7 65535 16 3 16	21 0 0 3 0 0	0 11 2047 11 3 11
14 0 0 3 0 0	21 1 3 5 1 0	0 12 4095 12 3 12
14 1 3 5 1 0	21 2 5 7 1 1	0 13 8191 13 3 13
14 2 5 7 1 1	21 3 7 3 3 2	0 14 16383 14 3 14
14 3 7 3 3 2	21 4 9 10 1 3	0 15 32767 15 3 15
14 4 9 10 1 3	21 5 15 4 3 4	1 0 0 4 0 0
14 5 15 4 3 4	21 6 31 5 3 5	1 1 3 5 1 0
14 6 31 5 3 5	21 7 65535 16 3 16	1 2 5 7 1 1
14 7 65535 16 3 16	22 0 0 3 0 0	1 3 9 10 1 3
15 0 0 3 0 0	22 1 3 5 1 0	1 4 15 4 3 4
15 1 3 5 1 0	22 2 5 7 1 1	1 5 31 5 3 5
15 2 5 7 1 1	22 3 7 3 3 2	1 6 63 6 3 6
15 3 7 3 3 2	22 4 9 10 1 3	1 7 127 7 3 7
15 4 9 10 1 3	22 5 15 4 3 4	1 8 255 8 3 8
15 5 15 4 3 4	22 6 31 5 3 5	1 9 511 9 3 9
15 6 31 5 3 5	22 7 65535 16 3 16	1 10 1023 10 3 10
15 7 65535 16 3 16	23 0 0 2 0 0	1 11 2047 11 3 11
16 0 0 3 0 0	23 1 3 5 1 0	1 12 4095 12 3 12
16 1 3 5 1 0	23 2 5 7 1 1	1 13 8191 13 3 13
16 2 5 7 1 1	23 3 65535 16 3 16	1 14 16383 14 3 14
16 3 7 3 3 2	24 0 0 2 0 0	1 15 32767 15 3 15
16 4 9 10 1 3	24 1 3 5 1 0	2 0 0 3 0 0
16 5 15 4 3 4	24 2 5 7 1 1	2 1 3 5 1 0
16 6 31 5 3 5	24 3 65535 16 3 16	2 2 5 7 1 1
16 7 65535 16 3 16	25 0 0 2 0 0	2 3 9 10 1 3
17 0 0 3 0 0	25 1 3 5 1 0	2 4 15 4 3 4
17 1 3 5 1 0	25 2 5 7 1 1	2 5 31 5 3 5
17 2 5 7 1 1	25 3 65535 16 3 16	2 6 63 6 3 6
17 3 7 3 3 2	26 0 0 2 0 0	2 7 127 7 3 7
17 4 9 10 1 3	26 1 3 5 1 0	3 0 0 3 0 0
17 5 15 4 3 4	26 2 5 7 1 1	3 1 3 5 1 0
17 6 31 5 3 5	26 3 65535 16 3 16	3 2 5 7 1 1
17 7 65535 16 3 16	27 0 0 2 0 0	3 3 9 10 1 3
18 0 0 3 0 0	27 1 3 5 1 0	3 4 15 4 3 4
18 1 3 5 1 0	27 2 5 7 1 1	3 5 31 5 3 5
18 2 5 7 1 1	27 3 65535 16 3 16	3 6 63 6 3 6
18 3 7 3 3 2	28 0 0 2 0 0	3 7 127 7 3 7
18 4 9 10 1 3	28 1 3 5 1 0	4 0 0 3 0 0
18 5 15 4 3 4	28 2 5 7 1 1	4 1 3 5 1 0
18 6 31 5 3 5	28 3 65535 16 3 16	4 2 5 7 1 1
18 7 65535 16 3 16	29 0 0 2 0 0	4 3 9 10 1 3
19 0 0 3 0 0	29 1 3 5 1 0	4 4 15 4 3 4
19 1 3 5 1 0	29 2 5 7 1 1	4 5 31 5 3 5



4 6 63 6 3 6	1 13 8191 13 3 13	8 6 63 6 3 6
4 7 127 7 3 7	1 14 16383 14 3 14	8 7 127 7 3 7
5 0 0 3 0 0	1 15 32767 15 3 15	9 0 0 3 0 0
5 1 3 5 1 0	2 0 0 3 0 0	9 1 3 5 1 0
5 2 5 7 1 1	2 1 3 5 1 0	9 2 5 7 1 1
5 3 9 10 1 3	2 2 5 7 1 1	9 3 9 10 1 3
5 4 15 4 3 4	2 3 9 10 1 3	9 4 15 4 3 4
5 5 31 5 3 5	2 4 15 4 3 4	9 5 31 5 3 5
5 6 63 6 3 6	2 5 31 5 3 5	9 6 63 6 3 6
5 7 127 7 3 7	2 6 63 6 3 6	9 7 127 7 3 7
6 0 0 3 0 0	2 7 127 7 3 7	10 0 0 3 0 0
6 1 3 5 1 0	3 0 0 3 0 0	10 1 3 5 1 0
6 2 5 7 1 1	3 1 3 5 1 0	10 2 5 7 1 1
6 3 9 10 1 3	3 2 5 7 1 1	10 3 9 10 1 3
6 4 15 4 3 4	3 3 9 10 1 3	10 4 15 4 3 4
6 5 31 5 3 5	3 4 15 4 3 4	10 5 31 5 3 5
6 6 63 6 3 6	3 5 31 5 3 5	10 6 63 6 3 6
6 7 127 7 3 7	3 6 63 6 3 6	10 7 127 7 3 7
7 0 0 3 0 0	3 7 127 7 3 7	11 0 0 3 0 0
7 1 3 5 1 0	4 0 0 3 0 0	11 1 3 5 1 0
7 2 5 7 1 1	4 1 3 5 1 0	11 2 5 7 1 1
7 3 9 10 1 3	4 2 5 7 1 1	11 3 9 10 1 3
7 4 15 4 3 4	4 3 9 10 1 3	11 4 15 4 3 4
7 5 31 5 3 5	4 4 15 4 3 4	11 5 31 5 3 5
7 6 63 6 3 6	4 5 31 5 3 5	11 6 63 6 3 6
7 7 127 7 3 7	4 6 63 6 3 6	11 7 127 7 3 7
alloc_3	4 7 127 7 3 7	
12	5 0 0 3 0 0	
0 0 0 4 0 0	5 1 3 5 1 0	
0 1 3 5 1 0	5 2 5 7 1 1	
0 2 5 7 1 1	5 3 9 10 1 3	
0 3 9 10 1 3	5 4 15 4 3 4	
0 4 15 4 3 4	5 5 31 5 3 5	
0 5 31 5 3 5	5 6 63 6 3 6	
0 6 63 6 3 6	5 7 127 7 3 7	
0 7 127 7 3 7	6 0 0 3 0 0	
0 8 255 8 3 8	6 1 3 5 1 0	
0 9 511 9 3 9	6 2 5 7 1 1	
0 10 1023 10 3 10	6 3 9 10 1 3	
0 11 2047 11 3 11	6 4 15 4 3 4	
0 12 4095 12 3 12	6 5 31 5 3 5	
0 13 8191 13 3 13	6 6 63 6 3 6	
0 14 16383 14 3 14	6 7 127 7 3 7	
0 15 32767 15 3 15	7 0 0 3 0 0	
1 0 0 4 0 0	7 1 3 5 1 0	
1 1 3 5 1 0	7 2 5 7 1 1	
1 2 5 7 1 1	7 3 9 10 1 3	
1 3 9 10 1 3	7 4 15 4 3 4	
1 4 15 4 3 4	7 5 31 5 3 5	
1 5 31 5 3 5	7 6 63 6 3 6	
1 6 63 6 3 6	7 7 127 7 3 7	
1 7 127 7 3 7	8 0 0 3 0 0	
1 8 255 8 3 8	8 1 3 5 1 0	
1 9 511 9 3 9	8 2 5 7 1 1	
1 10 1023 10 3 10	8 3 9 10 1 3	
1 11 2047 11 3 11	8 4 15 4 3 4	
1 12 4095 12 3 12	8 5 31 5 3 5	

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

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.3 Encoder and Decoder

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, 508-493-2241, e-mail: pan@gauss.enet.dec.com      *
*                                                              *
* VERSION 2.5                                                  *
* 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 *

```



```

* 8jul92 Susanne Ritscher   MS-DOS, MSC 6.0 port fixes.      *
* 19aug92 Soren H. Nielsen   Fixed bug in I_CRC_calc and in  *
*                               II_CRC_calc. Added function: new_ext *
*                               for better MS-DOS compatability *
*26nov92 Susanne Ritscher     AIFF for MS-DOS                *
*****
*                               *
*                               *
* MPEG/audio Phase 2 coding/decoding multichannel          *
*                               *
* 7/27/93   Susanne Ritscher, IRT Munich                    *
*                               *
* 8/27/93   Susanne Ritscher, IRT Munich                    *
* Channel-Switching is working                               *
* 9/1/93    Susanne Ritscher, IRT Munich                    *
* all channels normalized                                     *
* 9/20/93   channel-switching is only performed at a        *
* certain limit of TC_ALLOC dB, which is included          *
* in encoder.h                                              *
*                               *
* Version 1.0 Shareware                                     *
*                               *
* 07/12/94   Susanne Ritscher, IRT Munich                    *
* Tel: +49 89 32399 458                                       *
* Fax: +49 89 32399 415                                       *
*****/

/******

*
* 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][16] = {
                                {0,32,64,96,128,160,192,224,256,288,320,352,384,416,448,1000},

```

```

        {0,32,48,56,64,80,96,112,128,160,192,224,256,320,384,1000},
        {0,32,40,48,56,64,80,96,112,128,160,192,224,256,320,1000}
    };

double /*far*/ multiple[64] = {
2.000000000000000, 1.58740105196820, 1.25992104989487,
1.000000000000000, 0.79370052598410, 0.62996052494744, 0.500000000000000,
0.39685026299205, 0.31498026247372, 0.250000000000000, 0.19842513149602,
0.15749013123686, 0.125000000000000, 0.09921256574801, 0.07874506561843,
0.062500000000000, 0.04960628287401, 0.03937253280921, 0.031250000000000,
0.02480314143700, 0.01968626640461, 0.015625000000000, 0.01240157071850,
0.00984313320230, 0.007812500000000, 0.00620078535925, 0.00492156660115,
0.003906250000000, 0.00310039267963, 0.00246078330058, 0.001953125000000,
0.00155019633981, 0.00123039165029, 0.000976562500000, 0.00077509816991,
0.00061519582514, 0.000488281250000, 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
};

int sb_groups[12] = { 0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 26 /*31*/};

int transmission_channel[8][5] = {
    {0, 1, 2, 3, 4},
    {0, 1, 5, 3, 4},
    {0, 1, 6, 3, 4},
    {0, 1, 2, 5, 4},
    {0, 1, 2, 3, 6},
    {0, 1, 2, 5, 6},
    {0, 1, 6, 5, 4},
    {0, 1, 5, 3, 6},
};

/******
*
* 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 fullname[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 separator */

#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/
/* quantization parameters for each subband in layer II encoding
/*
/******

int read_bit_alloc(table, alloc) /* read in table, return # subbands */
int table;
al_table *alloc;
{
    long 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(0);
}

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, "%ld %ld %ld %ld %ld %ld\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 + fr_ps->stereomc);
    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);
    }
    printf("sblim = %d, table = %d, br_per_ch = %d\n", sblim, table, br_per_ch);
    return sblim;
}

int js_bound(layer, m_ext)
int layer, m_ext;
{
    static int jsb_table[3][4] = { { 4, 8, 12, 16 }, { 4, 8, 12, 16 },
                                    { 0, 4, 8, 16 } }; /* layer+m_ext -> jsbound */

    if(layer<1 || layer >3 || m_ext<0 || m_ext>3) {
        fprintf(stderr, "js_bound bad layer/modext (%d/%d)\n", layer, m_ext);
        exit(0);
    }
    return(jsb_table[layer-1][m_ext]);
}

int js_bound1(layer, m_ext) /* other boundaries for multichannel*/
int layer, m_ext;          /* diese laut Gerhards paper, nicht fix*/
{
    static int jsb_table[3][8] = { { 0, 0, 0, 0, 0, 0, 0, 0 }, { 4, 5, 6, 7, 8, 10, 12, 16 },
                                    { 0, 0, 0, 0, 0, 0, 0, 0 } }; /* layer+m_ext -> jsbound */

    if(layer<1 || layer >3 || m_ext<0 || m_ext>3) {
        fprintf(stderr, "js_bound bad layer/modext (%d/%d)\n", layer, m_ext);
        exit(0);
    }
    return(jsb_table[layer-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;
    if(hdr->mode != MPG_MD_NONE)
        fr_ps->stereo = (hdr->mode == MPG_MD_MONO) ? 1 : 2;
    else fr_ps->stereo = 0;
    fr_ps->stereomc = 3;
    if (hdr->layer == 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->layer, hdr->mode_ext);
    else    fr_ps->jsbound = fr_ps->sblimit;
            fr_ps->jsboundmc = 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[7][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[7][SBLIMIT], scfsi[7][SBLIMIT], scalar[7][3][SBLIMIT];
frame_params *fr_ps;
FILE *s;
{
int stereo = fr_ps->stereo;
int sblimit = fr_ps->sblimit;
int jsbound = fr_ps->jsbound;
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,j,k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;

    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;
```

```

{
    int ans = 0;
    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 == 44100)
        return(0);
    else if(sRate == 48000)
        return(1);
    else if(sRate == 32000)
        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 *mem_alloc(block, item)
unsigned long block;
char *item;
{
#ifdef MSDOS
void *ptr;
#else
void _far *ptr;
#endif

#ifdef MACINTOSH
    ptr = NewPtr(block);
#endif

#ifdef MSDOS
    ptr = (void _far *)_fmalloc((unsigned int)block); /* far memory, 92-07-08 sr */
#endif

```



```

    #if ! defined (MACINTOSH) && ! defined (MSDOS)
        ptr = (void *) malloc(block);
    #endif
        if (ptr != NULL){
    #ifdef MSDOS
        _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;
    {
        /*int i = 0;
        while(1)
        {
            if( &*ptr_addr++ )
                ++i;
            else break;
        }
        printf("%d\n", i);*/

        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

/* fixed bus alignment error, HP 27-may-93 */

register unsigned long top2bits;

register unsigned short *ps2;
register IEEE_DBL *p_dbl;
register SANE_EXT *p_ext;
SANE_EXT ext_align;
char *c_align;
int i;

    p_dbl = (IEEE_DBL *) pd;
    p_ext = &ext_align;
    top2bits = p_dbl->hi & 0xc0000000;
    p_ext->i1 = ((p_dbl->hi >> 4) & 0x3ff0000) | top2bits;
    p_ext->i1 |= ((p_dbl->hi >> 5) & 0x7fff) | 0x8000;
    p_ext->i2 = (p_dbl->hi << 27) & 0xf8000000;
    p_ext->i2 |= ((p_dbl->lo >> 5) & 0x07ffffff);
    ps2 = (unsigned short *) &(p_dbl->lo);
    ps2++;
    p_ext->s1 = (*ps2 << 11) & 0xf800;

    c_align = (char *) p_ext;
    for (i=0;i<10;i++)
        ps[i] = c_align[i];

#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

/* fixed bus alignment error, HP 27-may-93 */

register unsigned long top2bits;

register IEEE_DBL      *p_dbl;
register SANE_EXT      *p_ext;
SANE_EXT ext_align;
char *c_align;
int i;

    p_dbl = (IEEE_DBL *) pd;
    p_ext = &ext_align;

    c_align = (char *) p_ext;
    for (i=0;i<10;i++)
        c_align[i] = ps[i];

    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, byte_per_sample)
FILE *file_ptr;
IFF_AIFF *aiff_ptr;
int *byte_per_sample;

{

```

```

register char i;
register long seek_offset;

char temp_sampleRate[10];
char *dummy;
char holder;
Chunk FormChunk;
CommonChunk CommChunk;
SoundDataChunk SndDChunk;
identifier ident;

if (fseek(file_ptr, 0, SEEK_SET) != 0)
    return(-1);

if (fread(&FormChunk, sizeof(Chunk), 1, file_ptr) != 1)
    return(-1);

#ifdef MSDOS

    holder = FormChunk.ckID[0];
    FormChunk.ckID[0] = FormChunk.ckID[3];
    FormChunk.ckID[3] = holder;
    holder = FormChunk.ckID[1];
    FormChunk.ckID[1] = FormChunk.ckID[2];
    FormChunk.ckID[2] = holder;

/* fixed bug in next line, HP 27-may-93 */
    holder = FormChunk.formType[0];
    FormChunk.formType[0] = FormChunk.formType[3];
    FormChunk.formType[3] = holder;
    holder = FormChunk.formType[1];
    FormChunk.formType[1] = FormChunk.formType[2];
    FormChunk.formType[2] = holder;

    FormChunk.ckSize = _lrotl(FormChunk.ckSize, 8);
#endif

/* fixed bug in next line, HP 27-may-93 */
    if (strncmp(FormChunk.ckID,IFF_ID_FORM,4) != 0 ||
        strncmp(FormChunk.formType,IFF_ID_AIFF,4) != 0)
        return(-1); /* warning: different levels of indirection.7/8/92.sr*/

    if (fread(&ident,sizeof(identifier), 1, file_ptr) != 1)
        return(-1);

#ifdef MSDOS

    holder = ident.name[0];
    ident.name[0] = ident.name[3];
    ident.name[3] = holder;
    holder = ident.name[1];
    ident.name[1] = ident.name[2];
    ident.name[2] = holder;

    ident.ck_length = _lrotl(ident.ck_length, 8);

```

```

#endif

/* fixed bug in next line, HP 27-may-93 */
while(strncmp(ident.name,IFF_ID_COMM,4) != 0)
{
    dummy = calloc( ident.ck_length, sizeof(char));
/* changed "fread( &dummy," to "fread ( dummy," , HP 26-may-93 */
    if(fread( dummy, ident.ck_length, 1, file_ptr) != 1)
        return(-1);
    free(dummy);
/* fixed bug in next line, HP 27-may-93 */
    if(fread( &ident, sizeof(identifier),1, file_ptr) != 1)
        return(-1);

#ifdef MSDOS

    holder = ident.name[0];
    ident.name[0] = ident.name[3];
    ident.name[3] = holder;
    holder = ident.name[1];
    ident.name[1] = ident.name[2];
    ident.name[2] = holder;

    ident.ck_length = _lrotl(ident.ck_length, 8);

#endif
}

for( i = 0; i < 4; ++i)
    CommChunk.ckID[i] = ident.name[i];

    CommChunk.ckSize = ident.ck_length;

    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);

#ifdef MSDOS

    CommChunk.sampleSize = _rotr(CommChunk.sampleSize, 8);
    CommChunk.ckSize = _lrotl(CommChunk.ckSize, 8);
    CommChunk.numChannels = _rotr(CommChunk.numChannels, 8);
    CommChunk.numSampleFrames = _lrotl(CommChunk.numSampleFrames, 8);

#endif

```

```

*byte_per_sample = ceil((double)CommChunk.sampleSize / 8);

    for (i = 0; i < sizeof(char[10]); i++)
        temp_sampleRate[i] = CommChunk.sampleRate[i];

    extended_to_double(temp_sampleRate, &aiff_ptr->sampleRate);

/* to start the search again from the beginning, HP 27-may-93 */
    fseek (file_ptr, sizeof(Chunk), SEEK_SET);

    if (fread(&ident, sizeof(identifier), 1, file_ptr) != 1)
        return(-1);

#ifdef MSDOS

    holder = ident.name[0];
    ident.name[0] = ident.name[3];
    ident.name[3] = holder;
    holder = ident.name[1];
    ident.name[1] = ident.name[2];
    ident.name[2] = holder;

    ident.ck_length = _lrotl(ident.ck_length, 8);

#endif

/* fixed bug in next line, HP 27-may-93 */
    while(strncmp(ident.name,IFF_ID_SSND,4) != 0)
    {
        dummy = calloc( ident.ck_length, sizeof(char));
/* changed "fread( &dummy," to "fread ( dummy," HP 26-may-93 */
        if(fread( dummy, ident.ck_length, 1, file_ptr) != 1)
            return(-1);
        free(dummy);
        if(fread( &ident, sizeof(identifier),1, file_ptr) != 1)
            return (-1);
/* the following lines are not necessary, HP 27-may-93 */
/*
        {
            fseek(file_ptr, 0, SEEK_SET);
            if(fread( &ident, sizeof(identifier), 1, file_ptr) != 1)
                return(-1);
        }
*/

#ifdef MSDOS

    holder = ident.name[0];
    ident.name[0] = ident.name[3];
    ident.name[3] = holder;
    holder = ident.name[1];
    ident.name[1] = ident.name[2];
    ident.name[2] = holder;

    ident.ck_length = _lrotl(ident.ck_length, 8);

```

```

#endif
    }

    for(i = 0; i < 4; ++i)
        SndDChunk.ckID[i] = ident.name[i];

    SndDChunk.ckSize = ident.ck_length;

    if (fread(&SndDChunk.offset, sizeof(unsigned long), 1, file_ptr) != 1)
        return(-1);

    if (fread(&SndDChunk.blockSize, sizeof(unsigned long), 1,
file_ptr) != 1)
        return(-1);

#ifdef MSDOS

    SndDChunk.offset = _lrotl(SndDChunk.offset, 8);
    SndDChunk.blockSize = _lrotl(SndDChunk.blockSize, 8);

#endif

/* why seek behinde the SSND Chunk ???, HP 27-may-93 */
/*
    seek_offset = SndDChunk.ckSize - sizeof(SoundDataChunk) +
        sizeof(ChunkHeader);

    if (fseek(file_ptr, seek_offset, SEEK_CUR) != 0)
        return(-1);
*/

    aiff_ptr->numChannels    = CommChunk.numChannels;
    aiff_ptr->numSampleFrames = CommChunk.numSampleFrames;
    aiff_ptr->sampleSize     = CommChunk.sampleSize;
    aiff_ptr->blkAlgn.offset  = SndDChunk.offset;
    aiff_ptr->blkAlgn.blockSize = SndDChunk.blockSize;
    strncpy(aiff_ptr->sampleType, SndDChunk.ckID, 4);

    return(0);
}

/******
*
* 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);
    else

```



```

    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;

    strcpy( FormChunk.ckID, IFF_ID_FORM);
    strcpy( FormChunk.formType, IFF_ID_AIFF);
    strcpy( CommChunk.ckID, IFF_ID_COMM); /*7/7/93,SR,changed FormChunk to
CommChunk*/

    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;
    strncpy(/*(unsigned long *)*/ SndDChunk.ckID, aiff_ptr->sampleType, 4);

    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_alig_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_alig_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_struct *bs; /* bit stream structure */
{
    register int i=bs->buf_size-2-bs->buf_byte_idx;
    register unsigned long n;
    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_struct *bs; /* bit stream structure */
int minimum;           /* end of the buffer to empty */
{
    register int i;

#ifdef 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_struct *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, "w+")) == NULL) {
        printf("Could not create \"%s\".\n", bs_filenam);
        exit(0);
    }
    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_struct *bs; /* bit stream structure */
char *bs_filenam; /* name of the bit stream file */
int size; /* size of the buffer */
{
    register unsigned long n;
    register int i=0;
    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(0);
    }

    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 */
                break;

            default: /* detection of an binary character */
                flag--;
                i = 300;
                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(0);
}

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_struct *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_struct *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_struct *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_struct *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_struct *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_struct *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_struct *bs; /* bit stream structure */
int bit; /* bit to write into the buffer */
{
    register int i;

    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_struct *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;
    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_struct *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_struct *bs; /* bit stream structure */
unsigned int val; /* val to write into the buffer */
int N; /* number of bits of val */
{
    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) {
        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_aligned_putbits(bs, val, N)
Bit_stream_struct *bs; /* bit stream structure */
unsigned int val; /* val to write into the buffer */

```

```

int N;          /* number of bits of val */
{
    unsigned long aligning, sstell();
    register int tmp =0;

    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)

#ifdef MSDOS
        putbits(bs, (unsigned int)0, (int)(8-aligning)); /* casting, 92-07-08 sr */
#else
        putbits(bs, 0, 8-aligning);
#endif

    putbits(bs, val, N);
}

/*read the next byte aligned N bits from the bit stream */
unsigned long byte_ali_getbits(bs, N)
Bit_stream_struct *bs; /* bit stream structure */
int N;          /* number of bits of val */
{
    unsigned long aligning, sstell();
    unsigned long val;

    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, 8-aligning);

    return(getbits(bs, N));
}

/*return the current bit stream length (in bits)*/
unsigned long sstell(bs)
Bit_stream_struct *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_struct *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_struct *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 = stell(bs)%ALIGNING;
    if (aligning)
        /* getbits(bs, ALIGNING-aligning);*/ /*long/short casting.7/8/92.sr*/
        getbits(bs, (int) (ALIGNING-aligning)); /*sr*/

    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[7][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[7][SBLIMIT], scfsi[7][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 II_CRC_calcmc(fr_ps, bit_alloc, scfsi, crc)
frame_params *fr_ps;
unsigned int bit_alloc[7][SBLIMIT], scfsi[7][SBLIMIT];
unsigned int *crc;
{
    int i, k, m, l;
    layer *info = fr_ps->header;
    int stereo = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsboundmc;
    al_table *alloc = fr_ps->alloc;

    *crc = 0xffff; /* changed from '0' 92-08-11 shn */
    update_CRC(info->center, 2, crc);
    update_CRC(info->surround, 2, crc);
    update_CRC(info->lfe, 1, crc);
    update_CRC(info->audio_mix, 1, crc);
    update_CRC(info->matrix, 2, crc);
    update_CRC(info->multiling_ch, 3, crc);
    update_CRC(info->multiling_fs, 1, crc);

```

```

update_CRC(info->multiling_lay, 1, crc);
update_CRC(info->ext_bit_stream_present, 1, crc);
if(info->ext_bit_stream_present == 1)
{
    fprintf(stderr, "not done yet Extension Bitstream!!!\n");
    exit(0);
}

update_CRC(info->tc_sbgr_select, 1, crc);
update_CRC(info->dyn_cross_on, 1, crc);
update_CRC(info->mc_prediction_on, 1, crc);
if(info->tc_sbgr_select == 1)
    update_CRC(info->tc_allocation, 3, crc);
else
    for(i = 0; i < 12; i++)
        update_CRC(info->tc_alloc[i], 3, crc);

if(info->dyn_cross_on == 1)
{
    fprintf(stderr, "not done yet Dynamic Crosstalk!!!\n");
    exit(0);
}

if(info->mc_prediction_on == 1)
{
    fprintf(stderr, "not done yet Prediction!!!\n");
    exit(0);
}

for (i=0;i<sblimit;i++)
    if( i < jsbound)
        for(m = 2; m < 5; ++m)
        {
            if(fr_ps->header->tc_sbgr_select == 1)
                k = transmission_channel[fr_ps->header->tc_allocation][m];
            else
            {
                if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
                else
                {
                    for(l = 1; l < 12; l++)
                    {
                        if((sb_groups[l-1] < i) && (i <= sb_groups[l])) /* bug!!
8/21/93,SR*/
                        {
                            k = transmission_channel[fr_ps->header->tc_alloc[l]][m];
                            break;
                        }
                    }
                }
            }

            if((i < 12) || (k != 2) || (fr_ps->header->center != 3))
                update_CRC(bit_alloc[k][i], (*alloc)[i][0].bits, crc);
        }
    }
}

```

```

    for (i=0;i<sblimit;i++)
        for(m = 2; m < 5; ++m)
        {
            if(fr_ps->header->tc_sbgr_select == 1)
                k = transmission_channel[fr_ps->header->tc_allocation][m];
            else
            {
                if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
                else
                {
                    for(l = 1; l < 12; l++)
                    {
                        if((sb_groups[l-1] < i) && (i <= sb_groups[l])) /* bug!! 8/21/93,SR*/
                        {
                            k = transmission_channel[fr_ps->header->tc_alloc[l]][m];
                            break;
                        }
                    }
                }
            }
        }

        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 MSDOS
/* -----
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 '\\':
            case '/':
            case ':': found=-1; break;
            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
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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
* *
* VERSION 2.5 *
* 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. *
* 8 jul 92 Susanne Ritscher MS-DOS, MSC 6.0 port fixes. *
* 19 aug 92 Soren H. Nielsen Printout of bit allocation. *
* UNIX port fixes. MS-DOS file name *
* extensions fix *
* dec 92 Susanne Ritscher Changed to multi-channel with several *
* options *
*****/
*
*
* MPEG/audio Phase 2 coding/decoding multichannel *
*
* 7/27/93 Susanne Ritscher, IRT Munich *
* 8/27/93 Susanne Ritscher, IRT Munich *
* Channel-Switching is working *
* 9/1/93 Susanne Ritscher, IRT Munich *

```



```

*          all channels normalized          *
* 9/20/93    channel-switching is only performed at a      *
*          certain limit of TC_ALLOC dB, which is included *
*          in encoder.h          *
*
*          *
* Version 1.0 Shareware          *
*          *
* 07/12/94    Susanne Ritscher, IRT Munich          *
*          Tel: +49 89 32399 458          *
*          Fax: +49 89 32399 415          *
*****/

/*****
*
* Global Conditional Compile Switches
*
*****/
/*#define PRINTOUT*/
#define UNIX          /* Unix conditional compile switch */
/*#define MACINTOSH          /* Macintosh conditional compile switch */
/*#define MS_DOS          /* IBM PC conditional compile switch
Microsoft C ver. 6.0 */

#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 */
/* #define PROTO_ARGS          /* unix gcc uses arg. prototypes */
#endif /* UNIX */

#ifdef MACINTOSH
/* #define TABLES_PATH ":tables:" /* where to find data files */
#define PROTO_ARGS          /* Mac uses argument prototypes */
#endif /* MACINTOSH */

#ifdef MS_DOS
#define PROTO_ARGS /* DOS uses argument prototypes */
#endif /* MS_DOS */

/* 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> */ /* removed 92-08-05 shn */
#include <stdlib.h> /* put in 92-08-05 shn */

```

```

#endif /* UNIX */

#ifdef MACINTOSH
#include <stdlib.h>
#include <console.h>
#endif /* MACINTOSH */

#ifdef MS_DOS
/* #include <alloc.h> */ /* removed 92-07-08 sr */
#include <malloc.h> /* put in 92-07-08 sr */
/* #include <mem.h> */ /* removed 92-07-08 sr */
#include <stdlib.h>
#endif /* MS_DOS */

/*****
*
* Global Definitions
*
*****/

/* General Definitions */

#ifdef MS_DOS
#define FLOAT double
#else
#define FLOAT float
#endif

#define FALSE 0
#define TRUE (!FALSE)
#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 FFT_SIZE 1024
#define HAN_SIZE 512
#define SCALE_BLOCK 12
#define SCALE_RANGE 64
#define SCALE 32768
#define CRC16_POLYNOMIAL 0x8005

#define MAX_PRED_COEFF 3.96875 /* gleichfoermige Quantisierung der */
/* Praediktorkoeff. im Bereich */

```

```

/* [-MAX_PRED_COEF .. +MAX_PRED_COEF] */

/* 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
#define MPG_MD_NONE        4

/* Multi-channel Definitions - Mode Values */

#define MPG_MC_STEREO      0

/* AIFF Definitions */

#ifndef MS_DOS
#define IFF_ID_FORM        "FORM" /* HP400 unix v8.0: double quotes 1992-07-24 shn */
#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      BINARY /* 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;
    int center;
    int surround;
    int audio_mix;
    int matrix;
    int lfe;
    int multiling_ch;
    int multiling_fs;
    int multiling_lay;
    int ext_bit_stream_present;
    int tc_alloc[12];
    int mc_prediction_on;
    int tc_sbgr_select;
    int tc_allocation;
    int dyn_cross_on;
} layer, *the_layer;

/* Parent Structure Interpreting some Frame Parameters in Header */

typedef struct {
    layer          *header;
    int            actual_mode;
    int            actual_modemc;
    al_table       *alloc;
    int            tab_num;
    int            stereo;
    int            stereomc;
    int            jsbound;
    int            jsboundmc;
    double         mnrr_min;
    int            sblimit;
} 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 identifier_struct{
    ID name;
    long ck_length;
}identifier;

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;
        ID/*char**/ sampleType;/*must be allocated 21.6.93 SR*/
        blockAlign blkAlgn;
    } IFF_AIFF;

/* "bit_stream.h" Type Definitions */

typedef struct bit_stream_struct {
    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_struct;

/*****
 *
 * Global Variable External Declarations
 *
 *****/

extern char      *mode_names[4];
extern char      *layer_names[3];
extern double    s_freq[4];
extern int       bitrate[3][16];
extern double    multiple[64];
extern int       sb_groups[12];
extern int       transmission_channel[8][5];

/*****
 *
 * 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     *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*, int*);
extern int      aiff_seek_to_sound_data(FILE*, /*int*/);
extern int      aiff_write_headers(FILE*, IFF_AIFF*);
extern int      refill_buffer(Bit_stream_struct*);
extern void     empty_buffer(Bit_stream_struct*, int);
extern void     open_bit_stream_w(Bit_stream_struct*, char*, int);
extern void     open_bit_stream_r(Bit_stream_struct*, char*, int);
extern void     close_bit_stream_r(Bit_stream_struct*);
extern void     close_bit_stream_w(Bit_stream_struct*);
extern void     alloc_buffer(Bit_stream_struct*, int);
extern void     dealloc_buffer(Bit_stream_struct*);
extern void     back_track_buffer(Bit_stream_struct*, int);
extern unsigned int  get1bit(Bit_stream_struct*);
extern void     put1bit(Bit_stream_struct*, int);
extern unsigned long look_ahead(Bit_stream_struct*, int);
extern unsigned long getbits(Bit_stream_struct*, int);
extern void     putbits(Bit_stream_struct*, unsigned int, int);
extern void     byte_ali_putbits(Bit_stream_struct*, unsigned int, int);
extern unsigned long byte_ali_getbits(Bit_stream_struct*, int);
extern unsigned long sstell(Bit_stream_struct*);
extern int      end_bs(Bit_stream_struct*);
extern int      seek_sync(Bit_stream_struct*, 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*, long);

#ifdef MACINTOSH
extern void     set_mac_file_attr(char[MAX_NAME_SIZE], short, OsType,
        OsType);
#endif

#ifdef MSDOS
extern char     *new_ext(char *filename, char *extname); /* 92-08-19 shn */
#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     *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     dealloc_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();

#ifdef MSDOS
extern char      *new_ext(); /* 92-08-19 shn */
#endif

#endif
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, 508-493-2241, e-mail: pan@gauss.enet.dec.com      *
*                                                              *
* VERSION 2.5                                                *
* 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.
*****
*
*
* MPEG/audio Phase 2 coding/decoding multichannel
*
* Version 1.0
*
* 7/27/93  Susanne Ritscher, IRT Munich
*
*          thanks to
*          Ralf Schwalbe,  Telekom FTZ Berlin
*          Heiko Purnhagen, Uni Hannover
*
*
* Version 2.0
*
* 8/27/93  Susanne Ritscher, IRT Munich
*          Channel-Switching is working
*
* Version 2.1
*
* 9/1/93   Susanne Ritscher, IRT Munich

```

```

*          all channels normalized          *
*
*
* Version 3.0
*
* 06/16/94   Ralf Schwalbe, Telekom FTZ Berlin      *
*             changed all sources and variables due to MPEG2 - *
*             DIS of March 1994                      *
*             - dematrix and denormalize procedure      *
*             - new tc - allocation (0-7)              *
*             - some new structures and variables as a basis *
*             for further decoding modes                *
*
*
*
*
* Version 1.0 Shareware
*
* 07/12/94   Ralf Schwalbe, Telekom FTZ Berlin      *
*             Tel: +49 30 6708 2406                  *
*             Fax: +49 30 6774 539                    *
*****/

/*****/

#include "common.h"
#include "decoder.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_struct *bs;
frame_params *fr_ps;
{
    layer *hdr = fr_ps->header;

    hdr->version = get1bit(bs);
    hdr->lay = 4-getbits(bs,2);
    hdr->error_protection = !get1bit(bs); /* error protect. TRUE/FALSE */
    hdr->bitrate_index = getbits(bs,4);
    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);
}

/*****
/*
/* 7.7.93 Susanne Ritscher Systeminformation for multi-channel */
/* 27.5.94 Ralf Schwalbe Systeminformation and names due to */
/* MPEG 2 DIS from March 1994 */
/*
*****/

void mc_header(bs, fr_ps)
Bit_stream_struct *bs;
frame_params *fr_ps;
{
    layer *hdr = fr_ps->header;

    hdr->center = getbits(bs, 2);
    hdr->surround = getbits(bs,2);
    hdr->lfe = get1bit(bs);
    hdr->audio_mix = get1bit(bs);
    hdr->dematrix_procedure = getbits(bs,2);
    hdr->no_of_multi_lingual_ch = getbits(bs,3);
    hdr->multi_lingual_fs = get1bit(bs);
    hdr->multi_lingual_layer = get1bit(bs);
    hdr->ext_bit_stream_present = get1bit(bs);
    if( hdr->ext_bit_stream_present == 1)
        hdr->n_ad_bytes = getbits(bs,8);
}

void mc_composite_status_info(bs, fr_ps)
Bit_stream_struct *bs;
frame_params *fr_ps;
{
    layer *hdr = fr_ps->header;
    int sbgr, j, pci;

    hdr->tc_sbgr_select = get1bit(bs);
    hdr->dyn_cross_on = get1bit(bs);
    hdr->mc_prediction_on = get1bit(bs);

    if(hdr->tc_sbgr_select == 1)
    {
        hdr->tc_allocation = getbits(bs,3);/* tc_allocation is valid for */
        for(sbgr = 0; sbgr < 12; sbgr++) /* all sbgr R.S. */
            hdr->tc_alloc[sbgr] = 0;
    }
    else
    {
        hdr->tc_allocation = 0;
        for(sbgr = 0; sbgr < 12; sbgr++)
            hdr->tc_alloc[sbgr] = getbits(bs, 3);
    }
}

```

```

if( hdr->dyn_cross_on == 1)
{
    hdr->dyn_cross_LR = get1bit(bs);
    for(sbgr = 0; sbgr < 12; sbgr++)
        hdr->dyn_cross_mode[sbgr] = getbits(bs,4);
}
else
    for(sbgr = 0; sbgr < 12; sbgr++)
        hdr->dyn_cross_mode[sbgr] = 0;

if( hdr->mc_prediction_on == 1)
{
    /* not installed here */
}
}

/*****
/*
/* 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, l, m)
Bit_stream_struct *bs;
unsigned int bit_alloc[5][SBLIMIT];
frame_params *fr_ps;
int *l, *m;
{
    int i,j;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    al_table *alloc = fr_ps->alloc;

    for (i=0;i<jsbound;i++)
        for (j=*l;j<*m;j++)
        {
            if((fr_ps->header->center != 3) || (i < 12) || (j !=2))
                bit_alloc[j][i] = (char) getbits(bs,(*alloc)[i][0].bits);
            else
                bit_alloc[j][i] = 0;
        }

    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=*l;j<*m;j++)
        bit_alloc[j][i] = 0;
}

/***** Layer I *****/

void I_decode_bitalloc(bs, bit_alloc, fr_ps)
Bit_stream_struct *bs;

```

```

unsigned int bit_alloc[5][SBLIMIT];
frame_params *fr_ps;
{
    int i,j;
    int stereo = fr_ps->stereo;
    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_struct *bs;
unsigned int bit_alloc[5][SBLIMIT], scale_index[5][3][SBLIMIT];
frame_params *fr_ps;
{
    int i,j;
    int stereo = fr_ps->stereo;

    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, l, m)
Bit_stream_struct *bs;
unsigned int scfsi[5][SBLIMIT], bit_alloc[5][SBLIMIT],
            scale_index[5][3][SBLIMIT];
frame_params *fr_ps;
int *l, *m;
{
    int i,j;

```

```

int px,pci;
int sblimit = fr_ps->sblimit;

for (i=0;i<sblimit;i++) for (j=*1;j<*m;j++)
    if (bit_alloc[j][i]) scfsi[j][i] = (char) getbits(bs,2);
for (i=sblimit;i<SBLIMIT;i++) for (j=*1;j<*m;j++)
    scfsi[j][i] = 0;

for (i = 0; i < sblimit; i++) for (j = *1; j < *m; 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=*1;j<*m;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_struct *bs;
frame_params *fr_ps;
{
    int i,j,k;
    int stereo = fr_ps->stereo;
    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 sample[5][3][SBLIMIT];
unsigned int bit_alloc[5][SBLIMIT];
Bit_stream_struct *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;
}

/***** mc - layer2 stuff *****/
/* 19.10.93 R.S. */

void II_buffer_samplemc(bs,sample,bit_alloc,fr_ps,mc_channel)
unsigned int sample[5][3][SBLIMIT];
unsigned int bit_alloc[5][SBLIMIT];
Bit_stream_struct *bs;
frame_params *fr_ps;
int *mc_channel;
{
    int i,j,k,m,sbgr,l;
    unsigned int nlevels, c=0;
    int sblimit = fr_ps->sblimit;
    al_table *alloc = fr_ps->alloc;

    for (i = 0; i < sblimit; i++)
    {
        if( i == 0) sbgr = 0 ;
        else
            for(l = 1; l < 12; l++)
                if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
                {
                    sbgr = l; break;
                }

        for (j = 2; j < *mc_channel; j++)
        {
            if (bit_alloc[j][i])
            {
                if(fr_ps->header->dyn_cross_on == 0)
                {
                    /* check for grouping in subband */
                    if ((*alloc)[i][bit_alloc[j][i]].group==3)
                    {
                        k = (*alloc)[i][bit_alloc[j][i]].bits;

```



```

        for (m=0;m<3;m++)
            sample[j][m][i] = (unsigned int) getbits(bs,k);
    }
    else
    { /* bit_alloc = 3, 5, 9 */
        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 /* 10.6.94 R.S. not ready yet */
switch(fr_ps->header->dyn_cross_mode[sbgr])
{
    case 8:
    case 10: /* and so one */
        break;
}
/* switch end */
}
else
{ /* for no sample transmitted */
    for (k=0;k<3;k++) sample[j][k][i] = 0;
}
}
}

for (i = sblimit; i < SBLIMIT; i++)
    for (j = 2; j < *mc_channel; j++)
        for (k = 0; k < 3; k++)
            sample[j][k][i] = 0;
}

/*****
/*
/* Restore the compressed sample to a fractional 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, z)
unsigned int sample[5][3][SBLIMIT];
unsigned int bit_alloc[5][SBLIMIT];
double fraction[12][5][3][SBLIMIT];
frame_params *fr_ps;
int *z;
{
    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;
#ifdef MSDOS
            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[*z][k][j][i] = 0.0;
            else fraction[*z][k][j][i] = -1.0;

            /* Form a 2's complement sample */
            fraction[*z][k][j][i] += (double) (sample[k][j][i] & ((1<<x-1)-1)) /
                (double) (1L<<x-1);

            /* Dequantize the sample */
            fraction[*z][k][j][i] += d[( *alloc)[i][bit_alloc[k][i]].quant];
            fraction[*z][k][j][i] *= c[( *alloc)[i][bit_alloc[k][i]].quant];
        }
    else fraction[*z][k][j][i] = 0.0;

    for (i=sblimit; i<SBLIMIT; i++)
        for (j=0; j<3; j++)
            for (k = 0; k < stereo; k++)

```

```

        fraction[*z][k][j][i] = 0.0;
    }

/***** MC Layer II stuff *****/

void II_dequantize_samplemc(sample, bit_alloc, fraction, fr_ps, mc_channel, z)
unsigned int sample[5][3][SBLIMIT];
unsigned int bit_alloc[5][SBLIMIT];
double fraction[12][5][3][SBLIMIT];
frame_params *fr_ps;
int *mc_channel, *z;
{
    int i, j, k, x, sbgr, l;
    int sblimit = fr_ps->sblimit;
    al_table *alloc = fr_ps->alloc;

    for (i = 0; i < sblimit; i++)
    {
        if (i == 0) sbgr = 0;
        else
            for (l = 1; l < 12; l++)
                if ((sb_groups[l-1] < i) && (i <= sb_groups[l]))
                {
                    sbgr = l; break;
                }
        for (j = 0; j < 3; j++)
            for (k = 2; k < *mc_channel; k++)
                if (bit_alloc[k][i])
                {
                    if (fr_ps->header->dyn_cross_on == 0)
                    {
                        /* locate MSB in the sample */
                        x = 0;

#ifdef MSDOS
                        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[*z][k][j][i] = 0.0;
                        else fraction[*z][k][j][i] = -1.0;

                        /* Form a 2's complement sample */
                        fraction[*z][k][j][i] += (double) (sample[k][j][i] & ((1<<x-1)-1)) /
                            (double) (1L<<x-1);

                        /* Dequantize the sample */
                        fraction[*z][k][j][i] += d[(*alloc)[i][bit_alloc[k][i]].quant];
                        fraction[*z][k][j][i] *= c[(*alloc)[i][bit_alloc[k][i]].quant];
                    }
                    else /* 10.6.94 R.S. not ready yet */
                    {

```

```

        switch(fr_ps->header->dyn_cross_mode[sbgr])
        {
            case 8:
            case 10: /* and so one */
                break;
        } /* end switch */
    } /* if bit_alloc */
    else fraction[*z][k][j][i] = 0.0;
}
for (i = sblimit; i < SBLIMIT; i++)
    for (j = 0; j < 3; j++)
        for(k = 2; k < *mc_channel; k++)
            fraction[*z][k][j][i] = 0.0;
}

/***** Layer I stuff *****/

void I_dequantize_sample(sample, fraction, bit_alloc, fr_ps)
unsigned int /*far*/ sample[5][3][SBLIMIT];
unsigned int bit_alloc[5][SBLIMIT];
double /*far*/ fraction[5][3][SBLIMIT];
frame_params *fr_ps;
{
    int i, nb, k;
    int stereo = fr_ps->stereo;

    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, z)
double /*far*/ fraction[12][5][3][SBLIMIT];
unsigned int scale_index[5][3][SBLIMIT];
frame_params *fr_ps;
int x;

```

```

int *z;
{
    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[*z][j][0][i] *= multiple[scale_index[j][x][i]];
        fraction[*z][j][1][i] *= multiple[scale_index[j][x][i]];
        fraction[*z][j][2][i] *= multiple[scale_index[j][x][i]];
    }
}

/***** MC Layer II Stuff *****/

void II_denormalize_samplemc(fraction, scale_index, fr_ps, x, mc_channel, z)
double /*far*/ fraction[12][5][3][SBLIMIT];
unsigned int scale_index[5][3][SBLIMIT];
frame_params *fr_ps;
int x;
int *mc_channel, *z;
{
    int i,j,k,sbgr,l,bl=0;
    int sblimit = fr_ps->sblimit;

    for (i = 0; i < sblimit; i++)
    {
        if (i == 0) sbgr = 0 ;
        else
            for(l = 1; l < 12; l++)
                if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
                {
                    sbgr = l; break;
                }
        if(fr_ps->header->dyn_cross_on == 0)
            for (j = 2; j < *mc_channel; j++)
            {
                fraction[*z][j][0][i] *= multiple[scale_index[j][x][i]];
                fraction[*z][j][1][i] *= multiple[scale_index[j][x][i]];
                fraction[*z][j][2][i] *= multiple[scale_index[j][x][i]];
            }
        else /* 10.6.94 R.S. not ready yet */
            switch(fr_ps->header->dyn_cross_mode[sbgr])
            {
                case 8:
                    break; /* and so one */
            } /* end switch */
    } /* for sblimit */
}

/***** Layer I stuff *****/

void I_denormalize_sample(fraction,scale_index,fr_ps)
double /*far*/ fraction[5][3][SBLIMIT];
unsigned int scale_index[5][3][SBLIMIT];
frame_params *fr_ps;

```

```

{
    int i,j,k;
    int stereo = fr_ps->stereo;

    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,80,fp);
    }
    fclose(fp);
}

```

```

int SubBandSynthesis (bandPtr, channel, samples)
double *bandPtr;
int channel;
long *samples;
{
    long foo;
    register int i,j,k;
    register double *bufOffsetPtr, sum;
    static int init = 1;
    typedef double NN[64][32];
    static NN *filter;
    typedef double BB[5][2*HAN_SIZE];
    static BB *buf;
    static int bufOffset = 64;
    static double *window;
    int clip = 0;          /* count & return how many samples clipped */

    if (init) {
        buf = (BB *) mem_alloc(sizeof(BB),"BB");
        filter = (NN *) mem_alloc(sizeof(NN), "NN");
        create_syn_filter(*filter);
        window = (double *) mem_alloc(sizeof(double) * HAN_SIZE, "WIN");
        read_syn_window(window);
        bufOffset = 64;
        init = 0;
    }
    if (channel == 0) bufOffset = (bufOffset - 64) & 0x3ff;
    bufOffsetPtr = &((*buf)[channel][bufOffset]);

    for (i=0; i<64; i++)
    {
        sum = 0;
        for (k=0; k<32; k++)
            sum += bandPtr[k] * (*filter)[i][k];
        bufOffsetPtr[i] = sum;
    }
}

```

```

    }

    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) & 0x3ff];
        }
        foo = floor(sum * SCALE + 0.5);
        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)
long /*far*/ pcm_sample[5][3][SBLIMIT];
int num;
frame_params *fr_ps;
int done;
FILE *outFile;
unsigned long *psampFrames;
{
    int i,j,l;
    int stereo = fr_ps->stereo;
    int mc_channel = fr_ps->mc_channel;
    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 + mc_channel; 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_struct *bs;
unsigned int *old_crc;
{

```



```

    *old_crc = getbits(bs, 16);
}

void recover_CRC_error(pcm_sample, error_count, fr_ps, outFile, psampFrames)
long /*far*/ pcm_sample[5][3][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 < 12; 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 < 12; i++)
            out_fifo(pcm_sample, num, fr_ps, done, outFile, psampFrames);
    }
}

void dematricing(pcm_sample, fr_ps, p)
double pcm_sample[12][5][3][SBLIMIT];
frame_params *fr_ps;
int *p;
{
    double matr1; /* normalizing factor */
    double matr2; /* matricing factor */
    double matr3; /* matricing factor */
    int i, j, k, tc_alloc, l, sbgr = 0;
    layer *info = fr_ps->header;
    double tmp_sample;

```

```

switch(info->dematrix_procedure)
{
    /* factors according to Committee Draft and telefax 10/20/93 */
    /* by E.Schroeder, DTB Hannover. 01/14/94, SR */

    case 0: matr1 = 2.42797851625;
            matr2 = 2 + 1.41632080078125;
            break;
    case 1: matr1 = 1.5 + (0.5 * 1.41632080078125);
            matr2 = 0.5 * (1.5 + 0.5 * (1.41632080078125));
            matr3 = 0.5 + (0.75 * 1.41632080078125);
            break;
    case 2: matr1 = 2.42797851625;
            matr2 = 2 + 1.41632080078125;
            printf("NOT DONE MATRIX 2 !!\n");
            exit(0);
            break;
    case 3: matr1 = 1.0;
            matr2 = 1.0;
            break;
}

for(j = 0; j < 3; ++j)
{
    for(k = 0; k < SBLIMIT; k++)
    {
        if(fr_ps->header->tc_sbgr_select == 1)
            tc_alloc = fr_ps->header->tc_allocation;
        else
        {
            if(k == 0) sbgr = 0;
            else
                for(l = 1; l < 12; l++)
                {
                    if((sb_groups[l-1] < k) && (k <= sb_groups[l]))
                    {
                        sbgr = l; /* search the valid subband group */
                        break;
                    }
                }
            tc_alloc = fr_ps->header->tc_alloc[sbgr];
            /* no prediction, but different tc_alloc's per subband*/

        } /* else tc_sbgr_select == 0 */

        switch(tc_alloc)
        {
            case 0:
                pcm_sample[*p][0][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][3][j][k];
                pcm_sample[*p][1][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][4][j][k];
                break;

            case 1:

```

```

    tmp_sample = pcm_sample[*p][2][j][k]; /* L */
    pcm_sample[*p][2][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][3][j][k];
    pcm_sample[*p][1][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][4][j][k];
    pcm_sample[*p][0][j][k] = tmp_sample;
    break;

    case 2:
        tmp_sample = pcm_sample[*p][2][j][k]; /* R */
        pcm_sample[*p][2][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][4][j][k];
        pcm_sample[*p][0][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][3][j][k];
        pcm_sample[*p][1][j][k] = tmp_sample;
        break;

    case 3:
        tmp_sample = pcm_sample[*p][3][j][k]; /* L in T3 */
        pcm_sample[*p][3][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][3][j][k] - pcm_sample[*p][2][j][k];
        pcm_sample[*p][1][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][4][j][k];
        pcm_sample[*p][0][j][k] = tmp_sample;
        break;

    case 4:
        tmp_sample = pcm_sample[*p][4][j][k]; /* R in T4 */
        pcm_sample[*p][0][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][3][j][k];
        pcm_sample[*p][4][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][4][j][k] - pcm_sample[*p][2][j][k];
        pcm_sample[*p][1][j][k] = tmp_sample;
        break;

    case 5:
        tmp_sample = pcm_sample[*p][3][j][k];
        pcm_sample[*p][3][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][3][j][k] - pcm_sample[*p][2][j][k];
        pcm_sample[*p][0][j][k] = tmp_sample;
        tmp_sample = pcm_sample[*p][4][j][k];
        pcm_sample[*p][4][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][4][j][k] - pcm_sample[*p][2][j][k];
        pcm_sample[*p][1][j][k] = tmp_sample;
        break;

    case 6:
        tmp_sample = pcm_sample[*p][2][j][k]; /* R in T2 */
        pcm_sample[*p][2][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][4][j][k];
        pcm_sample[*p][1][j][k] = tmp_sample;
        tmp_sample = pcm_sample[*p][3][j][k]; /* L in T3 */
        pcm_sample[*p][3][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][3][j][k] - pcm_sample[*p][2][j][k];
        pcm_sample[*p][0][j][k] = tmp_sample;
        break;

```

```

        case 7:
            tmp_sample = pcm_sample[*p][2][j][k];
            pcm_sample[*p][2][j][k] = pcm_sample[*p][0][j][k] -
pcm_sample[*p][2][j][k] - pcm_sample[*p][3][j][k];
            pcm_sample[*p][0][j][k] = tmp_sample;
            tmp_sample = pcm_sample[*p][4][j][k];
            pcm_sample[*p][4][j][k] = pcm_sample[*p][1][j][k] -
pcm_sample[*p][4][j][k] - pcm_sample[*p][2][j][k];
            pcm_sample[*p][1][j][k] = tmp_sample;
            break;

        } /* switch end loop */
    } /* for k < sblimit loop */
} /* for j < 3 loop */

/* denormalized signals */
if( fr_ps->header->dematrix_procedure != 3 ) /* dematrixing */
    for( j = 0; j < 3; ++j)
        for(k = 0; k < SBLIMIT; k++)
            { /* Lo / Ro */
                pcm_sample[*p][0][j][k] = pcm_sample[*p][0][j][k] * matr1;
                pcm_sample[*p][1][j][k] = pcm_sample[*p][1][j][k] * matr1;
                if(fr_ps->header->dematrix_procedure != 1)
                { /* surround channels */
                    pcm_sample[*p][2][j][k] = pcm_sample[*p][2][j][k] * matr2;
                    pcm_sample[*p][3][j][k] = pcm_sample[*p][3][j][k] * matr2;
                    pcm_sample[*p][4][j][k] = pcm_sample[*p][4][j][k] * matr2;
                }
                else
                {
                    pcm_sample[*p][2][j][k] = pcm_sample[*p][2][j][k] * matr2;
                    pcm_sample[*p][3][j][k] = pcm_sample[*p][3][j][k] * matr3;
                    pcm_sample[*p][4][j][k] = pcm_sample[*p][4][j][k] * matr3;
                }
            }
    }
}
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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
* *
* VERSION 2.5 *
* 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.     *
*****
*
*
*          MPEG/audio Phase 2 coding/decoding multichannel      *
*
*
* 7/27/93   Susanne Ritscher, IRT Munich                      *
*
*          thanks to                                           *
*          Ralf Schwalbe,  Telekom FTZ Berlin                 *
*          Heiko Purnhagen, Uni Hannover                      *
*
*
* 8/27/93   Susanne Ritscher, IRT Munich                      *
*          Channel-Switching is working                       *
*
* 9/1/93    Susanne Ritscher, IRT Munich                      *
*          all channels normalized                             *
*
*
* 06/16/94  Ralf Schwalbe, Telekom FTZ Berlin                 *
*          all sources and variables adapted due to MPEG-2 - *
*          DIS from March 1994                                *
*          - dematrix and denormalize procedure              *
*          - new tc - allocation (0-7)                       *
*          - some new structures and variables as a basis    *
*          for further decoding modes                         *
*
*
*          Version 1.0  Shareware                             *
*
*
* 07/12/94  Ralf Schwalbe, Telekom FTZ Berlin                 *
*          Tel: +49 30 6708 2406                             *
*          Fax: +49 30 6774 539                               *
*
*
*****/
/******/
*
* Decoder Include Files
*
*****/
/******/
*

```

## \* Decoder Definitions

\*

```

*****/

```

```

#define DFLT_OPEXT ".dec" /* default output file name extension */

```

```

#define FILTYP_DEC_AIFF "AIFF" /* '->' . 7/13/92. sr */

```

```

#define FILTYP_DEC_BNRY "TEXT" /* '->' . 7/13/92. sr */

```

```

#define CREATR_DEC_AIFF "Sd2a" /* '->' . 7/13/92. sr */

```

```

#define CREATR_DEC_BNRY "???" /* '->' . 7/13/92. sr */

```

```

#define SYNC_WORD (long) 0xffff

```

```

#define SYNC_WORD_LENGTH 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

```

```

extern void usage(void);

```

```

#else

```

```

extern void usage();

```

```

#endif

```

```

/* The following functions are in the file "decode.c" */

```

```

#ifdef PROTO_ARGS

```

```

extern void decode_info(Bit_stream_struct*, frame_params*);

```

```

extern void II_decode_bitalloc(Bit_stream_struct*, unsigned int[5][SBLIMIT],
                               frame_params*, int*, int*);

```

```

extern void I_decode_bitalloc(Bit_stream_struct*, unsigned int[2][SBLIMIT],
                               frame_params*);

```

```

extern void I_decode_scale(Bit_stream_struct*, unsigned int[2][SBLIMIT],
                           unsigned int[2][3][SBLIMIT], frame_params*);

```

```

extern void II_decode_scale(Bit_stream_struct*, unsigned int[5][SBLIMIT],
                            unsigned int[5][SBLIMIT], unsigned
                            int[5][3][SBLIMIT],

```

```

                            frame_params*, int*, int*);

```

```

extern void I_buffer_sample(Bit_stream_struct*, unsigned int[2][3][SBLIMIT],
                            unsigned int[2][SBLIMIT], frame_params*);

```

```

extern void II_buffer_sample(Bit_stream_struct*, unsigned int[5][3][SBLIMIT],
                             unsigned int[5][SBLIMIT], frame_params*);
extern void II_buffer_samplemc(Bit_stream_struct*, unsigned int[5][3][SBLIMIT],
                               unsigned int[5][SBLIMIT], frame_params*, int*);
extern void read_quantizer_table(double[17], double[17]);
extern void II_dequantize_sample(unsigned int[5][3][SBLIMIT],
                                 unsigned int[5][SBLIMIT],
                                 double[12][5][3][SBLIMIT],
                                 frame_params*, int*);
extern void II_dequantize_samplemc(unsigned int[5][3][SBLIMIT],
                                   unsigned int[5][SBLIMIT],
                                   double[12][5][3][SBLIMIT],
                                   frame_params*, int*, int*);
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[12][5][3][SBLIMIT],
                                  unsigned int[5][3][SBLIMIT], frame_params*, int, int*);
extern void II_denormalize_samplemc(double[12][5][3][SBLIMIT],
                                    unsigned int[5][3][SBLIMIT], frame_params*, int, int*,
                                    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, long*);
extern void read_syn_window(double[HAN_SIZE]);
extern void window_sample(double*, double*);
extern void out_fifo(long[5][3][SBLIMIT], int, frame_params*, int, FILE*,
                    unsigned long*);
extern void buffer_CRC(Bit_stream_struct*, unsigned int*);
extern void recover_CRC_error(long[5][3][SBLIMIT], int, frame_params*,
                              FILE*, unsigned long*);
extern void mc_header(Bit_stream_struct*, frame_params*);
extern void mc_composite_status_info(Bit_stream_struct*, frame_params*);
extern void dematricing(double[12][5][3][SBLIMIT], frame_params*,
                        int*);

#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 mc_ext_error_check();
extern void buffer_CRC();
extern void recover_CRC_error();
#endif
encode.c
/*****
Copyright (c) 1991 MPEG/audio software simulation group, 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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
* *
* VERSION 2.5 *
* 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. *
* 92-08-11 Soren H. Nielsen Fixed bug: allocation of space in the *
*      bitstream for the CRC-word. Fixed                    *
*      reading of window from file.                        *
* 92-11-06 Soren H. Nielsen Fixed scalefactor calculation.  *
*****
*
*
* MPEG/audio Phase 2 coding/decoding multichannel          *
*
* 7/27/93 Susanne Ritscher, IRT Munich                      *
* 8/10/93 changed matricing to 7 channels                   *
*      added void matricing_fft                             *
* 8/12/93 added int required_bits,                          *
*      int max_alloc                                         *
*      implemented the new mc_header (third working draft)*
* 8/13/93 added channel-switching in required_bits and     *
*      II_subband_quantisation, II_encode_scale,           *
*      II_encode_bit_alloc, II_encode_sample,              *
*      encode_info                                           *
*      all channels normalized                               *
* 9/20/93 channel-switching is only performed at a         *
*      certain limit of TC_ALLOC dB, which is included     *
*      in encoder.h                                          *
* 1/04/94 get out some rubbish                               *
*
* 01/05/94 implemented the Committee Draft header          *
*
* 01/12/94 changed matricing procedure according to        *
*      Committee Draft                                       *
*
* Version 1.0 Shareware                                     *
*
* 07/12/94 Susanne Ritscher, IRT Munich                    *
*      Tel: +49 89 32399 458                                *
*      Fax: +49 89 32399 415                                *
*****/

#include "common.h"
#include "encoder.h"

```

```

/*=====
|
| 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,
                           byte_per_sample, aiff)

FILE *musicin;
long sample_buffer[5760];
unsigned long num_samples, frame_size;
int *byte_per_sample;
int *aiff;
{
    unsigned long samples_read;
    static unsigned long samples_to_read;
    static char init = TRUE;
    short pcm_sample_buffer[5760]; /*for correct reading of pcm-data*/
    int i;

    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 ((*aiff==1) && (*byte_per_sample !=2)){

        if ((samples_read =
            fread(sample_buffer, *byte_per_sample, (int)samples_read, musicin)) == 0)
            printf("Hit end of audio data\n");

```

```

    }
    else{
        if ((samples_read =
            fread(pcm_sample_buffer, sizeof(short), (int)samples_read, musicin)) == 0)
            printf("Hit end of audio data\n");
        for(i = 0; i < 5760; ++i) sample_buffer[i] = pcm_sample_buffer[i];
    }

    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,
                       aiff_ptr, stereomc, fr_ps, aiff, byte_per_sample, buffer_matr)
FILE *musicin;
long /*far*/ buffer[5][1152];
unsigned long num_samples;
IFF_AIFF *aiff_ptr;
int stereo, stereomc;
frame_params *fr_ps;
int *aiff;
int *byte_per_sample;
double buffer_matr[7][1152];
{
    int j, i;
    long insamp[5760];
    unsigned long samples_read;
    /*int factor; */                               /*factor for matricing*/
    int lay;
    layer *info = fr_ps->header;

    lay = info->lay;

```

```

if (lay == 1){
  if(stereo == 2){ /* layer 1, stereo */
    samples_read = read_samples(musicin, insamp, num_samples,
                               (unsigned long) 768, byte_per_sample, aiff);
    for(j=0;j<448;j++) {
      if(j<64) {
        buffer[0][j] = buffer[0][j+384];
        buffer[1][j] = buffer[1][j+384];
        buffer[2][j] = 0;
        buffer[3][j] = 0;
        buffer[4][j] = 0;
      }
      else {
        buffer[0][j] = insamp[2*j-128];
        buffer[1][j] = insamp[2*j-127];
        buffer[2][j] = 0;
        buffer[3][j] = 0;
        buffer[4][j] = 0;
      }
    }
  }
  else { /* layer 1, mono */
    samples_read = read_samples(musicin, insamp, num_samples,
                               (unsigned long) 384, byte_per_sample, aiff);
    for(j=0;j<448;j++){
      if(j<64) {
        buffer[0][j] = buffer[0][j+384];
        buffer[1][j] = 0;
        buffer[2][j] = 0;
        buffer[3][j] = 0;
        buffer[4][j] = 0;
      }
      else {
        buffer[0][j] = insamp[j-64];
        buffer[1][j] = 0;
        buffer[2][j] = 0;
        buffer[3][j] = 0;
        buffer[4][j] = 0;
      }
    }
  }
}
else {
  if( *aiff == 1){
    if(((stereo + stereomc) == 5) && (aiff_ptr->numChannels == 5))
    {
      info->center = 1;
      info->surround = 2;

      samples_read = read_samples(musicin, insamp, num_samples,
                                (unsigned long) 5760, byte_per_sample, aiff);
      for(j=0; j<1152; j++)
      {
        buffer[0][j] = insamp[5*j];
        buffer[1][j] = insamp[5*j+1];
        buffer[2][j] = insamp[5*j+2];

```

```

        buffer[3][j] = insamp[5*j+3];
        buffer[4][j] = insamp[5*j+4];

#ifdef MSDOS
        buffer[0][j] = _lrotl(buffer[0][j],8);
        buffer[1][j] = _lrotl(buffer[1][j],8);
        buffer[2][j] = _lrotl(buffer[2][j],8);
        buffer[3][j] = _lrotl(buffer[3][j],8);
        buffer[4][j] = _lrotl(buffer[4][j],8);
#endif

    }

    matricing_fft(buffer, buffer_matr, fr_ps);
}

else if(( stereo == 0) && ( aiff_ptr->numChannels == 5)){

    info->center = 1;
    info->surround = 2;

    samples_read = read_samples(musicin, insamp, num_samples,
(unsigned long) 5760, byte_per_sample, aiff);
    for(j=0; j<1152; j++) {
        buffer[0][j] = 0;
        buffer[1][j] = 0;
        buffer[2][j] = insamp[5*j+2];
        buffer[3][j] = insamp[5*j+3];
        buffer[4][j] = insamp[5*j+4];

#ifdef MSDOS
        buffer[2][j] = _lrotl(buffer[2][j],8);
        buffer[3][j] = _lrotl(buffer[3][j],8);
        buffer[4][j] = _lrotl(buffer[4][j],8);
#endif
    }
/*    matricing(buffer, factor);*/
}

else if(( stereomc == 0) && ( aiff_ptr->numChannels == 5)){

    info->center = 0;
    info->surround = 0;

    samples_read = read_samples(musicin, insamp, num_samples,
(unsigned long) 5760, byte_per_sample, aiff);
    for(j=0; j<1152; j++) {
        buffer[0][j] = insamp[5*j];
        buffer[1][j] = insamp[5*j+1];
        buffer[2][j] = 0;
        buffer[3][j] = 0;
        buffer[4][j] = 0;

#ifdef MSDOS
        buffer[0][j] = _lrotl(buffer[0][j],8);
        buffer[1][j] = _lrotl(buffer[1][j],8);
#endif
    }
}

```

```

    }
    /*          matricing(buffer, factor);*/
    }

}

else { /* layerII, stereo */
    if( stereo == 2){
        samples_read = read_samples(musicin, insamp, num_samples,
            (unsigned long) 2304, byte_per_sample, aiff);
        for( j = 0; j < 1152; j++){ /* fixed bug 28.6.93 S.R. */
            buffer[0][j] = insamp[2*j];
            buffer[1][j] = insamp[2*j+1];
            buffer[2][j] = 0;
            buffer[3][j] = 0;
            buffer[4][j] = 0;
        }
    }

    else{
        /* layer 2 (or 3), mono */
        samples_read = read_samples(musicin, insamp, num_samples,
            (unsigned long) 1152, byte_per_sample, aiff);
        for(j=0;j<1152;j++){
            buffer[0][j] = insamp[j];
            buffer[1][j] = 0;
            buffer[2][j] = 0;
            buffer[3][j] = 0;
            buffer[4][j] = 0;
        }
    }
}
}
return(samples_read);
}

/******
/*
/* matricing()
/*
/* The five-channel signal must be matricied to guarantee
/* the compatibility to the stereo-decoder.
/* There must be something of the surround information in the
/* two front channels. In a five-channel decoder there will
/* be a dematricing.
/* There must be 7 channels for channel switching 8/10/93, SR
/*
/* Channel 5 and 6 are the not matriced signals L and R
/*
/******

void matricing(sb_sample, fr_ps)
double sb_sample[7][3][12][SBLIMIT];
frame_params *fr_ps;

{

```

```

double matr1; /* factor for center */
double matr2; /* factor for surrounds */
double matr3; /* factor for surrounds */
int i, j, k, l;

layer *info = fr_ps->header;
switch(info->matrix)
{
    /* Changed the factors according to Draft International Standard */

    case 0: matr1 = 0.411865234375; /* normalizing */
            matr2 = 0.7060546875; /* matricing */
            break;
    case 1: matr1 = 1/(1.5 + 0.5*sqrt(2)); /* normalizing */
            matr2 = 0.7060546875; /* matricing C */
            matr3 = 0.5; /* matricing Ls, Rs */
            break;
    case 2: matr1 = 0.411865234375;
            matr2 = 0.7060546875;
            printf("NOT DONE YET!!!\n");
            exit(0);
            break;
    case 3: matr1 = 1.0;
            matr2 = 1.0;
            break;
}

for( j = 0; j < 3; ++j)
{
    for(l = 0; l < 12; l++)
    {
        for(k = 0; k < SBLIMIT; k++)
        {
            sb_sample[5][j][l][k] = sb_sample[0][j][l][k] * matr1;
            sb_sample[6][j][l][k] = sb_sample[1][j][l][k] * matr1;

            sb_sample[2][j][l][k] = sb_sample[2][j][l][k] * matr1 * matr2;

            if(info->matrix != 1)
            {
                sb_sample[3][j][l][k] = sb_sample[3][j][l][k] * matr1 * matr2;
                sb_sample[4][j][l][k] = sb_sample[4][j][l][k] * matr1 * matr2;
            }
            else
            {
                sb_sample[3][j][l][k] = sb_sample[3][j][l][k] * matr1 * matr3;
                sb_sample[4][j][l][k] = sb_sample[4][j][l][k] * matr1 * matr3;
            }

            sb_sample[0][j][l][k] = sb_sample[5][j][l][k] + sb_sample[2][j][l][k] +
sb_sample[3][j][l][k];
            sb_sample[1][j][l][k] = sb_sample[6][j][l][k] + sb_sample[2][j][l][k] +
sb_sample[4][j][l][k];
        }
    }
}

```

```

}

/******
/*
/* matricing_fft()
/*
/* To get the best results in psychoacoustics there must be both,
/* the matriced and the not matriced signal. This matricing
/* may be in full bandwidth.
/* 8/10/93 SR
/* Channel 5 and 6 are the not matriced signals L and R
/*
/*
/******

void matricing_fft(buffer, buffer_matr, fr_ps)
long buffer[5][1152];
double buffer_matr[7][1152];
frame_params *fr_ps;

{
double matr1; /* factor for normalizing */
double matr2; /* factor for matricing */
double matr3; /* factor for matricing */
int i, j, k, l;
double dummy;

layer *info = fr_ps->header;
switch(info->matrix)
{
/* factors according to Draft International Standard */
case 0: matr1 = 0.411865234375; /* normalizing factor */
matr2 = 0.7060546875; /* matricing factor */
break;
case 1: matr1 = 1/(1 + sqrt(2)); /* normalizing factor */
matr2 = 0.7060546875; /* matricing C */
matr3 = 0.5; /* matricing Ls, Rs */
break;
case 2: matr1 = 0.411865234375; /* normalizing factor */
matr2 = 0.7060546875; /* matricing factor */
printf(" NOT DONE YET !\n");
exit(0);
break;
case 3: matr1 = 1.0;
matr2 = matr1;
break;
}

/* no normalizing until now!!! 01/14/94, SR */
for(i = 0; i < 1152; ++i)
{
buffer_matr[5][i] = buffer[0][i];
buffer_matr[6][i] = buffer[1][i];
buffer_matr[2][i] = buffer[2][i];
buffer_matr[3][i] = buffer[3][i];
buffer_matr[4][i] = buffer[4][i];

if(info->matrix != 3)

```



```

{
  if(info->matrix != 1)
  {
    buffer_matr[0][i] = buffer[0][i] + matr2*buffer[2][i] + matr2*buffer[3][i];
    buffer_matr[1][i] = buffer[1][i] + matr2*buffer[2][i] + matr2*buffer[4][i];
  }
  else
  {
    buffer_matr[0][i] = buffer[0][i] + matr2*buffer[2][i] + matr3*buffer[3][i];
    buffer_matr[1][i] = buffer[1][i] + matr2*buffer[2][i] + matr3*buffer[4][i];
  }
}
else
{
  buffer_matr[0][i] = buffer[0][i];
  buffer_matr[1][i] = buffer[1][i];
}
}
}

```

```

/*****
/*
/* 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, 80, fp); /* changed from 150, 92-08-11 shn */
    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,80,fp); /* changed from 150, 92-08-11 shn */
}
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)
long **buffer;
double z[HAN_SIZE];
int k;
{
typedef double XX[5][HAN_SIZE];
static XX *x;
int i, j;
static off[5] = {0, 0, 0, 0, 0};
static char init = 0;
static double *c;

if (!init) {
    c = (double *) mem_alloc(sizeof(double) * HAN_SIZE, "window");
    read_ana_window(c);
    x = (XX *) mem_alloc(sizeof(XX), "x");
    for (i = 0; i < 5; i++)
        for (j = 0; j < HAN_SIZE; j++)
            (*x)[i][j] = 0;
    init = 1;
}

for (i=0; i<32; i++) (*x)[k][31-i+off[k]] = (double) *(*buffer)++/SCALE;
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, k;
    static char init = 0;
    typedef double MM[SBLIMIT][64];
    static MM /*far*/ *m;
    double sum1, sum2;

#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
    /* Window */
    for (i=0; i<64; i++)
    {
        for (k=0, sum1 = 0.0; k<8; k++)
            sum1 += z[i+64*k];
        y[i] = sum1;
    }

    /* Filter */
    for (i=0; i<SBLIMIT; i++)
    {
        for (k=0, sum1=0.0 ;k<64;k++)
            sum1 += (*m)[i][k] * y[k];
        s[i] = sum1;
    }

    /* 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()
/* encode_infomc1() SR
/* encode_infomc2() SR
/*
/* PURPOSE: Puts the syncword and header information on the output
/* bitstream.
/*
*****/

void encode_info(fr_ps,bs)
frame_params *fr_ps;
Bit_stream_struct *bs;
{
    layer *info = fr_ps->header;

    putbits(bs,0xffff,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);
}

```

```

void encode_infomc1(fr_ps,bs)
frame_params *fr_ps;
Bit_stream_struct *bs;
{
    layer *info = fr_ps->header;

    putbits(bs,info->center, 2);
    putbits(bs,info->surround, 2);
    put1bit(bs,info->lfe);
    put1bit(bs,info->audio_mix);
    putbits(bs,info->matrix, 2);
    putbits(bs,info->multiling_ch, 3);
    put1bit(bs,info->multiling_fs);
    put1bit(bs,info->multiling_lay);
    put1bit(bs,info->ext_bit_stream_present);
}

void encode_infomc2(fr_ps, bs, npredcoef)
frame_params *fr_ps;
Bit_stream_struct *bs;
int npredcoef[8];
{
    layer *info = fr_ps->header;
    int i, j;

    put1bit(bs, info->tc_sbgr_select);
    put1bit(bs, info->dyn_cross_on);
    put1bit(bs, info->mc_prediction_on);
    if(info->tc_sbgr_select == 1)
        putbits(bs, info->tc_allocation, 3);
    else
    {
        for(i = 0; i < 12; i++)
            putbits(bs, info->tc_alloc[i], 3);
    }

    if(info->dyn_cross_on == 1)
    {
        fprintf(stderr, "Not done yet Dynamic Crosstalk!!!\n");
        exit(0);
    }

    if(info->mc_prediction_on == 1)
    {
        fprintf(stderr, "Not done yet Prediction!!!\n");
        exit(0);
    }
}

```

```

/*****
/*
/* 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[7][3][SCALE_BLOCK][SBLIMIT];
double /*far*/ joint_sample[2][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][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[7][3][SCALE_BLOCK][SBLIMIT];
double /*far*/ joint_sample[2][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[0][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[7][3][SCALE_BLOCK][SBLIMIT];
unsigned int scalar[7][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-1,scalar[k][0][i]=0;j>=0;j--)
                if (s[i] < multiple[j]) { /* <= changed to <, 1992-11-06 shn */
                    scalar[k][0][i] = j;
                    break;
                }
    }
}

/***** Layer II *****/

void II_scale_factor_calc(sb_sample,scalar,sblimit, l, m)
double /*far*/ sb_sample[7][3][SCALE_BLOCK][SBLIMIT];
unsigned int scalar[7][3][SBLIMIT];
int sblimit;
int l, m;
{
    int i,j, k,t;
    double s[SBLIMIT];

    for (k = l; k < m; 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-1,scalar[k][t][i]=0;j>=0;j--)
            if (s[i] < multiple[j])
                {
                    /* <= changed to <, 1992-11-06 shn*/
                    scalar[k][t][i] = j;
                    break;
                }
        for (i=sblimit;i<SBLIMIT;i++) scalar[k][t][i] = SCALE_RANGE-1;
    }
}

/*****
/* void II_scale_factor_calc1(sb_sample, scalar, stereo, sblimit)
/*
/* in case of any joint stereo the scalefactor must be computed
/* a second time for the combind samples
/*
*****/

void II_scale_factor_calc1(sb_sample,scalar, sblimit, dim)
double /*far*/ sb_sample[2][3][SCALE_BLOCK][SBLIMIT];
unsigned int scalar[2][3][SBLIMIT];
int sblimit;
int dim;
{
    int i,j, k,t;
    double s[SBLIMIT];

    for (t=0;t<3;t++) {
        for (i=0;i<sblimit;i++)
            for (j=1, s[i] = mod(sb_sample[dim][t][0][i]);j<SCALE_BLOCK;j++)
                if (mod(sb_sample[dim][t][j][i]) > s[i])
                    s[i] = mod(sb_sample[dim][t][j][i]);

        for (i=0;i<sblimit;i++)
            for (j=SCALE_RANGE-1,scalar[dim][t][i]=0;j>=0;j--)
                if (s[i] < multiple[j]) { /* <= changed to <, 1992-11-06 shn */
                    scalar[dim][t][i] = j;
                    break;
                }
            for (i=sblimit;i<SBLIMIT;i++) scalar[dim][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, cha_sw, aiff)
unsigned int scalar[7][3][SBLIMIT];
frame_params *fr_ps;
double /*far*/ max_sc[7][SBLIMIT];
int cha_sw;
int aiff;
{
    int i,j,k,l,m;
    int max;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;

    if(aiff != 1)
    {
        l = 0; m = 2;
    }
    else
    {
        l = 0;
        m = 7;
    }

    for (k = l; k < m; 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;

    if(aiff == 1)
    {
        if((fr_ps->header->matrix == 3) || (cha_sw == 0))
            fr_ps->header->tc_sbgr_select = 1;
        else tc_alloc(fr_ps, max_sc);
    }
}

/*****
/*
/* tc_alloc (Layer II, multichannel)
/*
/* PURPOSE: For each subbandgroup the three transmissionchannels are
/*           determined by taking the channel with the lowest level
/*           according to the tabel tc_allocation in the draft
/* 8/10/93, SR
/*
/*           changed to a certain limit of TC_ALLOC which must be stepped
/*           beyond, before there is channel-switching
/* 9/20/93 SR
*****/

void tc_alloc(fr_ps, max_sc)
frame_params *fr_ps;
double max_sc[7][SBLIMIT];

```

```

{
    int i, l, k;
    int min;
    double min1;
    double min2[7][12];

    for(i = 0; i < 8; i++)
    {
        if(((20 * log10(max_sc[2][i])) - (20 * log10(max_sc[5][i]))) > TC_ALLOC)
        {
            if(max_sc[6][i] < max_sc[5][i])
            {
                min = 6;
            }
            else if(max_sc[6][i] == max_sc[5][i])
            {
                if(max_sc[3][i] <= max_sc[5][i])
                {
                    min = 5;
                }
                else min = 6;
            }
            else min = 5;
        }
        else if(((20 * log10(max_sc[2][i])) - (20 * log10(max_sc[6][i]))) > TC_ALLOC)
        {
            if(max_sc[5][i] < max_sc[6][i])
            {
                min = 5;
            }
            else if(max_sc[6][i] == max_sc[5][i])
            {
                if(max_sc[3][i] <= max_sc[5][i])
                {
                    min = 5;
                }
                else min = 6;
            }
            else min = 6;
        }
        else
        {
            min = 2;
        }

        switch(min)
        {
            case 5: if(max_sc[4][i] <= max_sc[6][i]) /* left front, Rs*/
                    fr_ps->header->tc_alloc[i] = 1;
                    else fr_ps->header->tc_alloc[i] = 7; /*R*/
                    break;
            case 6: if(max_sc[3][i] <= max_sc[5][i]) /* right front, Ls*/
                    fr_ps->header->tc_alloc[i] = 2;
                    else fr_ps->header->tc_alloc[i] = 6; /* L */
                    break;
            case 2: if(((20 * log10(max_sc[3][i])) - (20 * log10(max_sc[5][i]))) > TC_ALLOC)
                    {
                        if(max_sc[4][i] <= max_sc[6][i])
                        {
                            fr_ps->header->tc_alloc[i] = 3;
                        }
                        else fr_ps->header->tc_alloc[i] = 5;
                    }
                    else
                    {
                        if(((20 * log10(max_sc[4][i])) - (20 * log10(max_sc[6][i]))) > TC_ALLOC)
                        {
                            fr_ps->header->tc_alloc[i] = 4;
                        }
                    }
                }
            }
        }
    }
}

```

```

        else fr_ps->header->tc_alloc[i] = 0;
    }
    break;
}
}

```

```

for(i = 8; i < 12; i++)
    for(k = 2; k < 7; k++)
        min2[k][i] = 0.0;

```

```

for(i = 8; i < 12; i++) /*taking the average scalefactor of each sb-group*/
{
    for(k = 2; k < 7; k++)
    {
        for(l = (sb_groups[i-1] + 1); l <= sb_groups[i]; l++)
        {
            min2[k][i] += max_sc[k][l];
        }
        min2[k][i] = min2[k][i] / (sb_groups[i] - sb_groups[i-1]);
    }
}

```

```

if(((20 * log10(min2[2][i])) - (20 * log10(min2[5][i]))) > TC_ALLOC)
{
    if(min2[6][i] < min2[5][i])
        min = 6;
    else if(min2[6][i] == min2[5][i])
    {
        if(min2[3][i] <= min2[5][i])
            min = 5;
        else min = 6;
    }
    else min = 5;
}
else if(((20 * log10(min2[2][i])) - (20 * log10(min2[6][i]))) > TC_ALLOC)
{
    if(min2[5][i] < min2[6][i])
        min = 5;
    else if(min2[6][i] == min2[5][i])
    {
        if(min2[3][i] <= min2[5][i])
            min = 5;
        else min = 6;
    }
    else min = 6;
}
else
{
    min = 2;
}

```

```

switch(min)
{
    case 5: if(min2[4][i] <= min2[6][i]) /* left front, Rs*/
        fr_ps->header->tc_alloc[i] = 1;
        else fr_ps->header->tc_alloc[i] = 7; /* R*/
        break;
    case 6: if(min2[3][i] <= min2[5][i]) /* right front, Ls*/
        fr_ps->header->tc_alloc[i] = 2;
        else fr_ps->header->tc_alloc[i] = 6; /* L */
        break;
    case 2: if(((20 * log10(min2[3][i])) - (20 * log10(min2[5][i])))) > TC_ALLOC)
        {
            if(min2[4][i] <= min2[6][i])
                fr_ps->header->tc_alloc[i] = 3;
            else fr_ps->header->tc_alloc[i] = 5;
        }
        else
        {
            if(((20 * log10(min2[4][i])) - (20 * log10(min2[6][i])))) > TC_ALLOC)
                fr_ps->header->tc_alloc[i] = 4;
            else fr_ps->header->tc_alloc[i] = 0;
        }
        break;
    }
}

```

```

if(fr_ps->header->tc_alloc[0] == fr_ps->header->tc_alloc[1] &&
   fr_ps->header->tc_alloc[1] == fr_ps->header->tc_alloc[2] &&
   fr_ps->header->tc_alloc[2] == fr_ps->header->tc_alloc[3] &&
   fr_ps->header->tc_alloc[3] == fr_ps->header->tc_alloc[4] &&
   fr_ps->header->tc_alloc[4] == fr_ps->header->tc_alloc[5] &&
   fr_ps->header->tc_alloc[5] == fr_ps->header->tc_alloc[6] &&
   fr_ps->header->tc_alloc[6] == fr_ps->header->tc_alloc[7] &&
   fr_ps->header->tc_alloc[7] == fr_ps->header->tc_alloc[8] &&
   fr_ps->header->tc_alloc[8] == fr_ps->header->tc_alloc[9] &&
   fr_ps->header->tc_alloc[9] == fr_ps->header->tc_alloc[10] &&
   fr_ps->header->tc_alloc[10] == fr_ps->header->tc_alloc[11])
{
    fr_ps->header->tc_sbgr_select = 1;
    fr_ps->header->tc_allocation = fr_ps->header->tc_alloc[0];
}
else fr_ps->header->tc_sbgr_select = 0; /* added 8/20/93,SR*/
}

```

```

/*****

```

```

/*
/* 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[7][3][SBLIMIT];
frame_params *fr_ps;
double /*far*/ max_sc[7][SBLIMIT];
{
    int i,j,k, max;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;

    for (k = 0; k < stereo+stereomc+2; 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,scfsi_dyn)
unsigned int scalar[7][3][SBLIMIT];
unsigned int scfsi[7][SBLIMIT];
frame_params *fr_ps;
unsigned int scfsi_dyn[7][SBLIMIT];

{
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    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+stereomc+2; 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];

    } /* switch */
  } /* subband */

}

/******
/*
/* 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[7][3][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *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, l, z, scfsi_dyn)
unsigned int bit_alloc[7][SBLIMIT], scfsi[7][SBLIMIT];
unsigned int scalar[7][3][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *bs;
int *l;
int *z;
unsigned int scfsi_dyn[7][SBLIMIT];
{
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int i,j,k, m, n;

    for (i=0;i<sblimit;i++)
        for (m = *l; m < *z; m++)
        {
            if(fr_ps->header->tc_sbgr_select == 1)
                k = transmission_channel[fr_ps->header->tc_allocation][m];
            else
            {
                if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
                else
                {
                    for(n = 1; n < 12; n++)
                    {
                        if((sb_groups[n-1] < i) && (i <= sb_groups[n]))
                        {
                            k = transmission_channel[fr_ps->header->tc_alloc[n]][m];
                            break;
                        }
                    }
                }
            }

            if (bit_alloc[k][i]) putbits(bs,scfsi[k][i],2);
        }
}

```

```

for (i=0;i<sblimit;i++)
for (m = *l;m < *z; m++)
{
    if(fr_ps->header->tc_sbgr_select == 1)
    k = transmission_channel[fr_ps->header->tc_allocation][m];
    else
    {
        if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
        else
        {
            for(n = 1; n < 12; n++)
            {
                if((sb_groups[n-1] < i) && (i <= sb_groups[n])) /*bug!! 8/21/93,SR*/
                {
                    k = transmission_channel[fr_ps->header->tc_alloc[n]][m];
                    break;
                }
            }
        }
    }
}

if (bit_alloc[k][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};*/
static double snr[18] = { 0.00, 6.03, 11.80, 15.81, /* 0, 3, 5, 7 */
                        19.03, 23.50, 29.82, 35.99, /* 9,15,31,63 */
                        42.08, 48.13, 54.17, 60.20, /* 127, ... */
                        66.22, 72.25, 78.27, 84.29, /* 2047, ... */
                        90.31, 96.33}; /* 16383, ... */

int I_bits_for_nonoise(perm_smr, fr_ps)
double /*far*/ perm_smr[7][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]) >= fr_ps->mnr_min)
                    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]) >= fr_ps->mnr_min) break;
            if(k>0) req_bits += (k+1)*12 + 6*((i>=jsbound)?stereo:1);
        }
    return req_bits;
}

```

```

}

/***** Layer II *****/

int II_bits_for_nonoise(perm_smr, scfsi, fr_ps, a, b, aiff)
double /*far*/ perm_smr[7][SBLIMIT];
unsigned int scfsi[7][SBLIMIT];
frame_params *fr_ps;
int a;
int b;
int *aiff;
{
    int sb,ch,ba,i;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    al_table *alloc = fr_ps->alloc;
    int bbal = 0;
    int berr;
    int banc = 32;
    int bancmc = 0;
    int maxAlloc, sel_bits, sc_bits, smp_bits;
    int req_bits = 0;
    static int sfsPerScfsi[] = { 3,2,1,2 }; /* lookup # sfs per scfsi */

    if( *aiff == 1)
    {
        bancmc += 31; /* mc_header + crc + tc_sbgr_select+ dyn_cross_on +
                     mc_prediction_on 01/05/94, SR*/
        if(fr_ps->header->tc_sbgr_select == 0)
            bancmc += 36;
        else
            bancmc += 3;

        if(fr_ps->header->dyn_cross_on == 1)
        {
            fprintf(stderr, "not done yet Dynamic Crosstalk!!!\n");
            exit(0);
        }

        if(fr_ps->header->mc_prediction_on == 1)
        {
            fprintf(stderr, "not done yet Prediction!!!\n");
            exit(0);
        }
    }
    else bancmc = 0;
    if (fr_ps->header->error_protection) berr=16; else berr=0; /* added 92-08-11 shn */
    /*fixed bug (no errormc) 28.6.93 SR */
    for (sb = 0; sb < jsbound; ++sb)
        bbal += (stereo+stereomc) * (*alloc)[sb][0].bits;

    for (sb = jsbound; sb < sblimit; ++sb)
        bbal += (stereo-1+stereomc) * (*alloc)[sb][0].bits;

    req_bits = banc + bancmc + bbal + berr;

```

```

    for(sb = 0; sb < sblimit; ++sb)
    {
        for(ch = 0; ch < ((sb < jsbound)? 5 : 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)])
                >= fr_ps->mnr_min)
                break;
            if(((b - a) >= 1) && (sb >= jsbound))
            { /* check other JS channels */
                for(;ba<maxAlloc-1; ++ba)
                    if( (-perm_smr[1-ch ][sb] + snr[(*alloc)[sb][ba].quant+((ba>0)?1:0)])
                        >= fr_ps->mnr_min)
                        break;
            }
        }

        if(ba>0)
        {
            smp_bits = 12 * ((*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;
}

```

```

/*****
/*now for the independent channels, 8/6/93, SR          */
*****/
int II_bits_for_indi(perm_smr, scfsi, fr_ps, a, b, aiff)
double /*far*/ perm_smr[7][SBLIMIT];
unsigned int scfsi[7][SBLIMIT];
frame_params *fr_ps;
int *a;
int *b;
int *aiff;
{

```

```

int sb,ch,ba,i;
int stereo = fr_ps->stereo;
int stereomc = fr_ps->stereomc;
int sblimit = fr_ps->sblimit;
int jsbound = fr_ps->jsbound;
al_table *alloc = fr_ps->alloc;
int req_bits = 0, bbal = 0;
int berr; /* before: =0 92-08-11 shn */
int banc = 32; /* header ISO Layer II */
int bancmc; /* header multichannel = 93, 5.7.93,SR*/
int maxAlloc, sel_bits, sc_bits, smp_bits;
static int sfsPerScfsi[] = { 3,2,1,2 }; /* lookup # sfs per scfsi */

    for(sb = jsbound; sb < sblimit; ++sb)
    {
        for(ch = 2; ch < 5; ++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)])
                >= fr_ps->mnr_min)
                break; /* we found enough bits */

            if(ba>0)
            {
                smp_bits = 12 * ((*alloc)[sb][ba].group * (*alloc)[sb][ba].bits);
                /* scale factor bits required for subband */
                sel_bits = 2;
                sc_bits = 6 * sfsPerScfsi[scfsi[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 (fr_ps->mnr_min). Otherwise, the system
/* iteratively allocates less bits by using joint stereo until one
/* of the following occurs:

```

```

/* - there are no more noisy subbands (MNR >= fr_ps->mnr_min)
/* - 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[7][SBLIMIT];
unsigned int bit_alloc[7][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, ltmin, scfsi, bit_alloc, adb,
                           fr_ps, aiff, sb_sample, scalar, max_sc,
                           cha_sw, buffer, spiki,
                           joint_sample, j_scale, scfsi_dyn, frameNum)
double perm_smr[7][SBLIMIT]; /* minimum masking level */
double ltmin[7][SBLIMIT]; /* minimum masking level */
unsigned int scfsi[7][SBLIMIT];

```

```

unsigned int bit_alloc[7][SBLIMIT];
int *adb;
frame_params *fr_ps;
int *aiff;
double sb_sample[7][3][12][SBLIMIT];
unsigned int scalar[7][3][SBLIMIT];
double max_sc[7][SBLIMIT];
int cha_sw;
double buffer[7][1152];
double spiki[7][SBLIMIT];
double joint_sample[2][3][12][SBLIMIT];
unsigned int j_scale[2][3][SBLIMIT];
unsigned int scfsi_dyn[7][SBLIMIT];
int frameNum;
{
    int noisy_sbs, nn;
    int mode, mode_ext, lay, modemc_hlp;
    int rq_db, av_db = *adb;
    int a, b, i, l, m, sb, k;
    int sbbound = -30;
    float adb_help;
    double smr_pred[7][SBLIMIT];
    float preco[12][4];
    float prega[3][32];
    int bits1, bits2;

    /*****layer II two channels *****/
    if(*aiff == 0){
        if((mode = fr_ps->actual_mode) == MPG_MD_JOINT_STEREO){
            a = 0;
            b = 1;
            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, &a, &b, aiff)
                > *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, &a, &b, aiff);
                } while( (rq_db > *adb) && (mode_ext > 0));
                fr_ps->header->mode_ext = mode_ext;
            } /* well we either eliminated noisy sbs or mode_ext == 0 */
        }
    }

    /***** layer II five channels *****/
    else{

        if((mode = fr_ps->actual_mode) == MPG_MD_JOINT_STEREO)

```

```

{
    a = 0;
    b = 1;
    fr_ps->header->mode = MPG_MD_STEREO;
    fr_ps->header->mode_ext = 0;
    fr_ps->jsbound = fr_ps->sblimit;
    adb_help = *adb;
    if((rq_db = II_bits_for_nonoise(perm_smr, scfsi, fr_ps, a, b, aiff)
        + II_bits_for_indi(perm_smr, scfsi, fr_ps, &a, &b, aiff))
        > adb_help)
    {
        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, a, b, aiff)
                + II_bits_for_indi(perm_smr, scfsi, fr_ps, &a, &b, aiff);
        } while( (rq_db > *adb) && (mode_ext > 0));
        fr_ps->header->mode_ext = mode_ext;
    } /* well we either eliminated noisy sbs or mode_ext == 0 */

}

} /* end of else (five-channel) */

noisy_sbs = II_a_bit_allocation(perm_smr, scfsi, bit_alloc, adb,
                                fr_ps, aiff);

}

/******
/*
/* 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[7][SBLIMIT];
unsigned int bit_alloc[7][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;
    double mnrl[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 bbal, banc, berr;

    if (!init) {
        banc = 32;    /* before: berr = 0; 92-08-11 shn */
        init = 1;

        if (fr_ps->header->error_protection) berr = 16; else berr= 0; /* 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++) {
        mnrl[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 = mnrl[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 > mnrl[k][i]) {
                small = mnrl[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 = 12; 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 */
            }
        }
    } while (bspl + bscf + scale_bits + smpl_bits < ad);
}

```



```

        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] < fr_ps->mnr_min) ++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, aiff)
double /*far*/ perm_smr[7][SBLIMIT]; /* minimum masking level */
unsigned int scfsi[7][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
int *adb;
frame_params *fr_ps;
int *aiff; /* flag for masking level */
{
    int i, min_ch, min_sb, oth_ch, k, increment, scale, seli, ba, j, l;
    int adb_hlp, adb_hlp1, adb_hlp2;
    int bspl, bscf, bscl, ad, noisy_sbs;
    double mnr[7][SBLIMIT], small;
    char used[7][SBLIMIT];
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    int jsboundmc = fr_ps->jsboundmc;
    al_table *alloc = fr_ps->alloc;

```

```

        double dynsmr = 0.0; /* border of SMR for dynamic datarate */
        static char init= 0;
        static int banc, berr;
        int bbal, bancmc;
        int ll;
        static int sfsPerScfsi[] = { 3,2,1,2 }; /* lookup # sfs per scfsi */
        if (!init)
        {
            init = 1; banc = 32; /* banc: bits for header */;
            if (fr_ps->header->error_protection) berr=16; else berr=0; /* added 92-08-11 shn */
        }

        bancmc = 0;
        bbal = 0;
        if( *aiff == 1)
        {
            if(fr_ps->actual_mode == MPG_MD_JOINT_STEREO)
            {
                /*JS Lo + Ro */
                for(i = 0; i < jsbound; ++i)
                    bbal += 5 * (*alloc)[i][0].bits;
                for(i = jsbound; i < sblimit; ++i)
                    bbal += 4 * (*alloc)[i][0].bits;
            }
            else
            {
                for( i = 0; i < sblimit; ++i)
                    bbal += 5 * (*alloc)[i][0].bits;
                if(fr_ps->header->center == 3) bbal -= 41;
            }
        }
        else
        {
            for(i = 0; i < jsbound; ++i)
                bbal += stereo * (*alloc)[i][0].bits;
            for(i = jsbound; i < sblimit; ++i)
                bbal += (*alloc)[i][0].bits;
        }

        if(fr_ps->header->ext_bit_stream_present == 0)
            bancmc += 33; /* mc_header + crc + tc_sbgr_select+ dyn_cross_on +
                           mc_prediction_on 01/05/94, SR new! 05/04/94, SR*/
        else
        {
            fprintf(stderr, "not done yet Extension Bitstream!!!\n");
            exit(0);
        }

        if(fr_ps->header->tc_sbgr_select == 0)
            bancmc += 36;
        else
            bancmc += 3;

        if(fr_ps->header->mc_prediction_on == 1)
        {

```

```

        fprintf(stderr, "not done yet Prediction!!!\n");
        exit(0);
    }

    for (i = 0; i < sblimit; i++)
        for (k = 0; k < (stereo+stereomc+2); k++)
        {
            mnrc[k][i]=snr[0]-perm_smr[k][i];
            /* mask-to-noise-level = signal-to-noise-level - minimum-masking-*/
            /* threshold*/

            bit_alloc[k][i] = 0;
            used[k][i] = 0;
        }

    if(*aiff != 1)
    {
        *adb -= bbal + berr + banc;
    }
    else *adb -= bbal + berr + banc + bancmc;

    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++) /* searching for the sb min SMR*/
            for(j = 0; j < (stereo+stereomc); ++j)
            {
                if(fr_ps->header->tc_sbgr_select == 1)
                {
                    k = transmission_channel[fr_ps->header->tc_allocation][j];
                }
                else
                {
                    {
                        if(i == 0)
                        {
                            k = transmission_channel[fr_ps->header->tc_alloc[0]][j];
                            l = 0;
                        }
                        else
                        {
                            {
                                for(l = 1; l < 12; l++)
                                {
                                    if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
                                    {
                                        k = transmission_channel[fr_ps->header->tc_alloc[l]][j];
                                        break;
                                    }
                                }
                            }
                        }
                    }
                }
            }
    }

```

```

    }

    if ((used[k][i] != 2) && (small > mnr[k][i]))
    {
        small = mnr[k][i];
        min_sb = i; min_ch = k;
        ll = 1;          /*sb-group*/
    }
}

if(min_sb > -1)
{ /* there was something to find */
    /* find increase in bit allocation in subband [min] */
    increment = 12 * ((*alloc)[min_sb][bit_alloc[min_ch][min_sb]+1].group *
        (*alloc)[min_sb][bit_alloc[min_ch][min_sb]+1].bits);
    /* how many bits are needed */

    if (used[min_ch][min_sb])
        increment -= 12 * ((*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] */
    /* above js bound, need both chans */

    if((fr_ps->actual_mode == MPG_MD_JOINT_STEREO) &&
        ((min_ch == 0) || (min_ch == 1)))
        oth_ch = 1 - min_ch;

    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((fr_ps->actual_mode == MPG_MD_JOINT_STEREO) && (min_sb >=
jsbound) &&
            ((min_ch == 0) || (min_ch == 1)))
        {
            /* 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 (*adb > 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 */

        /*****!!! here make a difference between the different JS-modes !!!!!***/
        if((fr_ps->actual_mode == MPG_MD_JOINT_STEREO) && (min_sb >= jsbound) &&
        (stereo == 2) &&
            ((min_ch == 0) || (min_ch == 1)))
        {
            /* 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];
            mn_r[oth_ch][min_sb] = -perm_smr[oth_ch][min_sb] +
                snr[(*alloc)[min_sb][ba].quant+1];
        }
        } /* end of if-loop if min_sb > -1 */
    } while(min_sb > -1); /* until could find no channel */
    /* Calculate the number of bits left */
    /* fprintf(stderr,"bsamp = %d\t",bspl); */
    /* fprintf(stderr,"bscf = %d\t",bscf); */
    /* fprintf(stderr,"bscf si = %d\n",bsel); */
    /* fflush(stderr); */

    *adb -= bspl+bscf+bsel;
    for (i=sblimit;i<SBLIMIT;i++) for (k = 0; k < 7; k++) /*for all 7 channels, 8/21/93,SR*/
        bit_alloc[k][i]=0;

    /* NUR FUER HEUTE DYN DATENRATE!!! 5.5.94, SR*/
    /* *adb = 0;*/

    /*not used !?! perhaps later!! 8/21/93, SR*/
    noisy_sbs = 0; small = mn_r[0][0]; /* calc worst noise in case */
    for(k = 0; k < (stereo+stereomc); ++k)
    {
        for (i=0;i<sblimit;i++)
        {
            if (small > mn_r[k][i]) small = mn_r[k][i];
            if(mn_r[k][i] < fr_ps->mn_r_min) ++noisy_sbs; /* noise is not masked */
        }
    }

    return noisy_sbs;
}

/*****
/*
/* I_subband_quantization (Layer I)
/* II_subband_quantization (Layer II)
/* II_subband_quantisationmc (MPEG2) SR
/* 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[75][3][SBLIMIT];
double /*far*/ sb_samples[7][3][12][SBLIMIT];
unsigned int j_scale[2][3][SBLIMIT];
double /*far*/ j_samps[2][3][12][SBLIMIT]; /* L+R for j-stereo if necess */
unsigned int bit_alloc[7][SBLIMIT];
unsigned int /*far*/ sbband[7][3][12][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<12;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][0][j][i] / multiple[j_scale[0][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[7][3][SBLIMIT];
double /*far*/ sb_samples[7][3][12][SBLIMIT];
unsigned int j_scale[2][3][SBLIMIT];
double /*far*/ j_samps[2][3][12][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
unsigned int /*far*/ sbband[7][3][12][SBLIMIT];
frame_params *fr_ps;
{
    int i, j, k, s, n, qnt, sig, l, m;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    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<12;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[0][s][j][i] / multiple[j_scale[0][s][i]];
                        else
                            d = sb_samples[k][s][j][i] / multiple[scalar[k][s][i]];
                    }
                }
}

```

```

        if (mod(d) >= 1.0) /* > changed to >=, 1992-11-06 shn */
        {
            printf("Not scaled properly, %d %d %d %d\n",k,s,j,i);
            printf("Value %1.10f\n",sb_samples[k][s][j][i]);
            printf("Channel %d\n", k);
        }
        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;
#ifdef 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<12;i++)
            for (m=0;m<stereo;m++)
                sbband[m][s][i][j] = 0;
}

```

```

void II_subband_quantizationmc(scalar, sb_samples, j_scale, j_samps,
                             bit_alloc, sbband, fr_ps)

```

```

unsigned int scalar[7][3][SBLIMIT];
double /*far*/ sb_samples[7][3][12][SBLIMIT];
unsigned int j_scale[2][3][SBLIMIT];
double /*far*/ j_samps[2][3][12][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
unsigned int /*far*/ sbband[7][3][12][SBLIMIT];
frame_params *fr_ps;
{
    int i, j, k, s, n, qnt, sig, m, l, ll;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    unsigned int stps;
    double d;
    al_table *alloc = fr_ps->alloc;

```



```

for(i = 1; i < SBLIMIT; i++)
{
    for(l = 1; l < 12; l++)
    {
        if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
            break;
    }
}

for (s=0;s<3;s++)
for (j=0;j<12;j++)
for (i=0;i<sblimit;i++)
{
    for(l = 1; l < 12; l++)
    {
        if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
        {
            ll = l;
            break;
        }
    }
}

for (m = 2; m < 5; ++m)
{
    if(fr_ps->header->tc_sbgr_select == 1)
        k = transmission_channel[fr_ps->header->tc_allocation][m];
    else
    {
        if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
        else
        {
            k = transmission_channel[fr_ps->header->tc_alloc[ll]][m];
        }
    }

    if(bit_alloc[k][i])
        d = sb_samples[k][s][j][i] / multiple[scalar[k][s][i]];

    if (mod(d) >= 1.0) /* > changed to >=, 1992-11-06 shn */
    {
        printf("Not scaled properly, %d %d %d %d\n",k,s,j,i);
        printf("Value %1.10f\n",sb_samples[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;
#ifdef 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<12;i++) for (k = 2; k < 7; k++) sbband[k][s][i][j] = 0;
}

/*****
/*
/* I_encode_bit_alloc (Layer I)
/* II_encode_bit_alloc (Layer II)
/* II_encode_bit_allocmc (Layer II multichannel)
/*
/* 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[7][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *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[7][SBLIMIT];
frame_params *fr_ps;

```

```

Bit_stream_struct *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);
}

void II_encode_bit_allocmc(bit_alloc, fr_ps, bs)
unsigned int bit_alloc[7][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *bs;
{
    int i,k, l, m;
    int stereo = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsboundmc = fr_ps->jsboundmc;
    al_table *alloc = fr_ps->alloc;

    for (i=0;i<sblimit;i++)
    {
        for(m = 2; m < 5; ++m)
        {
            if(fr_ps->header->tc_sbgr_select == 1)
                k = transmission_channel[fr_ps->header->tc_allocation][m];
            else
            {
                if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
                else
                {
                    for(l = 1; l < 12; l++)
                    {
                        if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
                        {
                            k = transmission_channel[fr_ps->header->tc_alloc[l]][m];
                            break;
                        }
                    }
                }
            }
            putbits(bs, bit_alloc[k][i], (*alloc)[i][0].bits);
        }
    }
}

/*****
/*

```

```

/* I_sample_encoding (Layer I)
/* II_sample_encoding (Layer II)
/* II_sample_encodingmc (Layer II) SR
/*
/* 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[7][3][12][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *bs;
{
    int i,j,k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;

    for(j=0;j<12;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[7][3][12][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *bs;
{
    unsigned int temp;
    unsigned int i,j,k,s,x,y;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    int jsboundmc = fr_ps->jsboundmc;
    al_table *alloc = fr_ps->alloc;

    for (s=0;s<3;s++)
        for (j=0;j<12;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);
        }
    }
}

```

/\*\*\*\*\*\* Layer II five channels \*\*\*\*\*/

```

void II_sample_encodingmc(sbband, bit_alloc, fr_ps, bs)
unsigned int /*far*/ sbband[7][3][12][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
frame_params *fr_ps;
Bit_stream_struct *bs;
{
    unsigned int temp;
    unsigned int i,j,k,s,x,y, l, m;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsboundmc = fr_ps->jsboundmc;
    al_table *alloc = fr_ps->alloc;

    for (s=0;s<3;s++)
        for (j=0;j<12;j+=3)
            for (i=0;i<sblimit;i++)
                for (m = 2; m < 5; m++)
                {
                    if(fr_ps->header->tc_sbgr_select == 1)
                        k = transmission_channel[fr_ps->header->tc_allocation][m];
                    else
                    {
                        if(i == 0) k = transmission_channel[fr_ps->header->tc_alloc[0]][m];
                        else
                        {
                            for(l = 1; l < 12; l++)
                            {
                                if((sb_groups[l-1] < i) && (i <= sb_groups[l]))
                                {
                                    k = transmission_channel[fr_ps->header->tc_alloc[l]][m];
                                    break;
                                }
                            }
                        }
                    }
                }

    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_struct *bs;
{
    putbits(bs, crc, 16);
}
encoder.h
/*****
Copyright (c) 1991 MPEG/audio software simulation group, 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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
* *
* VERSION 2.5 *
* 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. *
* 13jul92 Susanne Ritscher   MS-DOS, MSC 6.0 port fix. *
* 92-11-06 Soren H. Nielsen   Changed POWERNORM to 96 dB in order *
*                               to get FFT levels conforming to ISO. *
* dec 92 Susanne Ritscher   Changed to multi channel with several *
*                               options *
*****
*                               *
*                               *
* MPEG/audio Phase 2 coding/decoding multichannel *
*                               *
* 7/27/93   Susanne Ritscher, IRT Munich *
* 8/27/93   Susanne Ritscher, IRT Munich *
*           Channel-Switching is working *
* 9/1/93    Susanne Ritscher, IRT Munich *
*           all channels normalized *
*                               *
* 9/20/93   channel-switching is only performed at a *
*           certain limit of TC_ALLOC dB, which is included *
*           in encoder.h *
*                               *
* Version 1.0 Shareware *
*                               *
* 07/12/94   Susanne Ritscher, IRT Munich *
*           Tel: +49 89 32399 458 *
*           Fax: +49 89 32399 415 *
*****/

/******
*
* 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      'r'        /* default mode is stereo front channels */
#define DFLT_MODMC    's'        /* default mode multi channel is stereo */
#define DFLT_PSY      1           /* default psych model is 2 */

```

```

#define DFLT_SFQ      48    /* default input sampling rate is 44.1 kHz, now 48kHz 28.6.93 SR */
#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'

#define TC_ALLOC      0.0

/* 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 */
/* Now optionally in fr_ps */

/*#define NOISY_MIN_MNR 0.0*/

/* Psychoacoustic 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
                        /*96.0*/
                        /* Full amplitude, 32767, should correspond to
                           96 dB, 1992-11-06 Soren H. Nielsen */
                        /* 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 double  D5SBL[5][SBLIMIT];
typedef double  D7SBL[7][SBLIMIT]; /*added because of 7 channels, 8/10/93, SR*/

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     IHLK[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 Variable External Declarations
*
*****/

#ifdef MS_DOS
/* extern unsigned _stklen = 16384; */ /* removed, 92-07-13 sr */
#endif

/*****
*
* 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], IFF_AIFF*, int*, int*);
extern void  parse_args(int, char**, frame_params*, int*, unsigned long*,
                               char[MAX_NAME_SIZE], char[MAX_NAME_SIZE], int*, int*);
extern void  print_config(frame_params*, int*, unsigned long*,
                               char[MAX_NAME_SIZE], char[MAX_NAME_SIZE], int*);
extern 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*, long[5760], unsigned long,
                                unsigned long,int*,
                                int*);
extern unsigned long get_audio(FILE*, long[5][1152], unsigned long,
                                int, IFF_AIFF*, int,
                                layer, int*, int*);
extern void matcing( long[5][1152], int);
extern void read_ana_window(double[HAN_SIZE]);
extern void window_subband(long**, 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[2][3][SCALE_BLOCK][SBLIMIT]);
extern void II_combine_LR(double[5][3][SCALE_BLOCK][SBLIMIT],
                        double[2][3][SCALE_BLOCK][SBLIMIT], int);
extern void I_scale_factor_calc(double[2][3][SCALE_BLOCK][SBLIMIT],
                                unsigned int[2][3][SBLIMIT], int);
extern void II_scale_factor_calc(double[2][3][SCALE_BLOCK][SBLIMIT],
                                unsigned int[2][3][SBLIMIT], int, int);
extern void II_scale_factor_calc1(double[2][3][SCALE_BLOCK][SBLIMIT],
                                unsigned int[2][3][SBLIMIT], int, 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[5][SBLIMIT],
                            unsigned int[5][SBLIMIT],
                            unsigned int[5][3][SBLIMIT], frame_params*,
                            Bit_stream_struc*, int*, int*);
extern void I_encode_scale(unsigned int[2][3][SBLIMIT],
                            unsigned int[2][SBLIMIT], frame_params*,
                            Bit_stream_struc*);
extern int II_bits_for_nonnoise(double[5][SBLIMIT], unsigned int[5][SBLIMIT],
                                frame_params*, int*, int*, int*);
extern void II_main_bit_allocation(double[2][SBLIMIT],
                                unsigned int[2][SBLIMIT], unsigned int[2][SBLIMIT],
                                int*, int*, int*, frame_params*, int*);
extern int II_a_bit_allocation(double[5][SBLIMIT], unsigned int[5][SBLIMIT],
                                unsigned int[5][SBLIMIT], int*, int*, int*, frame_params*, int*, int*);
extern int I_bits_for_nonnoise(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[5][3][SBLIMIT],
        double[5][3][12][SBLIMIT], unsigned int[2][3][SBLIMIT],
        double[2][3][12][SBLIMIT], unsigned int[5][SBLIMIT],
        unsigned int[5][3][12][SBLIMIT], frame_params*);
extern void II_subband_quantization(unsigned int[5][3][SBLIMIT],
        double[5][3][12][SBLIMIT], unsigned int[2][3][SBLIMIT],
        double[2][3][12][SBLIMIT], unsigned int[5][SBLIMIT],
        unsigned int[5][3][12][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][12][SBLIMIT],
        unsigned int[2][SBLIMIT], frame_params*,
        Bit_stream_struc*);
extern void II_sample_encoding(unsigned int[2][3][12][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      matricing();
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      II_combine_SLR();
extern void      II_combine_LRC();
extern void      II_combine_CSLR();
extern void      II_combine_ALL();
extern void      I_scale_factor_calc();
extern void      II_scale_factor_calc();
extern void      II_scale_factor_calc1();
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();
extern void    matricing_fft();
extern void    encode_infomc2();
extern void    tc_alloc();
extern int     II_bits_for_indi();
extern int     required_bits();
extern int     max_alloc();
extern void    II_subband_quantizationmc();
extern void    II_encode_bit_allocmc();
extern void    II_sample_encodingmc();

#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(long[5][1152], double[5][SBLIMIT],
                           double[5][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(long[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();
extern double  non_lin_add();

#endif

/* The following functions are in the file "psy.c" */

#ifdef  PROTO_ARGS
extern void    psycho_anal(long 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]);
#else
extern void    fft();
#endif
musicin.c
/*****
Copyright (c) 1991 MPEG/audio software simulation group, 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, 508-493-2241, e-mail: pan@gauss.enet.dec.com      *
*                                                              *
* VERSION 2.5                                                *
* 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 *
* 10 jul 92 Susanne Ritscher Bug fix in main, scale factor calc. *
* 5 aug 92 Soren H. Nielsen Printout of bit allocation. *
* 19 aug 92 Soren H. Nielsen Changed MS-DOS file name extensions. *
* 2 dec 92 Susanne Ritscher Start of changes to multi-channel with*
*          several options *
*

```

```

*****

```

```

*
*
* MPEG/audio Phase 2 coding/decoding multichannel
*
*
* 7/27/93    Susanne Ritscher, IRT Munich
*
*
* 8/13/93    implemented channel-switching by changing
*             a lot in encode.c
*
* 8/27/93    Susanne Ritscher, IRT Munich
*             Channel-Switching is working
* 9/1/93     Susanne Ritscher, IRT Munich
*             all channels normalized
* 9/20/93    channel-switching is only performed at a
*             certain limit of TC_ALLOC dB, which is included
*             in encoder.h
* 1/04/94    try get all the rubbish out!
*
*
* Version 1.0 Shareware
*
* 07/12/94    Susanne Ritscher, IRT Munich
*             Tel: +49 89 32399 458
*             Fax: +49 89 32399 415
*****/
#ifdef MS_DOS
#include <dos.h>
#endif
#include "common.h"
#include "encoder.h"
#include <math.h>

/* Global variable definitions for "musicin.c" */

FILE          *musicin;
Bit_stream_struct 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,
                  pcm_aiff_data, aiff, byte_per_sample, cha_sw)
frame_params  *fr_ps;
int            *psy;
unsigned long  *num_samples;
char           original_file_name[MAX_NAME_SIZE];
char           encoded_file_name[MAX_NAME_SIZE];
IFF_AIFF       *pcm_aiff_data;
int            *aiff;
int            *byte_per_sample;
int            *cha_sw;
{
    int j;
    long int i;
    char t[50];

    layer *info = fr_ps->header;

    *aiff = 0;          /* flag for AIFF-Soundfile*/
    *cha_sw = 0;

    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(0);
        }

#ifdef MSDOS
        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 MSDOS
            strcpy(encoded_file_name,new_ext(original_file_name, DFLT_EXT));
            /* replace old extension with new one, 92-08-19 shn */
#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 (aiff_read_headers(musicin, pcm_aiff_data, byte_per_sample) == 0) {

        printf(">>> Using Audio IFF sound file headers\n");

        *aiff = 1;
        aiff_check(original_file_name, pcm_aiff_data);

/* if (aiff_seek_to_sound_data(musicin) == -1) {
    printf("Could not seek to PCM sound data in \"%s\".\n",
        original_file_name);
    exit(0);
} */

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 */
#ifdef 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("no multichannel coding!!\n");
        printf("What is the sampling frequency? <48000>[Hz]: ");
        gets(t);
        i = atol(t);
        switch (i) {
            case 48000 : info->sampling_frequency = 1;
                printf(">>> %ld Hz sampling freq selected\n", i);
                break;
            case 44100 : info->sampling_frequency = 0;
                printf(">>> %ld Hz sampling freq selected\n", i);
                break;
            case 32000 : info->sampling_frequency = 2;
                printf(">>> %ld Hz sampling freq selected\n", i);
                break;
            default: info->sampling_frequency = 1;
                printf(">>> Default 48 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(0);
        }

/* 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;
}

    if( *aiff == 1)
    {
        printf("Which mode do you want for the two front channels?\n");
        printf("Available: (<s>)tereo, (j)oint stereo, ");
        printf("(d)ual channel, s(i)ngle Channel, n<o>ne: ");
        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;
            case 'o':
            case 'O':
                info->mode = MPG_MD_NONE; info->mode_ext = 0;
                printf(">>> Front-channels are not coded");
                break;
            default:
                info->mode = MPG_MD_STEREO; info->mode_ext = 0;
                printf(">>> Using default stereo mode\n");
                break;
        }
    }
    else /* not aiff */
    {
        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 'T':
        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;
}
}

*psy = 1;

printf("What is the total bitrate? <%u>[kbps]: ", DFLT_BRT);
gets(t);
i = atol(t);
if (i == 0) i = -10;
j=0;
while (j<=15)
{
    if (bitrate[info->lay-1][j] == (int) i) break;
    j++;
}
if (j==16)
{
    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");

/* Option for multichannel for matricing, 7/12/93,SR*/
if(*aiff == 1)
{
    printf("which kind of matrix do you want(<(-3, -3) = 0>;(-xx, -3) = 1;");
    printf(" (-oo, -3) = 2; (-oo, -oo) = 3) ");
    gets(t);
    if(strcmp(t,"") == 0) info->matrix = 0;
    else info->matrix = atoi(t);
    printf("The matrix %d is chosen\n", info->matrix);
}

```

```

        printf("Do you want to have Channel-switching based on SCF?(y/<n>");
        gets(t);
        if(*t == 'y')
        {
            *cha_sw = 1;
            printf("Channel-switching on SCF is used!\n");
        }
        else
        {
            *cha_sw = 0;
            printf("Channel-switching is not used\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 of the two front channels
/* -r 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
/* -g is followed by the matrix
/* -k is followed by the channel-switching flag based on SCF
/*
/* 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, aiff, byte_per_sample,
           cha_sw, pcm_aiff_data)
int  argc;
char **argv;
frame_params *fr_ps;

```

```

int    *psy;
unsigned long *num_samples;
char    inPath[MAX_NAME_SIZE];
char    outPath[MAX_NAME_SIZE];
int      *aiff;
int      *byte_per_sample;
int      *cha_sw;
IFF_AIFF      *pcm_aiff_data;
{
    FLOAT srates;
    int      brate;
layer *info = fr_ps->header;
    int  err = 0, i = 0;

    *cha_sw = 0;
    fr_ps->mnr_min = 0.0;
    info->matrix = 0;
    *aiff = 0;

    /* preset defaults */
    inPath[0] = '\0'; outPath[0] = '\0';
    info->lay = DFLT_LAY;
    switch(DFLT_MOD) {
        case 'r': 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();
    }
    info->bitrate_index = DFLT_BRT;
    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 if (*arg == 'n')
                        { info->mode = MPG_MD_NONE; 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 'r':
                srates = atof(arg); argUsed = 1;
                if( (info->sampling_frequency =
                    SmpFrqIndex((long)(1000*srate))) < 0)
                    err = 1;
                    break;
            case 'b':
                brate = atoi(arg); argUsed = 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;

```

```

        case 'g':    info->matrix = atoi(arg);
                    argUsed = 1;
                    break;
        case 'k':    *cha_sw = 1; 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( (info->bitrate_index = BitrateIndex(info->lay, brate)) < 0) 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(0);
}

open_bit_stream_w(&bs, outPath, BUFFER_SIZE);

if (aiff_read_headers(musicin, pcm_aiff_data, byte_per_sample) == 0) {

    printf(">>> Using Audio IFF sound file headers\n");

    *aiff = 1;
    printf(">>> Using Audio IFF sound file headers\n");

    aiff_check(inPath, pcm_aiff_data);

    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 */
#ifdef 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(" NO MULTICHANNEL CODING!!\n");
        if (fseek(musicin, 0, SEEK_SET) != 0) {
            printf("Could not seek to PCM sound data in \"%s\".\n", inPath);
            exit(0);
        }

        /* 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, aiff)
frame_params *fr_ps;
int *psy;
unsigned long *num_samples;
char inPath[MAX_NAME_SIZE];
char outPath[MAX_NAME_SIZE];
int *aiff;
{
    layer *info = fr_ps->header;

    printf("Encoding configuration:\n");
    if(*aiff == 1){
        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=%d psy model=%d\n",
            layer_names[info->lay-1], mode_names[info->mode],
            info->mode_ext, *psy);
    }
}

```

```

        if( info->bitrate_index != 0){
            if( bitrate[info->lay-1][info->bitrate_index] == 1000)
                printf("samp frq=%.1f kHz  total bitrate=dynamic bitrate\n",
                    s_freq[info->sampling_frequency]);
            else
                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),now portated
/* to multichannel (two front channels, one center and three surround
/* channels. There are different possibilities to code the AIFF-signal,
/* like coding only the surround channels or the front channels,
/* different bitrates for front and surropund channels etc.
/* dec1992 sr.
/*
/* 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)
/*
/******

#ifdef MS_DOS
extern unsigned _stklen = 16384;
#endif

main(argc, argv)
int  argc;
char **argv;
{
/*typedef double SBS[7][3][12][SBLIMIT];*/
/* SBS          *sbsample;*/
double sb_sample[7][3][12][SBLIMIT];
typedef double JSBS[2][3][12][SBLIMIT];
JSBS /*far*/      *j_sample;
typedef double IN[5][HAN_SIZE];
IN /*far*/        *win_que;
typedef unsigned int SUB[7][3][12][SBLIMIT];
SUB /*far*/       *subband;

frame_params fr_ps;
layer info;
char original_file_name[MAX_NAME_SIZE];
char encoded_file_name[MAX_NAME_SIZE];
long *win_buf[5];
static long buffer[5][1152];
double spiki[7][SBLIMIT];
static unsigned int bit_alloc[7][SBLIMIT], scfsi[7][SBLIMIT], scfsi_dyn[7][SBLIMIT];
static unsigned int scalar[7][3][SBLIMIT], j_scale[2][3][SBLIMIT];
static double ltmin[7][SBLIMIT], lgmin[7][SBLIMIT], max_sc[7][SBLIMIT], smr[7][SBLIMIT];
FLOAT snr32[32];
short sam[7][1056];
double buffer_matr[7][1152];
int whole_SpF, extra_slot = 0;
double avg_slots_per_frame, frac_SpF, slot_lag;
int model, stereo, error_protection, stereomc;
static unsigned int crc, crcmc;
int i, j, k, adb, p, l, m;
unsigned long bitsPerSlot, samplesPerFrame, frameNum = 0;
unsigned long frameBits, sentBits = 0;
unsigned long num_samples;
IFF_AIFF aiff_ptr;
int aiff = 0;
int byte_per_sample = 0;
int cha_sw = 0;
int predis = 0;

#ifdef MACINTOSH
console_options.nrows = MAC_WINDOW_SIZE;

```

```

    argc = ccommand(&argv);
#endif

#ifdef PRINTOUT
    int    loop_channel, loop_subband;

    al_table *loop_alloc; /* a pointer to a table */
    sb_alloc loop_struct; /* a structure of 4 int's */
    alloc_ptr loop_str_ptr; /* a pointer to an sb_alloc structure */
    float    loop_bits;
#endif

    /* Most large variables are declared dynamically to ensure
       compatibility with smaller machines */

    /*sb_sample = (SBS *) mem_alloc(sizeof(SBS), "sb_sample"); */
    j_sample = (JSBS *) mem_alloc(sizeof(JSBS), "j_sample");

    win_que = (IN *) mem_alloc(sizeof(IN), "Win_que");
    subband = (SUB *) mem_alloc(sizeof(SUB), "subband");

    /* 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;
    info.mode_ext = 0;
    fr_ps.tab_num = -1; /* no table loaded */
    fr_ps.alloc = NULL;
    info.version = MPEG_AUDIO_ID;
    info.bitrate_index = 0;
    info.lfe = 0; /* no low frequency effect channel present! */
    info.multiling_ch = 0; /* not done yet */
    info.multiling_fs = 0; /* dto */
    info.multiling_lay = 0; /* dto */
    info.ext_bit_stream_present = 0;
    info.audio_mix = 0;

    programName = argv[0];

```

```

if(argc==1)                                /* no command-line args */
    obtain_parameters(&fr_ps, &model, &num_samples, original_file_name,
                      encoded_file_name, &aiff_ptr, &aiff, &byte_per_sample,
                      &cha_sw);
else
    parse_args(argc, argv, &fr_ps, &model, &num_samples,
                original_file_name, encoded_file_name, &aiff, &byte_per_sample,
                &cha_sw, &aiff_ptr);
print_config(&fr_ps, &model, &num_samples,
             original_file_name, encoded_file_name, &aiff);

hdr_to_frps(&fr_ps);
stereo = fr_ps.stereo;
stereomc = fr_ps.stereomc;
if( aiff != 1) fr_ps.stereomc = 0;
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;    /* Bytes per frame within datastream*/
printf("slots/frame = %d\n", whole_SpF);
frac_SpF = avg_slots_per_frame - (double)whole_SpF;
slot_lag = -frac_SpF;
printf("frac SpF=%0.3f, tot bitrate=%d kbps, s freq=%0.1f 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;

#ifdef PRINTOUT

printf("\nFrame ");
for (loop_subband=0; loop_subband<SBLIMIT; loop_subband++)
    printf("%3d", loop_subband);
printf("\n");

#endif

while (get_audio(musicin, buffer, num_samples,
                 stereo, &aiff_ptr, stereomc, &fr_ps, &aiff, &byte_per_sample,
                 buffer_matr) > 0) {

/*the following allocation must happen within the while-loop. 1/5/93, SR*/

    info.mc_prediction_on = 0;
    info.tc_sbgr_select = 1;
    info.dyn_cross_on = 0;
    for(i = 0; i < 12; i++){

```

```

    info.tc_alloc[i] = 0;
}

fprintf(stderr, "{ %4lu}", frameNum++); fflush(stderr);
win_buf[0] = &buffer[0][0];
win_buf[1] = &buffer[1][0];
win_buf[2] = &buffer[2][0];
win_buf[3] = &buffer[3][0];
win_buf[4] = &buffer[4][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);
    }
}
adb = (whole_SpF+extra_slot) * bitsPerSlot;

switch (info.lay) {

/***** Layer I *****/

case 1 :
    for (j=0;j<12;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(***/sb_sample, scalar, 1);
    }

    put_scale(scalar, &fr_ps, max_sc);

    I_Psycho_One(buffer, max_sc, ltmin, &fr_ps);

    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 :
    if( aiff != 1){
for (i=0;i<3;i++) for (j=0;j<12;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]);
        }

        l = 0;
        if( stereo == 2) m = 2;
        else m = 1;
        II_scale_factor_calc(sb_sample, scalar, fr_ps.sblimit,
            l, m);
        pick_scale(scalar, &fr_ps, max_sc, cha_sw, aiff);
        if(fr_ps.actual_mode == MPG_MD_JOINT_STEREO) {
            II_combine_LR(sb_sample, *j_sample, fr_ps.sblimit);
            II_scale_factor_calc1(*j_sample, j_scale,
                l, fr_ps.sblimit, 0);
        }
    }

```

```

        /* this way we calculate more mono than we need */
        /* but it is cheap */

```

```

II_Psycho_One(buffer, max_sc, ltmin, &fr_ps,smr,spiki,aiff);

```

```

        II_transmission_pattern(scalar, scfsi, &fr_ps);

```

```

        II_main_bit_allocation(smr, ltmin, scfsi, bit_alloc, &adb,
            &fr_ps, &aiff, sb_sample,
            scalar, max_sc, cha_sw, buffer_matr,
            spiki, j_sample, j_scale);

```

```

        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);

```

```

        k = 0;
        i = 2;
        II_encode_scale(bit_alloc, scfsi, scalar, &fr_ps, &bs, &k, &i);
        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);
}

}

```

/\*\*\*\*\*\* Now Layer 2 with five channels\*\*\*\*\*\*/

```

else{
    for (i=0;i<3;i++) for (j=0;j<12;j++)
        for (k = 0; k < stereo+stereomc; k++) {
            window_subband(&win_buf[k], &(*win_que)[k][0], k);
            filter_subband(&(*win_que)[k][0], &(sb_sample)[k][i][j][0]);
        }
}

```

```

    matricing(sb_sample, &fr_ps);
    l = 0; m = 7;
    II_scale_factor_calc(sb_sample, scalar, fr_ps.sblimit,
        l, m);
    pick_scale(scalar, &fr_ps, max_sc, cha_sw, aiff);

    if(fr_ps.actual_mode == MPG_MD_JOINT_STEREO)
    {
        fprintf(stderr, "JOINT!!\n"); fflush(stderr);
        i = 0;
        II_combine_LR(sb_sample, *j_sample, fr_ps.sblimit);
        II_scale_factor_calc1(*j_sample, j_scale,
            fr_ps.sblimit, i);
    }

    II_Psycho_One(buffer_matr, max_sc, ltmin,
        &fr_ps, smr, spiki, aiff);

```

```

    II_transmission_pattern(scalar, scfsi, &fr_ps, scfsi_dyn);

    II_main_bit_allocation(smr, ltmin, scfsi, bit_alloc, &adb,
        &fr_ps, &aiff, sb_sample,
        scalar, max_sc, cha_sw, buffer_matr,
        spiki, j_sample, j_scale);

```

```

    /* PREDISTORTION, 4/7/94, SR*/
    if(fr_ps.actual_mode != MPG_MD_JOINT_STEREO)
    {
        if(info.matrix == 0)
        {
            adb = (whole_SpF+extra_slot) * bitsPerSlot;
            predistortion(sb_sample, scalar, bit_alloc,
                *subband, &fr_ps, smr, scfsi, &adb);
            predis = 1;
        }
    }

```



```

    }
    /***** PREDISTORTION-END*****/

    if (error_protection)
        II_CRC_calc(&fr_ps, bit_alloc, scfsi, &crc);

    II_CRC_calcmc(&fr_ps, bit_alloc, scfsi, &crcmc);

    encode_info(&fr_ps, &bs);

    if (error_protection) encode_CRC(crc, &bs);

    II_encode_bit_alloc(bit_alloc, &fr_ps, &bs);
    k = 0;
    i = 2;

    II_encode_scale(bit_alloc, scfsi, scalar, &fr_ps, &bs, &k, &i);

    II_subband_quantization(scalar, sb_sample, j_scale,
        *j_sample, bit_alloc, *subband, &fr_ps);

    II_sample_encoding(*subband, bit_alloc, &fr_ps, &bs);

    /***** Now bitstream for the surround channels *****/

    encode_infomc1(&fr_ps, &bs);

    encode_CRC(crcmc, &bs);

    encode_infomc2(&fr_ps, &bs); /*new draft 5.7.93 SR*/

    II_encode_bit_allocmc(bit_alloc, &fr_ps, &bs);
    k = 2;
    i = 5;

    II_encode_scale(bit_alloc, scfsi, scalar, &fr_ps, &bs, &k, &i);

    II_subband_quantizationmc(scalar, sb_sample, j_scale,
        *j_sample, bit_alloc, *subband, &fr_ps);

    II_sample_encodingmc(*subband, bit_alloc, &fr_ps, &bs);

    for (i = 0; i < adb; i++)
        put1bit(&bs, 0);

```

```

        /* init = 1;*/ /*no more init*/
    }

    /****** Layer 3, not done yet!!******/

    /*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;

#ifdef PRINTOUT
printf("\nFrame %4lu  channel 1 channel 2 channel 3 channel 4 channel 5\n",frameNum-1);
for (loop_subband=0; loop_subband<SBLIMIT; loop_subband++)
{
    printf("subband %2d :",loop_subband);
    for (loop_channel=0; loop_channel<fr_ps.stereo + stereomc; loop_channel++)
    {
        /* make loop_alloc point to the alloc-table in fr_ps */
        loop_alloc=fr_ps.alloc;
        loop_bits=(* loop_alloc)[loop_subband][bit_alloc[loop_channel][loop_subband]].bits;
        if ((loop_channel==1) &&
            (fr_ps.actual_mode == MPG_MD_JOINT_STEREO) &&
            (loop_subband >= fr_ps.jsbound))
            printf("js      ");
        else
        {
            if ((* loop_alloc)[loop_subband][bit_alloc[loop_channel][loop_subband]].steps == 0)
                printf(" - bits ");
            else
            {
                if ((* loop_alloc)[loop_subband][bit_alloc[loop_channel][loop_subband]].group == 1)
                    loop_bits /= 3;
                printf("%5.1f bits ",loop_bits);
            }
        }
    }
    /* for (loop_channel..) */
    printf("\n");
} /* for (loop_subband..) */
#endif
}/* end of while(get_audio) - loop */

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);
    if( aiff == 1) printf(" It is a multichannel file !\n");
    else printf(" It is a twochannel file!\n");
    printf("The MPEG encoded output file name is \"%s\".\n",
        encoded_file_name);

}

/*****
/*
/* 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, "-n mode  surround mode : s/d/j/m  (dflt %4c)\n", DFLT_MOD);
    fprintf(stderr, "-r 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, "-g matr  matrix                  (dflt 0)\n");
    fprintf(stderr, "-k       channel-switching      (dflt no)\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 */
{

    if (strcmp(pcm_aiff_data->sampleType, IFF_ID_SSND) != 0) {
        printf("Sound data is not PCM in \"%s\".\n", file_name);
        exit(0);
    }

    if (SmpFrqIndex((long)pcm_aiff_data->sampleRate) < 0) {
        printf("in \"%s\".\n", file_name);
        exit(0);
    }

    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(0);
    }

    if (pcm_aiff_data->numChannels != MONO &&
        pcm_aiff_data->numChannels != STEREO &&
        pcm_aiff_data->numChannels != 3 && /*changed that to five-channel 21.6.93.SR*/
        pcm_aiff_data->numChannels != 4 &&
        pcm_aiff_data->numChannels != 5) {
        printf("Sound data is not mono or stereo or fivechannel in \"%s\".\n", file_name);
        exit(0);
    }

    if (pcm_aiff_data->blkAlgn.blockSize != 0) {
        printf("Block size is not %lu bytes in \"%s\".\n", 0, file_name);
        exit(0);
    }

    if (pcm_aiff_data->blkAlgn.offset != 0) {
        printf("Block offset is not %lu bytes in \"%s\".\n", 0, file_name);
        exit(0);
    }

}

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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
*
*
* VERSION 2.5
* 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.
*****
*
*
* MPEG/audio Phase 2 coding/decoding multichannel
*
* Version 1.0
*
* 7/27/93 Susanne Ritscher, IRT Munich
*
* thanks to
* Ralf Schwalbe, Telekom FTZ Berlin
* Heiko Purnhagen, Uni Hannover

```

```

*
*
* Version 2.0
*
* 8/27/93    Susanne Ritscher, IRT Munich
*           Channel-Switching is working
*
* Version 2.1
*
* 9/1/93     Susanne Ritscher, IRT Munich
*           all channels normalized
*
* Version 3.0
*
* 06/16/94   Ralf Schwalbe, Telekom FTZ Berlin
*           all sources and variables adapted due to MPEG-2 - *
*           DIS from March 1994
*           - dematrix and denormalize procedure
*           - new tc - allocation (0-7)
*           - some new structures and variables as a basis
*           for further decoding modes
*
*
*
* Version 1.0 Shareware
*
* 07/12/94   Ralf Schwalbe, Telekom FTZ Berlin
*           Tel: +49 30 6708 2406
*           Fax: +49 30 6774 539
*****/

#include "common.h"
#include "decoder.h"

/*****
/*
/* This part contains the MPEG-1 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 bits_in_frame = 0;

/* Implementations */

main(argc, argv)
int argc;
char **argv;
{
typedef long PCM[5][3][SBLIMIT];
    PCM *pcm_sample;
typedef unsigned int SAM[5][3][SBLIMIT];
    SAM /*far*/ *sample;
typedef double FRA[12][5][3][SBLIMIT]; /* 7.10.93 R.S. mem - alloc for DOS */
    FRA *fraction;
typedef double VE[5][HAN_SIZE];
    VE /*far*/ *w;

    Bit_stream_struct bs;
    frame_params fr_ps;
    layer info;
    FILE *musicout;
    unsigned long sample_frames;
    int i, j, k, ii, stereo, done=FALSE, clip, sync, f, mc_channel;
    int error_protection, crc_error_count, total_error_count;
    int crc_error_countmc, total_error_countmc;
    unsigned int old_crc, new_crc;
    unsigned int bit_alloc[5][SBLIMIT], scfsi[5][SBLIMIT],
        scale_index[5][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 encoded_file_name1[MAX_NAME_SIZE]; /* 8/11/92.sr*/
    char decoded_file_name[MAX_NAME_SIZE];
    char t[50];
    int need_aiff;
    int topSb = 0;
    int l, m;

#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 *) mem_alloc((long) sizeof(PCM), "PCM Samp");
    sample = (SAM *) mem_alloc((long) sizeof(SAM), "Sample");
    fraction = (FRA *) mem_alloc((long) sizeof(FRA), "fraction"); /* R.S. */
    w = (VE *) mem_alloc((long) sizeof(VE), "w");

    fr_ps.header = &info;
    fr_ps.tab_num = -1; /* no table loaded */
    fr_ps.alloc = NULL;
    info.mode_ext = 0;
    info.version = MPEG_AUDIO_ID;

```

```

info.bitrate_index = 0;
info.bitrate_index1 = 0;
info.bitrate_index2 = 0;
info.lfe = 0; /* no low frequency effect channel present! */
info.no_of_multi_lingual_ch = 0;
info.multi_lingual_fs = 0;
info.multi_lingual_layer = 0;

for (i=0;i<HAN_SIZE;i++) for (j=0;j<5;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);
        f = strlen(encoded_file_name)-4; /*cut off extension.8/11/92.sr*/
        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);

    strcpy(encoded_file_name1, encoded_file_name); /*8/11/92.sr*/
    strcpy(&encoded_file_name1[f], DFLT_OPEXT); /*.dec-extension.8/11/92.sr*/

    printf ("Enter MPEG decoded file name <%s>: ", encoded_file_name1);
    gets (decoded_file_name);
    if (decoded_file_name[0] == NULL_CHAR)
        /*strcat (strcpy(decoded_file_name, encoded_file_name), DFLT_OPEXT);*/
        strcpy(decoded_file_name, encoded_file_name1);
    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;
    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;
            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 ((musicout = fopen(decoded_file_name, "w+b")) == NULL) {
    printf ("Could not create \"%s\".\n", decoded_file_name);
    exit(0);
}

total_error_countmc = 0;
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(0);
    }

sample_frames = 0;

while (!end_bs(&bs))
{
    sync = seek_sync(&bs, SYNC_WORD, SYNC_WORD_LENGTH);
    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 == 2)
            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 */

fprintf(stderr, "{ %4lu}", frameNum++); fflush(stderr);
if (error_protection) buffer_CRC(&bs, &old_crc);

switch (info.lay)
{
/* case 1: Layer I 30.05.1994 Ralf Schwalbe cut off Layer I -
    source code in case of multi-channel decoding */

    case 2:
        {
            bitsPerSlot = 8;    samplesPerFrame = 1152;
            l = 0;              m = stereo;
            II_decode_bitalloc(&bs, bit_alloc, &fr_ps, &l, &m);
            II_decode_scale(&bs, scfsi, bit_alloc, scale_index, &fr_ps, &l, &m);

            if (error_protection) {
                II_CRC_calc(&fr_ps, bit_alloc, scfsi, &new_crc);
                if (new_crc != old_crc)
                {
                    printf("\n ERROR in LAYER 2 - CRC! \n");
                    crc_error_count++;
                    total_error_count++;
                    recover_CRC_error(*pcm_sample, crc_error_count,
                                     &fr_ps, musicout,
                                     &sample_frames);
                }
                else crc_error_count = 0;
            }

            clip = 0;

            for (i=0;i<12;i++)
            {
                II_buffer_sample(&bs,(*sample),bit_alloc,&fr_ps);
                II_dequantize_sample((*sample),bit_alloc,*fraction,&fr_ps, &i);
                II_denormalize_sample(*fraction,scale_index,&fr_ps,i>>2, &i);

                if(topSb>0) /*debug : clear channels to 0 */
                    for(j=topSb; j<fr_ps.sblimit; ++j)
                        for(k=0; k<stereo; ++k)
                            (*fraction)[i][k][0][j] =
                                (*fraction)[i][k][1][j] =
                                (*fraction)[i][k][2][j] = 0;

            } /* end of for loop */

            /***/
            /*
            /* multichannel - decoding */
            /* 7.07.93 Susanne Ritscher */
            /* 13.10.93 Ralf Schwalbe */
            /* 30.05.94 Ralf Schwalbe */
            /***/

```

```

mc_header(&bs, &fr_ps);
hdr_to_frps(&fr_ps);
mc_channel = fr_ps.mc_channel;
crc_error_count = 0;
total_error_count = 0;
buffer_CRC(&bs, &old_crc); /* read CRC - check from header */
mc_composite_status_info(&bs, &fr_ps);

fr_ps.jsbound = 27;
l = 2;
m = stereo + mc_channel;

II_decode_bitalloc(&bs, bit_alloc, &fr_ps, &l, &m);
II_decode_scale(&bs, scfsi, bit_alloc, scale_index, &fr_ps, &l, &m);

#ifdef CRC_CHECK

mc_error_check(&fr_ps, bit_alloc, scfsi, &new_crc);
if (new_crc != old_crc)
{
    crc_error_count++;
    total_error_countmc++;
    for( i = 0; i < SBLIMIT; i++)
        for(ii = 2; ii < 5; ++ii)
            bit_alloc[ii][i] = 0;
    mc_channel = 3;
    fr_ps.mc_channel = 3;
    printf("ERROR in MC - CRC -> Multichannel extension can't be decoded
!\n");
}
else
    crc_error_count = 0;
#endif

clip = 0;
for (i=0;i<12;i++)
{
    II_buffer_samplemc(&bs,(*sample),bit_alloc,&fr_ps,&m);
    II_dequantize_samplemc((*sample),bit_alloc,*fraction,&fr_ps, &m, &i);
    II_denormalize_samplemc(*fraction,scale_index,&fr_ps,i>>2, &m, &i);

    if(topSb>0) /* debug : clear channels to 0 */
        for(j=topSb; j<fr_ps.sblimit; ++j)
            for(k= 1; k < m; ++k)
                (*fraction)[i][k][0][j] =
                (*fraction)[i][k][1][j] =
                (*fraction)[i][k][2][j] = 0;

    /* dematricing*/
    if(crc_error_count == 0)
        dematricing(*fraction, &fr_ps, &i);

    for (j=0;j<3;j++) for (k=0; k<5; k++)
    {
        clip += SubBandSynthesis (&((*fraction)[i][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;

                                }/*end of layer 2*/

                                }/*end of switch layer - loop*/

                                }/*end of while(!endof(bs)) - loop */

                                if (need_aiff)
                                {
                                    pcm_aiff_data.numChannels    = stereo + mc_channel;
                                    pcm_aiff_data.numSampleFrames = sample_frames;
                                    pcm_aiff_data.sampleSize      = 16;
                                    pcm_aiff_data.sampleRate      = s_freq[info.sampling_frequency]*1000;
                                    strcpy(pcm_aiff_data.sampleType, IFF_ID_SSND);
                                    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);
        if(total_error_countmc != 0)
            printf("There were %d frames, which were not in multichannel!!\n", total_error_countmc);

return(1);
}

```

```

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);
}

```

predisto.c

/\*\*\*\*\*\*

Copyright (c) 1994 Susanne Ritscher, IRT Munich, All Rights Reserved

predistortion.c

\*\*\*\*\*/

/\*\*\*\*\*\*

```

*
*
* MPEG/audio Phase 2 coding/decoding multichannel
*
* 04/08/94      started with predistortion
*
* 06/28/94      started with predistortion according to the DIS
*                and the tc_table
*
* Version 1.0 Shareware
*
* 07/12/94      Susanne Ritscher, IRT Munich
*                Tel: +49 89 32399 458
*                Fax: +49 89 32399 415

```

\*\*\*\*\*/

/\*\*\*\*\*\*

```

*
* This program computes the predistortion.
* The three additional channels are ACCORDING TO THE BITALLOCATION
* quantized and dequantized. With these dequantized samples
* the frontchannels are matrixed again in order to dematrice them in a
* more exact way. This should improve the quality of matrixed signals.

```

```

*
* 07/12/94 Susanne Ritscher
*****
#include "common.h"
#include "encoder.h"

void predistortion(sb_sample, scalar, bit_alloc, subband, fr_ps, perm_smr,
                  scfsi, adb, scfsi_dyn)
double sb_sample[7][3][12][SBLIMIT];
unsigned int scalar[7][3][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
unsigned int subband[7][3][12][SBLIMIT];
frame_params *fr_ps;
double perm_smr[7][SBLIMIT];
unsigned scfsi[7][SBLIMIT];
int *adb;
unsigned int scfsi_dyn[7][SBLIMIT];
{
    double sb_pre_sample[7][3][12][SBLIMIT]; /*predistortion*/
    double sbs_sample[7][3][12][SBLIMIT]; /*predistortion*/
    double test[12][7][3][SBLIMIT]; /*predistortion*/
    unsigned int sample[7][3][SBLIMIT];
    unsigned int keep_it[7][36][SBLIMIT];
    int l, m, i, n, k;
    int sblimit;
    int ch, gr, sb;
    int ad;

        sblimit = fr_ps->sblimit;
        l = 2; m = 7;

        subband_quantization_pre(scalar, sb_sample, bit_alloc, subband, fr_ps, l, m);

    for (ch=2;ch<7;ch++)
        for (sb=0;sb<sblimit;sb++)
            for (gr=0;gr<3;gr++)
                for (i=0;i<12;i++)
                    keep_it[ch][i+gr*12][sb] = subband[ch][gr][i][sb];

                    for(i = 0; i < 12; i++)
                    {
                        k = transmission_channel[fr_ps->header->tc_alloc[3]][4];
                        buffer_sample(keep_it,sample,bit_alloc,fr_ps, l, m, i);
                        II_dequantize_sample(sample, bit_alloc, sb_pre_sample, fr_ps, l, m, i);
                        II_denormalize_sample(sb_pre_sample, scalar, fr_ps, i>>2, l, m, i);
                    }

    for (ch = 2; ch < 7; ch++)
        for (sb=0;sb<sblimit;sb++)
            for (i = 0; i < 4; i++)
                for (gr=0;gr<3;gr++)
                    sbs_sample[ch][0][i*3 + gr][sb] = sb_pre_sample[ch][gr][i][sb];

    for (ch = 2; ch < 7; ch++)
        for (sb=0;sb<sblimit;sb++)

```

```

        for (i = 4; i < 8; i++)
            for (gr=0;gr<3;gr++)
                sbs_sample[ch][1][(i-4)*3 + gr][sb] = sb_pre_sample[ch][gr][i][sb];

    for (ch = 2; ch < 7; ch++)
        for (sb=0;sb<sblimit;sb++)
            for (i = 8; i < 12; i++)
                for (gr=0;gr<3;gr++)
                    sbs_sample[ch][2][(i-8)*3 + gr][sb] = sb_pre_sample[ch][gr][i][sb];

    matri(sbs_sample, fr_ps, sb_sample);

    l = 0; m = 2;

    II_scale_factor_calc(sb_sample,scalar,sblimit, l, m);
    trans_pattern(scalar, scfsi, fr_ps,scfsi_dyn);

    bit_all(perm_smr, scfsi, bit_alloc, adb, fr_ps);

}

```

```

static double snr[18] = { 0.00, 6.03, 11.80, 15.81, /* 0, 3, 5, 7 */
                        19.03, 23.50, 29.82, 35.99, /* 9,15,31,63 */
                        42.08, 48.13, 54.17, 60.20, /* 127, ... */
                        66.22, 72.25, 78.27, 84.29, /* 2047, ... */
                        90.31, 96.33};           /* 16383, ... */

```

```

int bit_all(perm_smr, scfsi, bit_alloc, adb, fr_ps)
double perm_smr[7][SBLIMIT]; /* minimum masking level */
unsigned int scfsi[7][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
int *adb;
frame_params *fr_ps;
{
    int i, min_ch, min_sb, oth_ch, k, increment, scale, seli, ba, j, l;
    int adb_hlp, adb_hlp1, adb_hlp2;
    int bspl, bscf, bsel, ad, noisy_sbs;
    double mnr[7][SBLIMIT], small;
    char used[7][SBLIMIT];
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    int jsbound = fr_ps->jsbound;
    int jsboundmc = fr_ps->jsboundmc;

```



```

    al_table *alloc = fr_ps->alloc;
    double dynsmr = 0.0; /* border of SMR for dynamic datarate */
    static char init= 0;
    static int banc, berr;
    int bbal, bancmc;
    int ll, pci;
    int bits = 0; /*bits already used for the front-channels*/
    static int sfsPerScfsi[] = { 3,2,1,2 }; /* lookup # sfs per scfsi */

    banc = 32; /* banc: bits for header */;
    if (fr_ps->header->error_protection) berr=16; else berr=0; /* added 92-08-11 shn */

    bancmc = 0;
    bbal = 0;
    if(fr_ps->header->mode == MPG_MD_STEREO)
    {
        for( i = 0; i < sblimit; ++i)
            bbal += 5 * (*alloc)[i][0].bits;
        if(fr_ps->header->center == 3) bbal -= 41;
    }
    if(fr_ps->header->mode == MPG_MD_JOINT_STEREO)
    {
        for(i = 0; i < jsbound; ++i)
            bbal += 5 * (*alloc)[i][0].bits;
        for(i = jsbound; i < sblimit; ++i)
            bbal += 4 * (*alloc)[i][0].bits;
    }

    if(fr_ps->header->ext_bit_stream_present == 0)
        bancmc += 33; /* mc_header + crc + tc_sbgr_select+ dyn_cross_on +
                        mc_prediction_on 01/05/94, SR new! 05/04/94, SR*/
    else
    {
        fprintf(stderr, "not done yet Extension Bitstream!!!\n");
        exit(0);
    }

    if(fr_ps->header->tc_sbgr_select == 0)
        bancmc += 36;
    else
        bancmc += 3;

    if(fr_ps->header->dyn_cross_on == 1)
    {
        fprintf(stderr, "not done yet Dynamic Crosstalk!!!\n");
        exit(0);
    }

    if(fr_ps->header->mc_prediction_on == 1)
    {
        fprintf(stderr, "not done yet Prediction!!!\n");
        exit(0);
    }

```

```

for(i = 0; i < 12; i++)
{
    for(l = ((i == 0) ? 0 : (sb_groups[i-1]+1)); l <= sb_groups[i]; l++)
    {
        for(j = 2; j < 5; j++)
        {
            k = transmission_channel[fr_ps->header->tc_alloc[i]][j];

            bits += 12 * ((*alloc)[l][bit_alloc[k][l]].group *
                          (*alloc)[l][bit_alloc[k][l]].bits);
            if(bit_alloc[k][l] != 0)
            {
                bits += 2;
                switch(scfsi[k][l])
                {
                    case 0: bits += 18; break;
                    case 1: bits += 12; break;
                    case 2: bits += 6; break;
                    case 3: bits += 12; break;
                }
            }
        }
    }
}

*adb -= bbal + berr + banc + bancmc + bits;
ad = *adb;

/* according to the hint by Warner ten Kate we take the perm_smr of L */
/* instead of perm_smr of Lo and the perm_smr of R instead perm_smr of */
/* Ro                                05/05/94, SR */
for(l = 0; l < sblimit; l++)
{
    perm_smr[0][l] = perm_smr[5][l];
    perm_smr[1][l] = perm_smr[6][l];
}

for(l = 0; l < sblimit; l++)
{
    for( k = 0; k < 2; k++)
    {
        mnr[k][l] = snr[0] - perm_smr[k][l];
        bit_alloc[k][l] = 0;
        used[k][l] = 0;
    }
}

bsp1 = bscf = bsel = 0;

do {
    small = 999999.0;

```

```

    min_sb = -1;
    min_ch = -1;

    for(i = 0; i < sblimit; i++)
    {
        for( k = 0; k < 2; k++)
        {
            if ((used[k][i] != 2) && (small > mnr[k][i]))
            {
                small = mnr[k][i];
                min_sb = i; min_ch = k;
            }
        }
    }
    if(min_sb > -1)
    {
        increment = 12 * ((*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 -= 12 * ((*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] */
        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 (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 */

    } /* end of if-loop if min_sb > -1 */
} while(min_sb > -1); /* until could find no channel */
/* Calculate the number of bits left */
/* fprintf(stderr,"bsamp = %d\t",bspl); */
/* fprintf(stderr,"bscf = %d\t",bscf); */
/* fprintf(stderr,"bscfsi = %d\n",bsel); */

```

```

/* fflush(stderr);          */

    ad -= bspl+bscf+bsel;
    *adb = ad;
}

void matri(sbs_sample, fr_ps, sb_sample)
double sbs_sample[7][3][12][SBLIMIT];
frame_params *fr_ps;
double sb_sample[7][3][12][SBLIMIT];

{
    int i, j, k, l, ch1, ch2;

    layer *info = fr_ps->header;

        for(k = 0; k < 8; k++)
        {
            switch(fr_ps->header->tc_alloc[k])
            {
                case 0:
                    for( j = 0; j < 3; ++j)
                        for(l = 0; l < 12; l++)
                        {
                            sb_sample[0][j][l][k] = sb_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                            sb_sample[1][j][l][k] = sb_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                        }

                    break;
                case 1:
                    for( j = 0; j < 3; ++j)
                        for(l = 0; l < 12; l++)
                        {
                            sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                            sb_sample[1][j][l][k] = sb_sample[6][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                        }
                    break;
                case 2:
                    for( j = 0; j < 3; ++j)
                        for(l = 0; l < 12; l++)
                        {
                            sb_sample[0][j][l][k] = sb_sample[5][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                            sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                        }
                    break;
                case 3:
                    for( j = 0; j < 3; ++j)
                        for(l = 0; l < 12; l++)
                        {

```

```

        sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[3][j][l][k];
        sb_sample[1][j][l][k] = sb_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[4][j][l][k];
    }
    break;
case 4:
    for( j = 0; j < 3; ++j)
        for(l = 0; l < 12; l++)
        {
            sb_sample[0][j][l][k] = sb_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[3][j][l][k];
            sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[4][j][l][k];
        }
    break;
case 5:
    for( j = 0; j < 3; ++j)
        for(l = 0; l < 12; l++)
        {
            sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[3][j][l][k];
            sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[4][j][l][k];
        }
    break;
case 6:
    for( j = 0; j < 3; ++j)
        for(l = 0; l < 12; l++)
        {
            sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[3][j][l][k];
            sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[4][j][l][k];
        }
    break;
case 7:
    for( j = 0; j < 3; ++j)
        for(l = 0; l < 12; l++)
        {
            sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[3][j][l][k];
            sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[4][j][l][k];
        }
    break;
    }
}

for(i = 8; i < 12; i++)
{
    switch(fr_ps->header->tc_alloc[i])
    {
        case 0:

```

```

        for( j = 0; j < 3; ++j)
            for( l = 0; l < 12; l++)
                for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                {
                    sb_sample[0][j][l][k] = sb_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                    sb_sample[1][j][l][k] = sb_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                }
            break;
        case 1:
            for( j = 0; j < 3; ++j)
                for( l = 0; l < 12; l++)
                    for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                    {
                        sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                        sb_sample[1][j][l][k] = sb_sample[6][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                    }
                break;
        case 2:
            for( j = 0; j < 3; ++j)
                for( l = 0; l < 12; l++)
                    for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                    {
                        sb_sample[0][j][l][k] = sb_sample[5][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                        sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                    }
                break;
        case 3:
            for( j = 0; j < 3; ++j)
                for( l = 0; l < 12; l++)
                    for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                    {
                        sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[3][j][l][k];
                        sb_sample[1][j][l][k] = sb_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                    }
                break;
        case 4:
            for( j = 0; j < 3; ++j)
                for( l = 0; l < 12; l++)
                    for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                    {
                        sb_sample[0][j][l][k] = sb_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                        sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[4][j][l][k];
                    }
                break;
        case 5:
            for( j = 0; j < 3; ++j)
                for( l = 0; l < 12; l++)

```

```

        for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
        {
            sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[3][j][l][k];
            sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sbs_sample[2][j][l][k] + sb_sample[4][j][l][k];
        }
        break;
    case 6:
        for( j = 0; j < 3; ++j)
            for(l = 0; l < 12; l++)
                for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                {
                    sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sb_sample[2][j][l][k] + sb_sample[3][j][l][k];
                    sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[4][j][l][k];
                }
            break;
    case 7:
        for( j = 0; j < 3; ++j)
            for(l = 0; l < 12; l++)
                for( k = sb_groups[i-1] + 1; k <= sb_groups[i]; k++)
                {
                    sb_sample[0][j][l][k] = sbs_sample[5][j][l][k] +
sb_sample[2][j][l][k] + sbs_sample[3][j][l][k];
                    sb_sample[1][j][l][k] = sbs_sample[6][j][l][k] +
sb_sample[2][j][l][k] + sb_sample[4][j][l][k];
                }
            break;
    }
}
}
}

```

```

buffer_sample(keep_it,sample,bit_alloc,fr_ps, l, z, q)
unsigned int keep_it[7][36][SBLIMIT];
unsigned int sample[7][3][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
frame_params *fr_ps;
int l, z, q;
{
    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;
    unsigned int x;

    for (i=0;i<sblimit;i++) for (j= 1;j<((i<jsbound)?z:l+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++)
    {
        sample[j][m][i] = keep_it[j][q*3+m][i];
    }
}
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= 1;j< z;j++) for (k=0;k<3;k++)
    sample[j][k][i] = 0;
}

```

```

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 subband_quantization_pre(scalar, sb_samples, bit_alloc, sbband, fr_ps, ch1, ch2)
unsigned int scalar[7][3][SBLIMIT];
double sb_samples[7][3][12][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
unsigned int sbband[7][3][12][SBLIMIT];
frame_params *fr_ps;
int ch1, ch2;
{
    int i, j, k, s, n, qnt, sig, m, l, ll, z;
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    int sblimit = fr_ps->sblimit;
    unsigned int stps, y;
    double d;
    al_table *alloc = fr_ps->alloc;

    for (s=0;s<3;s++)
        for (j=0;j<12;j=j+3)
            for (i=0;i<sblimit;i++)
                for(k = ch1; k < ch2; k++)

```



```

    {
        for(z = 0; z < 3; z++)
        {
            if(bit_alloc[k][i])
                d = sb_samples[k][s][j+z][i] / multiple[scalar[k][s][i]];

            if (mod(d) >= 1.0) /* > changed to >=, 1992-11-06 shn */
            {
                printf("Not scaled properly, %d %d %d %d\n",k,s,j+z,i);
                printf("Value %1.10f\n",sb_samples[k][s][j+z][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;
#ifdef 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+z][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+z][i] |= 1<<n;

            if(!bit_alloc[k][i]) sbband[k][s][j+z][i] = 0;
        }

        /*try try try ! 7/7/94 */

        /*
            if ((*alloc)[i][bit_alloc[k][i]].group != 3)
            {
                y = (*alloc)[i][bit_alloc[k][i]].steps;
                sbband[k][s][j][i] = sbband[k][s][j][i] +
                    sbband[k][s][j+1][i] * y +
                    sbband[k][s][j+2][i] * y * y;
                sbband[k][s][j+1][i] = sbband[k][s][j][i];
                sbband[k][s][j+2][i] = sbband[k][s][j][i];
            } */

        }
        for (s=0;s<3;s++)
            for (j=sblimit;j<SBLIMIT;j++)
                for (i=0;i<12;i++) for (k = 0; k < 7; k++) sbband[k][s][i][j] = 0;

```

```

}

void trans_pattern(scalar, scfsi, fr_ps, scfsi_dyn)
unsigned int scalar[7][3][SBLIMIT];
unsigned int scfsi[7][SBLIMIT];
frame_params *fr_ps;
unsigned int scfsi_dyn[7][SBLIMIT];

{
    int stereo = fr_ps->stereo;
    int stereomc = fr_ps->stereomc;
    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 < 2; 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];

    }
}
}

```

```

/*****
/*
/* Restore the compressed sample to a fractional number.
/* first complement the MSB of the sample
/* for Layer II :
/* Use the formula  $s = s' * c + d$ 
/*
/* taken out of decoder,modified for predistortion 7/12/94,SR
/*
*****/

```

```

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, l, m, z)
unsigned int sample[7][3][SBLIMIT];
unsigned int bit_alloc[7][SBLIMIT];
double fraction[7][3][12][SBLIMIT];
frame_params *fr_ps;
int l, m, z;
{
    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 = l;k < m ;k++)
        if (bit_alloc[k][i]) {

```

```

    /* locate MSB in the sample */
    x = 0;
#ifdef MSDOS
    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][z][i] = 0.0;
    else fraction[k][j][z][i] = -1.0;

    /* Form a 2's complement sample */
    fraction[k][j][z][i] += (double) (sample[k][j][i] & ((1<<x-1)-1)) /
        (double) (1L<<x-1);

    /* Dequantize the sample */
    fraction[k][j][z][i] += d[(*alloc)[i][bit_alloc[k][i]].quant];
    fraction[k][j][z][i] *= c[(*alloc)[i][bit_alloc[k][i]].quant];
}
else fraction[k][j][z][i] = 0.0;

for (i=sblimit;i<SBLIMIT;i++) for (j=0;j<3;j++) for(k = 1; k < m; k++)
    fraction[k][j][z][i] = 0.0;
}

```

```

void II_denormalize_sample(fraction, scale_index, fr_ps, x, l, m, z)
double /*far*/ fraction[7][3][12][SBLIMIT];
unsigned int scale_index[7][3][SBLIMIT];
frame_params *fr_ps;
int x;
int l, m, z;
{
    int i,j,k;
    int stereo = fr_ps->stereo;
    int sblimit = fr_ps->sblimit;

    for (i=0;i<sblimit;i++) for (j= 1; j < m; j++) {
        fraction[j][0][z][i] *= multiple[scale_index[j][x][i]];
        fraction[j][1][z][i] *= multiple[scale_index[j][x][i]];
        fraction[j][2][z][i] *= multiple[scale_index[j][x][i]];
    }
}

```

```

}
psy.c
/*****
Copyright (c) 1991 MPEG/audio software simulation group, 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, 508-493-2241, e-mail: pan@gauss.enet.dec.com      *
*                                                                *
* VERSION 2.5                                                  *
* 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". *
*****/

#include "common.h"
#include "encoder.h"

void psycho_anal(buffer,savebuf,chn,layers,snr32,sfreq)
long int *buffer;
short int savebuf[1056];
int chn, layers;
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, 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");
    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 = 448;
    syncsize = 1024;
}
else {
    flush = 384*3.0/2.0;
    syncsize = 1056;
}

/* calculate HANN window coefficients */
for(i=0;i<BLKSIZE;i++)window[i]=0.5*(1-cos(2.0*PI*i/(BLKSIZE-1.0)));
/* 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<(syncsize-flush))savebuf[j] = savebuf[j+flush];
        else savebuf[j] = *buffer++;
/*window data with HANN window*****/
        if(j<BLKSIZE){
            wsamp_r[j] = window[j]*((FLOAT) savebuf[j]);
            wsamp_i[j] = 0;
        }
    }
/*Compute FFT*****/
    fft(wsamp_r,wsamp_i,energy,phi);
/******
* calculate the unpredictability measure, given energy[f] and phi[f] *
*****/
    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;
    }
/*only update data "age" pointers after you are done with the second channel */
/*for layer 1 computations, for the layer 2 double computations, the pointers*/
/*are reset automatically on the second pass */
    if(lay==2 || chn==1){
        if(new==0){new = 1; oldest = 1;}
        else {new = 0; oldest = 0;}
        if(oldest==0)old = 1; else old = 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;
        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 Psychoacoustic 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);
}

/* 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;
long table;
{
    FILE *fp;
    long i,j,index;
    float a;
    char t[80], *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(0);
    }
    fgets(t, 150, fp);
    sscanf(t, "table %ld", &index);
    if(index != table){
        printf("error in absthr table %s",ta);
        exit(0);
    }
    for(j=0; j<HBLKSIZE; j++){
        fgets(t,80,fp);
        sscanf(t,"%f", &a);
        absthr[j] = a;
    }
    fclose(fp);
}

subs.c
/*****
Copyright (c) 1991 MPEG/audio software simulation group, 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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
* *
* VERSION 2.5 *
* 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. *
*****/
* *
* *

```

```

* MPEG/audio Phase 2 coding/decoding multichannel      *
*                                     *
* 7/27/93      Susanne Ritscher, IRT Munich             *
* 8/27/93      Susanne Ritscher, IRT Munich             *
*              Channel-Switching is working             *
*                                     *
* 9/1/93       Susanne Ritscher, IRT Munich             *
*              all channels normalized                  *
* 9/20/93      channel-switching is only performed at a *
*              certain limit of TC_ALLOC dB, which is included *
*              in encoder.h                             *
*                                     *
* Version 1.0 Shareware                               *
*                                     *
* 07/12/94     Susanne Ritscher, IRT Munich             *
*              Tel: +49 89 32399 458                    *
*              Fax: +49 89 32399 415                    *
*****/

#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                                              *
*****/

void fft(x_real,x_imag, energy, phi)
FLOAT x_real[BLKSIZE], x_imag[BLKSIZE], energy[BLKSIZE], phi[BLKSIZE];
{
static int  M, MM1;
static int  init=0, N, NV2, NM1;
static double w_real[LOGBLKSIZE], w_imag[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;

```

```

MM1 = LOGBLKSIZE-1;
N = BLKSIZE;
NV2 = BLKSIZE >> 1;
NM1 = BLKSIZE - 1;
for(L=0; L<M; L++){
    le = 1 << (M-L);
    le1 = le >> 1;
    w_real[L] = cos(PI/le1);
    w_imag[L] = -sin(PI/le1);
}
init++;
}
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[L] - u_imag*w_imag[L];
        u_imag = u_imag*w_real[L] + t_real*w_imag[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;
}
}
tonal.c
/*****
Copyright (c) 1991 MPEG/audio software simulation group, 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, 508-493-2241, e-mail: pan@gauss.enet.dec.com *
* *
* VERSION 2.5 *
* 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 *
* 92-11-06 Soren H. Nielsen Corrected power calculation in I_ and *
*           II_f_f_t. *
*****
*           *
*           *
* MPEG/audio Phase 2 coding/decoding multichannel *
*           *
* 7/27/93 Susanne Ritscher, IRT Munich *
* 8/27/93 Susanne Ritscher, IRT Munich *
* Channel-Switching is working *
* 9/1/93 Susanne Ritscher, IRT Munich *
* all channels normalized *
*           *
* 9/20/93 channel-switching is only performed at a *
*           certain limit of TC_ALLOC dB, which is included *
*           in encoder.h *
*           *
* 10/18/93 seperated smr and ltmin *
*           *
* Version 1.0 Shareware *
*           *
* 07/12/94 Susanne Ritscher, IRT Munich *
*           Tel: +49 89 32399 458 *
*           Fax: +49 89 32399 415 *
*****/
#define TONAL_WIDTH 0.1 /* When more than one tonal component is within
                        this width in Bark, the weaker one(s) are
                        eliminated */

#include "common.h"
#include "encoder.h"

/*****
/*
/* 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(0);
    }
    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(0);
        }
    }
    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 a,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(0);
    }
    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(0);
}
}
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, NV2, NM1;
    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;
NV2 = FFT_SIZE >> 1;
NM1 = FFT_SIZE - 1;
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] = 1,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 calculation corrected with a factor 4, both positive

```

```

        and negative frequencies exist, 1992-11-06 shn */
        power[i].x = 10 * log10(energy[i]*4.0) + 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-1)))/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.
/*
*****/

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 */

```

```

/*****
/*
/*      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<2 || first>499) run = 0; /* otherwise k+/-j will be out of bounds */
    else if(first<62) run = 2; /* components in layer II, which */
    else if(first<126) run = 3; /* are the boundaries for calc. */
    else if(first<254) 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<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;
        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);
                    weight += pow(10.0, power[j].x/10.0) * (ltg[power[j].map].bark-i);
                    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 */
        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]) );
        } /* locate next non-tonal component until finished; */
        /* add to list of non-tonal components */
        if(power[centre].type == TONE) centre++;
        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 < TONAL_WIDTH) { /* 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;
            }
            else {
                power[power[i].next].type = FALSE;
                power[power[i].next].x = DBMIN;
                power[i].next = power[power[i].next].next;
            }
        }
    }
}

```

```

    }
  }
  else old = i;
  i = power[i].next;
}
}

/* -----
The masking function parameters are here set according to the parameters in
the IRT real time implementation. The constant definitions are for convenience.
1993-07-23 shn
----- */

#define AV_TONAL_K -9.0 /* Masking index, tonal, constant part [dB] */
#define AV_NOISE_K -5.0 /* Masking index, noisy, constant part [dB] */
#define AV_TONAL_DZ -0.3 /* Masking index, tonal, CBR dependence [dB/Bark] */
#define AV_NOISE_DZ -0.3 /* Masking index, noisy, CBR dependence [dB/Bark] */

#define LOW_LIM_1 -1.0 /* 1st lower slope from 0 to LOW_LIM_1 [Bark] */
#define LOW_LIM_2 -3.0 /* 2nd lower slope from LOW_LIM_1 to LOW_LIM_2 [Bark] */

#define LOW_DZ_K_1 6.0 /* 1st lower slope, constant part [dB/Bark] */
#define LOW_DZ_SPL_1 0.4 /* 1st lower slope, SPL dependence [dB/(Bark*dB)] */
#define LOW_DZ_MIN_1 17.0 /* 1st lower slope, minimum value [dB/Bark] */
#define LOW_DZ_2 17.0 /* 2nd lower slope [dB/Bark] */

#define UP_LIM_1 1.0 /* 1st upper slope from 0 to UP_LIM_1 [Bark] */
#define UP_LIM_2 8.0 /* 2nd upper slope from UP_LIM_1 to UP_LIM_2 [Bark] */

#define UP_DZ_1 -18.0 /* 1st upper slope, constant part [dB/Bark] */
#define UP_SPL_1 0.0 /* 1st upper slope, SPL dependence [dB/(Bark*dB)] */
#define UP_DZ_2 -17.0 /* 2nd upper slope, constant part [dB/Bark] */
#define UP_SPL_2 -0.1 /* 2nd upper slope, SPL dependence [dB/(Bark*dB)] */

#define H_THR_OFFSET -12.0 /* Hearing threshold offset [dB] */
#define H_THR_OS_BR 0 /* 96 */ /* Minimum datarate for offset, [kbit/s per channel] */

#define MASK_ADD 2.0 /* Addition of maskers [dB] */
#define QUIET_ADD 3.0 /* Addition of masker and threshold in quiet [dB] */

/* *****
*
* 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 z, dz, spl, vf, tmps;

  for (k=1; k<sub_size; k++) /* Target frequencies */

```



```

{
    ltg[k].x = DBMIN;
    t = *tone;      /* calculate individual masking threshold */
    while(t != LAST) /* for tonal components, t, to find LTG */
    {
        z = ltg[power[t].map].bark; /* critical band rate of masker */
        dz = ltg[k].bark - z;      /* distance of bark value*/
        spl = power[t].x;          /* sound pressure level of masker */

        if (dz >= LOW_LIM_2 && dz < UP_LIM_2)
        {
            tmps = spl + AV_TONAL_K + AV_TONAL_DZ * z;

            /* masking function for lower & upper slopes */
            if (LOW_LIM_2 <= dz && dz < LOW_LIM_1)
            {
                if (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1 > LOW_DZ_MIN_1)
                    vf = LOW_DZ_2 * (dz - LOW_LIM_1) +
                        (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1) * LOW_LIM_1;
                else
                    vf = LOW_DZ_2 * (dz - LOW_LIM_1) + LOW_DZ_MIN_1 * LOW_LIM_1;
            }
            else if (LOW_LIM_1 <= dz && dz < 0)
            {
                if (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1 > LOW_DZ_MIN_1)
                    vf = (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1) * dz;
                else
                    vf = LOW_DZ_MIN_1 * dz;
            }
            else if (0 <= dz && dz < UP_LIM_1)
                vf = (UP_DZ_1 * dz);
            else if (UP_LIM_1 <= dz && dz < UP_LIM_2)
                vf = (dz - UP_LIM_1) * (UP_DZ_2 - UP_SPL_2 * spl) +
                    UP_DZ_1 * UP_LIM_1;
            tmps += vf;
            ltg[k].x = non_lin_add(ltg[k].x, tmps, MASK_ADD);
        }
        t = power[t].next;
    } /* while */

    t = *noise;      /* calculate individual masking threshold */
    while(t != LAST) /* for non-tonal components, t, to find LTG */
    {
        z = ltg[power[t].map].bark; /* critical band rate of masker */
        dz = ltg[k].bark - z;      /* distance of bark value*/
        spl = power[t].x;          /* sound pressure level of masker */

        if (dz >= LOW_LIM_2 && dz < UP_LIM_2)
        {
            tmps = spl + AV_NOISE_K + AV_NOISE_DZ * z;

            /* masking function for lower & upper slopes */
            if (LOW_LIM_2 <= dz && dz < LOW_LIM_1)
            {
                if (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1 > LOW_DZ_MIN_1)
                    vf = LOW_DZ_2 * (dz - LOW_LIM_1) +
                        (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1) * LOW_LIM_1;
                else
                    vf = LOW_DZ_2 * (dz - LOW_LIM_1) + LOW_DZ_MIN_1 * LOW_LIM_1;
            }
            else if (LOW_LIM_1 <= dz && dz < 0)
            {
                if (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1 > LOW_DZ_MIN_1)
                    vf = (LOW_DZ_SPL_1 * spl + LOW_DZ_K_1) * dz;
            }
        }
    }
}

```

```

        else
            vf = LOW_DZ_MIN_1 * dz;
        else if (0<=dz && dz<UP_LIM_1)
            vf = (UP_DZ_1 * dz);
        else if (UP_LIM_1<=dz && dz<UP_LIM_2)
            vf = (dz - UP_LIM_1) * (UP_DZ_2 - UP_SPL_2 * spl) +
                UP_DZ_1 * UP_LIM_1;
        tmps += vf;
        ltg[k].x = non_lin_add(ltg[k].x, tmps, MASK_ADD);
    }
    t = power[t].next;
}

if (bit_rate < H_THR_OS_BR)
    ltg[k].x = non_lin_add(ltg[k].hear, ltg[k].x, QUIET_ADD);
else
    ltg[k].x = non_lin_add(ltg[k].hear + H_THR_OFFSET, ltg[k].x, QUIET_ADD);

} /* for */
fflush(stderr);
}

```

```

/* -----
non_lin_add
A flexible addition function for levels.
Input: a,b: the levels to be added.
        c: the number of dB increase when a and b are equal.
Common values for c are 3.01 (power addition)
        and 6.02 (voltage addition).
10.0/(10*log10(2)) = 3.3219
Function added 1993-04-14 Soren H. Nielsen
----- */

```

```

double non_lin_add(a, b, c)
double a, b, c;
{
    c *= 3.3219;
    a = pow(10.0, a/c); b = pow(10.0, b/c);
    return(c*log10(a+b));
}

```

```

/*****
/*
/*      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, smr, spike, scale, sblimit, l, m)
double ltmin[SBLIMIT], smr[SBLIMIT], spike[SBLIMIT], scale[SBLIMIT];
int sblimit;
int l, m;
{
    int i,j;
    double max;

    for(i = 1; i < m; 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 */
        smr[i] = max;
    }
}

/*****
/*
/*      This procedure calls all the necessary functions to
/* complete the psychoacoustic analysis.
/*
/*****/

#define PRINTOUT      0

void II_Psycho_One(buffer, scale, ltmin, fr_ps, smr, spiki,aiff)
double /*far*/ buffer[7][1152];
double /*far*/ scale[7][SBLIMIT], ltmin[7][SBLIMIT];
frame_params *fr_ps;
double smr[7][SBLIMIT];
double spiki[7][SBLIMIT];
int aiff;
{
    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[7] = {256,256, 256, 256, 256, 256, 256}; /*7 channels! 8/10/93, SR*/
double *sample;
DSBL *spike;
static D1408 *fft_buf;
static mask_ptr /*far*/ power;
static g_ptr /*far*/ ltg;
int j, l, z, q;
static D1408 *fft_buf_hlp;
static int off_hlp = 256;

sample = (double *) mem_alloc(sizeof(DFFT), "sample");
spike = (DSBL *) mem_alloc(sizeof(D7SBL), "spike");

/* call functions for critical boundaries, freq. */
if(!init){ /* bands, bark values, and mapping */
    fft_buf = (D1408 *) mem_alloc((long) sizeof(D1408) * 7, "fft_buf");/*changed 5 to 7 for matricing
8/10/93,SR*/
    fft_buf_hlp = (D1408 *) mem_alloc((long) sizeof(D1408) * 1, "fft_buf_hlp");/*for bandwidth-limited
center!!SR*/
    power = (mask_ptr) 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] = fft_buf[2][i] =
                                fft_buf[3][i] = fft_buf[4][i] =
                                fft_buf[5][i] = fft_buf[6][i] = 0;
    /* There are some more fft_bufs because of 7 channels 8/10/93 SR*/
    init = 1;
}

if(aiff != 1)
{
    j = 0; l = 2;
}
else
{
    if(fr_ps->header->center == 1)
    {
        j = 0;
        l = 7;
    }

    if(fr_ps->header->center == 1)
    {
        for(i = 0; i < 1408; i++)
            fft_buf_hlp[0][i] = fft_buf[2][i];
        off_hlp = off[2];
    }
}

for(k = j; k < l; k++){

```

```

if(fr_ps->header->center == 1)
{
    /* 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-*/

if(fr_ps->header->center == 3)
{
    /* set center to 0, 9/2/93,SR*/
    for(z = 184; z < 512; z++)
        power[z].x = -103.670;
}
II_pick_max(power, &spike[k][0]);
    /*tonal sinusoidals, */
#ifdef PRINTOUT
    fprintf(stderr, "\nChannel %d", k);
    fprintf(stderr, "\nSignal value per subband, from the FFT:\n");
    for (i=0; i<sblimit; i++)
        fprintf(stderr, "%5.1f dB ", spike[k][i]);
    fprintf(stderr, "\nMax. signal peak per subband, SCF SPL:\n");
    for(i=0;i<sblimit;i++) /* from [II_smr] determine the SCF SPL */
        fprintf(stderr, "%5.1f dB ", 20 * log10(scale[k][i] * 32768));
    fflush(stderr);
#endif

    II_tonal_label(power, &tone); /*reduce noise & tonal */
    noise_label(power, &noise, ltg);
    /*components, find */

#ifdef PRINTOUT
    fprintf(stderr, "\nMaskers before sorting, FFT based levels:\n");
    for (i=0; i<511; i++)
    {
        if ((power[i].type == NOISE) && (power[i].x > -200))
            fprintf(stderr, "N:%3u %5.1f dB ", i, power[i].x);
        if ((power[i].type == TONE) && (power[i].x > -200))
            fprintf(stderr, "T:%3u %5.1f dB ", i, power[i].x);
    }
    fprintf(stderr, "tone = %d noise = %d \n", tone, noise);
    fflush(stderr);
#endif

    subsampling(power, ltg, &tone, &noise); /*global & minimal */

```

```

#if PRINTOUT
    fprintf(stderr, "\nMaskers after sorting:\n");
    for (i=0; i<511; i++)
    {
        if ((power[i].type == NOISE) && (power[i].x > -200))
            fprintf(stderr, "N:%3u %5.1f dB ", i, power[i].x);
        if ((power[i].type == TONE) && (power[i].x > -200))
            fprintf(stderr, "T:%3u %5.1f dB ", i, power[i].x);
    }
    fflush(stderr);
#endif

    threshold(power, ltg, &tone, &noise, /*threshold, and sgnl- */
    bitrate[info->lay-1][info->bitrate_index]/stereo); /*to-mask ratio*/
    /* fprintf(stderr, "sblimit : %d\n", sblimit); fflush(stderr); */
    II_minimum_mask(ltg, &ltmin[k][0], sblimit);

#if PRINTOUT
    fprintf(stderr, "\nMinimum masking threshold:\n");
    for (i=0; i<sblimit; i++)
        fprintf(stderr, "%5.1f dB ", ltmin[k][i]);
    fflush(stderr);
#endif

    for(i = 0; i < SBLIMIT; i++)
        spiki[k][i] = spike[k][i];

    i = 0; q = sblimit;
    II_smr(&ltmin[k][0], &smr[k][0], &spike[k][0], &scale[k][0], sblimit, i, q);

}
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_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, NV2, NM1;
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;
    NV2 = FFT_SIZE/2 >> 1;
    NM1 = FFT_SIZE/2 - 1;
    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] = 1,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 calculation corrected with a factor 4, both positive
       and negative frequencies exist, 1992-11-06 shn */
    power[i].x = 10 * log10(energy[i]*4) + 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/2-1)))/(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.

```



```

/*
/*****

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;
}

/*****
/*
/*      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<2 || first>249) run = 0; /* otherwise k+/-j will be out of bounds*/
        else if(first<62) run = 2; /* components in layer II, which */
        else if(first<126) 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,j;
    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)
long /*far*/ buffer[7][1152];
double /*far*/ scale[7][SBLIMIT], ltmin[7][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

### Patent statements

(This annex does not form an integral part of this Recommendation | International Standard)

The user's attention is called to the possibility that, for some of the processes specified in this part of ISO/IEC 13818, conformance with this specification may require use of an invention covered by patent rights.

By publication of this part of ISO/IEC 13818, 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 undertaken to file with the Information Technology Task Force (ITTF) a statement of willingness to grant a license under such rights that they hold on reasonable and non-discriminatory terms and conditions to applicants desiring to obtain such a license.

Information regarding such patents can be obtained from the following organisations.

The table summarises the formal patent statements received and indicates the parts of the standard to which the statement applies. The list includes all organisations that have submitted informal patent statements. However, if no "X" is present, no formal patent statement has yet been received from that organisation.

<b>Company</b>	<b>V</b>	<b>A</b>	<b>S</b>
AT&T	X	X	X
BBC Research Department			
Bellcore	X		
Belgian Science Policy Office	X		
BOSCH	X	X	X
CCETT			
CSELT	X		
David Sarnoff Research Center	X	X	X
Deutsche Thomson-Brandt GmbH	X	X	X
France Telecom CNET			
Fraunhofer Gesellschaft		X	X
GC Technology Corporation	X	X	X
General Instruments			
Goldstar			
Hitachi, Ltd.			
International Business Machines Corporation	X	X	X
IRT		X	
KDD	X		
Massachusetts Institute of Technology	X	X	X
Matsushita Electric Industrial Co., Ltd.	X	X	X
Mitsubishi Electric Corporation			
National Transcommunications Limited			
NEC Corporation		X	
Nippon Hoso Kyokai	X		
continued...			

<b>Company</b>	<b>V</b>	<b>A</b>	<b>S</b>
Nippon Telegraph and Telephone	X		
Nokia Research Center	X		
Norwegian Telecom Research	X		
Philips Consumer Electronics	X	X	X
OKI			
Qualcomm Incorporated	X		
Royal PTT Nederland N.V., PTT Research (NL)	X	X	X
Samsung Electronics			
Scientific Atlanta	X	X	X
Siemens AG	X		
Sharp Corporation			
Sony Corporation			
Texas Instruments			
Thomson Consumer Electronics			
Toshiba Corporation	X		
TV/Com	X	X	X
Victor Company of Japan Limited			

## **Annex E**

### **Bibliography**

(This annex does not form an integral part of this Recommendation | International Standard)

- 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 See the Normative Reference for ITU-R Rec 601 (formerly CCIR Rec 601)
- 5 See the Normative Reference for IEC Standard Publication 461
- 6 See the Normative Reference for ITU-T Rec. H.261
- 7 See the Normative reference for IEEE Standard Specification P1180-1990
- 8 ISO/IEC 10918-1 | ITU-T T.81 (JPEG)
- 9 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
- 10 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.
- 11 C. Gonzales and E. Viscito, "Flexibly scalable digital video coding". Image Communications, Vol. 5, Nos. 1-2, February 1993
- 12 A.W.Johnson, T.Sikora and T.K. Tan, "Filters for Drift Reduction in Frequency Scalable Video Coding Schemes" <Transmitted for publication to Electronic Letters.>
- 13 R.Mokry and D.Anastassiou, "Minimal Error Drift in Frequency Scalability for Motion-Compensated DCT Coding". IEEE Transactions on Circuits and Systems for Video Technology, <accepted for publication>
- 14 K.N. Ngan, J. Arnold, T. Sikora, T.K. Tan and A.W. Johnson. "Frequency Scalability Experiments for MPEG-2 Standard". Asia-Pacific Conference on Communications, Korea, August 1993.
- 15 T. Sikora, T.K. Tan and K.N. Ngan, "A Performance Comparison of Frequency Domain Pyramid Scalable Coding Schemes Within the MPEG Framework". Proc. PCS, Picture Coding Symposium, Lausanne, pp. 16.1 - 16.2, Switzerland March 1993.
- 16 Masahiro Iwahashi, "Motion Compensation Technique for 2:1 Scaled-down Moving Pictures". 8-14, Picture Coding Symposium '93.
- 17 Sikora, T. and Pang, K., "Experiments with Optimal Block-Overlapping Filters for Cell Loss Concealment in Packet Video", Proc. IEEE Visual Signal Processing and Communications Workshop, Melbourne, 21-22 Sept. 1993, pp. 247-250.
- 18 A. Puri "Video Coding Using the MPEG-2 Compression Standard", <to appear> Proc SPIE Visual Communications and Image Proc '93 Boston MA November, 1993.
- 19 A. Puri and A. Wong "Spatial Domain Resolution Scalable Video Coding", <to appear> Proc SPIE Visual Communications and Image Proc '93 Boston MA November, 1993.