SPIRIT DSP
Embedded Voice Experience

SPIRIT MP3 Encoder
User's Guide

Version 1.3

October, 2010

Copyright Information

Read this first

The document describes MP3 encoder software modules and detailed software interface definitions, contains flow-charts and testing procedures description. Document contains examples of the encoder integration and usage.

To read this document you are not required to have any special knowledge. However, prior experience with C-programming is desirable.

The MP3 encoder software performs the encoding of PCM audio data into MP3 compressed bitstream.

For more information about other SPIRIT codecs visit www.spiritDSP.com

SPIRIT reserves the right to make changes to its products to improve the products' technical characteristics without any notice. Customers are advised to obtain the latest version of relevant information to verify that the data is up-to-date before placing the orders.

SPIRIT warrants performance of its products to current specifications in accordance with SPIRIT's standard warranty. Testing and other quality control techniques are utilized to the extent deemed necessary to support this warranty.

Contents

1. INTRODUCTION 5
1.1. Purpose 5
1.2. Product overview 5
1.2.1. Supported features 5
1.2.2. Excluded features 5
1.3. Related products 5
1.4. Definitions, acronyms and abbreviations 7
2. FUNCTIONAL DESCRIPTION 8
2.1. MP3 Introduction. 8
2.2. MP3 format history 8
2.3. Algorithm overview 9
2.4. MP3 bitstream overview 10
2.5. Frame header format 11
2.6. Features and recommendations on encoder usage 16
2.6.1. Audio mass storage 16
2.6.2. Mobile messaging with audio attachments 16
2.6.3. Wireless audio data delivery via Bluetooth™ 16
3. SOFTWARE IMPLEMENTATION INTERFACE 18
3.1. Overview of SPIRIT's MP3 encoder interface 18
3.2. Data flow 18
3.3. Integration flow 18
3.4. Structure description 19
3.4.1. TMP3E_persist structure 19
3.4.2. TMP3E_scratch structure 19
3.5. Interface function description 20
3.5.1. MP3E__Init function 20
3.5.2. MP3E__Encode576 function 20
3.6. Testing and verification 22
3.6.1. Test strategy 22
3.6.2. Coding Quality and Testing Procedure. 22
3.6.3. Coding delay. 23
3.6.4. Technical characteristics estimation. 24
4. DELIVERY PACKAGE DESCRIPTION 25
4.1. Directory tree description 25
4.2. Test application source files 25
4.3. Scenario of testing 25
5. APPENDIXES 26
5.1. Call sequence example for the encoder integration 26

1. Introduction

1.1. Purpose

The purpose of this document is to describe API, integration process and test procedures of MP3 encoder (referred as MP3).

1.2. Product overview

The MP3 encoder package **(ver. 1.0)** contains the libraries of the encoder object modules, interface header and source files, a source C-code of test software (test environment) and a set of test vectors (for detailed information, see section 4).

Functionally, encoder software is multi-channel. One instance of encoder supports up to 2 audio channels (stereo encoding).

Encoder performs coding of audio signals, which are sampled at one of the standard MP3 sampling rates (ranging from 8 to 48 kHz) with 16-bit PCM (stored in the little-endian format), and forms a coded bitstream at a given rate which varies from 8 to 320 kbps.

Audio encoding operates independently on the "frame by frame" basis. Frame length is 1152 audio samples for MPEG-1 sampling rates and 576 samples for MPEG-2 and MPEG-2.5 sampling rates.

Specification of the MP3 encoder software product is represented in Table 1. For resource requirements please see readme.txt file.

Table 1. Performance

Algorithm	Encoder bitrate (kbps)	Audio quality (ODG)	Frame size (samples)	Algorithmic delay (samples)	Sampling rate (kHz)	Input signal format	Output bitstream format
MPEG-1 MP3 LC	32 – 320	See 3.6.2	1152	2367 – 11583	32 – 48	Linear 16-bit PCM	MP3
MPEG-2 & MPEG- 2.5 MP3	8 – 160		576	1791 – 6399	8 – 24		

1.2.1. Supported features

- MPEG-1, 2 or 2.5 formats.
- Layer 3 encoding.
- Mono or stereo input streams.
- Fixed bitrate encoding
- Free-format bitrate encoding

1.2.2. Excluded features

- No support for Layer 1 or Layer 2 encoding
- No Variable Bit Rate (VBR) mode encoding
- No CRC encoding

1.3. Related products

The encoder object can be efficiently used in conjunction with other products of SPIRIT:

MPEG-4 AAC Codec,

Dynamic Range Control,

Sample Rate Converter,
Automatic Gain Control,
Noise Cancellation,
Voice Activity Detector.

Since data format is very common, it can be easily interfaced with other Third Party products

1.4. Definitions, acronyms and abbreviations

For the purposes of this document, the following definitions, acronyms and abbreviations are used:

Table 2. Abbreviations

Name	Description
API	Application Programming Interface
CBR	Constant Bit Rate
CCS	Code Composer Studio
CRC	Cyclic Redundancy Check
DSK	DSP Starter Kit
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MCPS	Million Cycles Per Second
MDCT	Modified Discrete Cosine Transform
MP3	MPEG Layer-3 audio
MPEG	Motion Picture Experts Group
ODG	Objective Difference Grade
OS	Operating System
PAM	Psycho-Acoustic Model
PQMF	Pseudo Quadrature Mirror Filterbank
PC	Personal Computer
PCM	Pulse Code Modulation
XDAIS	eXpressDSP Interoperability Standard
XDAS	eXpressDSP Algorithm Standard

2. Functional Description

2.1. MP3 Introduction.

MP3 refers to the MPEG Layer 3 audio compression scheme that shrinks audio files with only a small sacrifice in sound quality. MP3 files can be compressed at different rates, but the more they are shrunk, the worse the sound quality. A standard MP3 compression is at a 10:1 ratio, and yields a file that is about 4 MB for a three-minute track.

The MPEG audio compression algorithm was developed by the Motion Picture Experts Group (MPEG), as a part of the International Organization for Standardization (ISO) standard for the high fidelity compression of digital audio. The MPEG-1 audio compression standard is one part of a multiple part standard that addresses the compression of video (ISO/IEC 11172-2), the compression of audio (ISO/IEC 11172-3), and the synchronization of the audio, video, and related data streams (ISO/IEC 11172-1). While the MPEG audio compression algorithm is lossy, it can often provide "transparent", perceptually lossless, compression.

The ISO/IEC-11172-3 standard specifies three similar formats, called “layers”, for the audio compression. All layers implements lossy compression of the digital audio for sampling rates 32, 44.1 and 48 KHz using sub-band coding with account for psychoacoustics principles. The main idea of these algorithms is to shape quantization noise in the frequency domain, so that it becomes undetectable by the average listener.

The further evolution of the standard is the MPEG-2 audio standard (ISO/IEC 13818-3). The MPEG-2 extends MPEG-1 for the sampling rates 16, 22.05 and 24 KHz. Also MPEG-2 introduces support for the multi-channel audio, however, this multi-channel extension is not considered in this document.

Also the unofficial extension of the Layer-3 audio for lower sampling frequencies (8, 11.025 and 12 KHz), called “MPEG 2.5 audio” is exists.

The widespread term “MP3” denotes the layer-3 audio (regardless of MPEG version), however, sometimes “MP3” used to denote all range of the MPEG audio algorithms. In this document we will use “MP3” with regard to all MPEG audio versions/layers.

2.2. MP3 format history

- 1987 the Fraunhofer Institut in Erlangen, Germany, started to work on perceptual audio coding in the framework of the EUREKA project EU147, Digital Audio Broadcasting (DAB).
- 1988 MPEG itself established, its full title Moving Picture Experts Group, not an organization in itself, but a subcommittee of the ISO/IEC (International Standards Organization/International Electrotechnical Commission).
- 1989 Fraunhofer Institut received a patent for MP3 in Germany.
- 1992 Fraunhofer’s algorithm was integrated in the emergent MPEG-1 standard.
- 1993 MPEG-1 (ISO/IEC 11172-3:1993) standard published.
- 1997 Since 1997, MP3 format was supported by several open-source projects
- 1998 MPEG-2 (ISO/IEC 13818-3:1998) standard published. The key difference with MPEG-1 standard is support for lower sampling frequencies (16-24 KHz). The original MPEG-1 supports frequencies from 32 to 48 KHz.
- 1998 WinAMP (free music player for Windows) appears.
- 1998 first portable MP3 player, Diamond Multimedia’s Rio 300 released
- 1999 The mp3 music distribution over the Internet becomes very popular. Many sites have emerged, allowing free music downloads. In 1999 the famous Napster (recently closed by RIAA lawsuit) appeared.

Currently, the MP3 format still remains the de-facto standard for audio compression. The key points of its success are:

MP3 format provides good audio quality at high compression ratios (near-CD-quality for 10x compression).

MP3 appeared at the “right time”, just as the demands for audio distribution over the Internet have risen.

Due to its “open” nature, MP3 has support from open-source community. As a result, numerous free encoders and players (WinAmp, MusicMatch Jukebox, etc.) are available.

2.3. Algorithm overview

The MP3 encoder consists of several components which process input PCM data stream as depicted in Figure 1

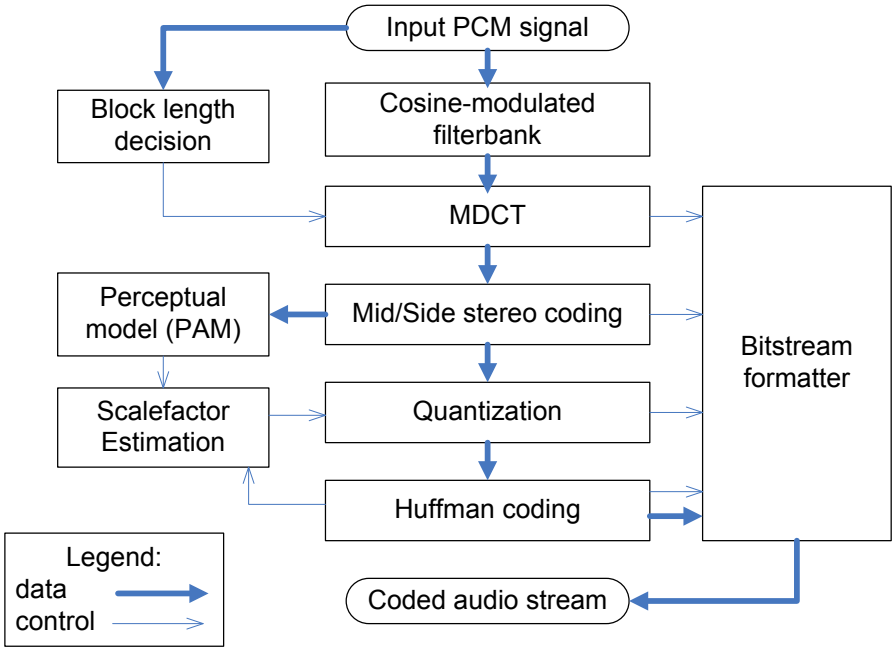

Figure 1. Block diagram of MP3 encoder

- **Blcok length decision** component switches between long and short MDCT window lengths depending on the nature of input signal (transient of stationary). The code operates on the input time-domain data, without converting the signal to frequency domain thereby significantly mitigates computational overhead while providing good transient conditions start/stop detection efficiency.
- **Cosine-modulated filterbank** component decomposes the time-domain signal into 32 uniform frequency bands. Signal in each frequency bands downsampled (critically-sampled filtrbank).
- **MDCT** component provides enhanced frequency resolution using a sine-windowed MDCT transform. Windowing is done as follows:

$$w_n = s_n \cdot \sin\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)\right), \qquad 0 \le n < N$$

where s_n is an audio sample value and N can be 36 (long window), 12 (short window). The MDCT spectral coefficients are defined as follows:

$$x_k = 2\sum_{n=0}^{N-1} w_n \cos\left(\frac{2\pi}{N}\left(n+\frac{1}{2}\left(\frac{N}{2}+1\right)\right)\left(k+\frac{1}{2}\right)\right), \quad 0 \le k < \frac{N}{2}$$

- **Mid/Side stereo coding** tool is used for stereo signal to reduce the bits required to store spectral coefficients for stereo channels by coding them as mid/side: M=(L+R)/2, S=(L-R)/2. The decision to code the coefficients as left/right or mid/side is made using the signal entropy. This method provides a good coding efficiency while requiring much less computational resources as compared to FFT-based methods.
- **Perceptual model (PAM)** tool accepts the mid/side coded MDCT coefficients, regularizes MDCT spectrum and estimates the noise masking threshold for the current frame of audio data. The usage of regularized MDCT coefficients in this tool instead of FFT spectrum allows to achieve high performance of the coding while keeping good perceived audio quality.

- **Scalefactor estimation** component uses the noise masking threshold estimated by the PAM tool to derive scalefactors for the subsequent quantization which introduce perceptually-minimal quantization noise into the signal. The estimation is based upon the statistical model for the quantization noise. This method allows to omit the costly distortion control loop in the rate-distortion control process. With this optimization the rate-distortion control process is reduced to one common scalefactor search loop. The adaptive search technique provides fast convergence: the worst average loop iterations number observed is less than two.
- **Quantization** tool quantizes MDCT coefficients using the following formula:

$$q_k = \mathrm{sign}(x_k) \cdot \mathrm{int}\left(|x_k|^{3/4} \cdot 2^{-\frac{3}{16}(sf - common\_sf)} + 0.4054 \right)$$

  where *sf* is a scalefactor for the spectral band to which coefficient x_k belongs, *common_sf* is a scalefactor shift value common for all bands.
- **Huffman coding** tool performs a compression of the quantized MDCT coefficients and scalefactors using Huffman algorithm.
- **Bitstream formatter** tool receives data and control output from other components, and produces the coded audio bitstream.

2.4. MP3 bitstream overview

The MP3 compressed bit stream can have one of several predefined fixed bit rates. The ranges for allowable bit rates are shown in the Table 3. Depending on the audio sampling rate, this translates to compression factors ranging from 2.3 to 48. It is allowed to encode different parts of the stream with different bit rates, producing so-called “Variable Bit Rate” (VBR) stream. In addition, the standard provides a "free-format” bit rate mode to support fixed bit rates other than the predefined rates. The coded bit stream supports an optional Cyclic Redundancy Check (CRC) error detection code.

Table 3. Bitrates and sampling rates for the MP3 audio

MPEG version	Sampling rates (kHz)	Layer-3 Bitrate, kbps
MPEG-1 (ISO/IEC 11172-3)	32, 44.1, 48	32-320
MPEG-2 (ISO/IEC 13818-3)	16, 22.05, 24	8-160
MPEG-2.5 (Unofficial extension)	8, 11.025, 12	8-160

The MPEG compressor encodes audio by the fixed size blocks. The block, called “frame” encodes predefined number of PCM samples (see Table 4). The decoding must start from the beginning of the frame, however it is possible to decode only a part of frame. The minimum decodable unit size is shown in the

Table 4. Frame sizes for the MPEG Layer-3 audio

MPEG version	Frame size, samples	Minimum decodable unit, samples
MPEG-1 (ISO/IEC 11172-3)	1152	576
MPEG-2(ISO/IEC 13818-3)	576	576
MPEG-2.5 (Unofficial extension)	576	576

The frame format is shown on Figure 2. The frame starts with a 32-bit frame header, which is used for stream synchronization and encodes basic audio information, such as sample rate, number of channels, bitrate, etc. The 16-bit CRC field is optional. Note that CRC is calculated not the entire frame, but only for the last 16 bits of the frame header and side info.

Layer-3	Header 32-bits	CRC 0-16 bits	Side info 72-256 bits	Scalefactors and Huffman-encoded data (“part 2&3”) 1 or 2 granules * 576 samples (0-4095 bits per gr.)	Ancillary data

Figure 2. MPEG audio frame format.

There are no main file headers in MPEG audio files. An MPEG audio file is built up from a sequence of successive frames. MPEG stream cannot have any “gaps” between the frames. Frame size can be determined from the information contained in the frame header, so it is possible to determine position of the next frame header if the position of the previous one is known. The MPEG audio stream can contain ancillary data, which are used to keep the specified bit rate (i.e. frame size) when the bit reservoir cannot be increased or is not used.

In the Variable Bit Rate (VBR) stream, the frame size can differ across the stream. There are no special “flags” etc. to figure out whether the MPEG audio stream is VBR or not. The decoder should always assume that frame sizes may differ from frame to frame.

A rarely used feature of MPEG audio is the “Free format” mode. In this mode the frames in the stream can have arbitrary bitrate (possibly, different from the fixed set of bitrates specified in the standard). The free-format frame size cannot be calculated from the information contained in the frame header. In order to support the free-format mode, the decoder must figure out free-format frame size by searching several consecutive frame headers. The VBR mode is inadmissible when using free-format mode, and free-format frames cannot be mixed with fixed-bitrate frames.

In case of Layer I or Layer II, frames are totally independent entities: each frame contains all information required for its decoding, so the decoding process can start right after first frame header is found in the stream. However, in case of Layer III, frames are not always independent. The structure of the Layer-3 bit stream is shown on Figure 3. The frame data can begin before the frame header (“bit reservoir”). The reference to the frame data (“main_data_begin”) is stored in the side info field of the frame. The bit reservoir technique allows to code the “hard to encode” audio fragments more efficiently. The size of the bit reservoir is limited to 511 bytes for MPEG1 and 255 bytes for MPEG2 and MPEG2.5. Note that in the extreme case the MPEG-2 frame data size can be only 1 byte (when decoding 8 kbps @ 24 KHz audio with CRC protection), so in this case up to 255 frames may be required before being able to decode one frame.

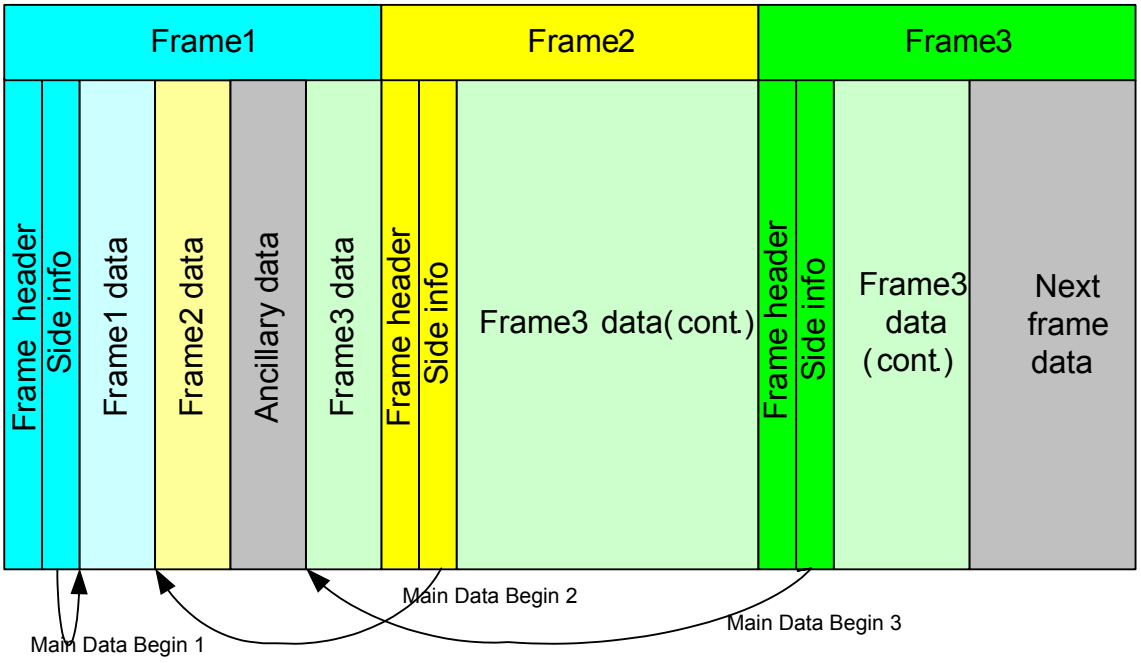

Figure 3. MP3 stream structure.

2.5. Frame header format

The first 32 bits (4 byte) of the audio frame are header information. Below, detailed description of these bits is given:

Table 5 Header bitstream syntax

Syntax	No. of bits	Hex bitmask
header(){		
Syncword	11	FF E0 00 00
Idex	1	00 10 00 00
ID	1	00 08 00 00
Layer	2	00 06 00 00
protection_bit	1	00 01 00 00
bitrate_index	4	00 00 F0 00
sampling_frequency	2	00 00 0C 00
padding_bit	1	00 00 02 00
private_bit	1	00 00 01 00
Mode	2	00 00 00 C0
mode_extension	2	00 00 00 30
Copyright	1	00 00 00 08
original/copy	1	00 00 00 04
Emphasis	2	00 00 00 03
}		

syncword (11 bits): The following bitstring: '1111 1111 111'.

IDex (1 bit): One bit to indicate the extended ID of the algorithm. It equals '1' for MPEG1 as well as for MPEG2 and '0' for MPEG2.5. Note: MPEG Version 2.5 is an unofficial extension of the MPEG 2 standard. It is an extension used for very low bitrate files, allowing the use of lower sampling frequencies.

ID (1 bit): One bit to indicate the ID of the algorithm. It equals '1' for MPEG1 and '0' for MPEG2 as well as for MPEG2.5 (i. e., for the lower sampling frequencies extension).

Table 6 Selection of decoded algorithms

ID	Index	Algorithm
0	0	MPEG25
0	1	Reserved
1	0	MPEG2
1	1	MPEG1

layer (2 bits): Two bits to indicate which layer is used. For Layer 3, the bitstring is '**01**'.

Table 7 Selection of MPEG audio layer

layer	Algorithm
00	Reserved
01	Layer III
10	Layer II
11	Layer I

protection_bit (1 bit): One bit to indicate whether redundancy has been added in the audio bitstream to facilitate error detection and concealment. It equals '1' if no redundancy has been added and '0' if redundancy has been added (the 16-bit CRC field follows the header).

bitrate_index: Indicates the bitrate. The all-zero value indicates the 'free format' condition, in which any fixed bitrate, not limited to those present in the list, can be used. Fixed means that a bitstream frame contains either N or N+1 slots, depending on the value of the padding bit. The **`bitrate_index`** can be one of the values found in Table 8. It indicates the total bitrate irrespective of the mode (stereo, joint_stereo, dual_channel, single_channel).

Table 8 Selection of bitrate by bitrate_index

Bitrate index	bitrate specified, kbps					
	MPEG1	MPEG1	MPEG2	MPEG1	MPEG2	MPEG2.5
	Layer 1	Layer 2	Layer 1	Layer 3	Layer 2 & 3	Layer 3
'0000'	Free	Free	Free	Free	Free	Free
'0001'	32	32	32	32	8	8
'0010'	64	48	48	40	16	16
'0011'	96	56	56	48	24	24
'0100'	128	64	64	56	32	32
'0101'	160	80	80	64	40	40
'0110'	192	96	96	80	48	48
'0111'	224	112	112	96	56	56
'1000'	256	128	128	112	64	64
'1001'	288	160	144	128	80	forbidden
'1010'	320	192	160	160	96	forbidden
'1011'	352	224	176	192	112	forbidden
'1100'	384	256	192	224	128	forbidden
'1101'	416	320	224	256	144	forbidden
'1110'	448	384	256	320	160	forbidden
'1111'	forbidden	forbidden	forbidden	forbidden	forbidden	forbidden

Layer3 achieves variable bitrate by switching between the **`bitrate_index`** values. It can be used either to optimize storage requirements or to get any mean (interpolated) data rate by switching between adjacent **`bitrate_`**index values in the bitrate table. However, in free format, fixed bitrate is mandatory. The decoder is also not required to support bitrates higher than 320 kbps, 160 kbps or 64 kbps (depending on ID and IDex) in free format mode.

For Layer II there are some restricted combinations of bitrate and mode. The list of allowed combinations is found in the table below.

Table 9 Allowed bitrate/channel combinations for Layer-2

bitrate	single channel	stereo	intensity stereo	dual channel
free	yes	yes	yes	yes
32	yes	no	no	no
48	yes	no	no	no
56	yes	no	no	no
64	yes	yes	yes	yes
80	yes	no	no	no
96	yes	yes	yes	yes

112	yes	yes	yes	yes
128	yes	yes	yes	yes
160	yes	yes	yes	yes
192	yes	yes	yes	yes
224	no	yes	yes	yes
256	no	yes	yes	yes
320	no	yes	yes	yes
384	no	yes	yes	yes

sampling_frequency: Indicates the sampling frequency according to Table 10 below.

Table 10 Sampling frequency selection

Bits	MPEG 1	MPEG 2	MPEG 2.5
00	44.1 kHz	22.05 kHz	11.025 kHz
01	48 kHz	24 kHz	12 kHz
10	32 kHz	16 kHz	8 kHz
11	reserved	reserved	reserved

padding_bit: If this bit equals '1', the bitstream frame contains an additional slot to adjust the mean bitrate and get the exact sampling frequency, otherwise this bit will be '0'. For example: 128kbps 44.1kHz layer II uses a lot of 418 bytes long frames and a few 417 bytes long ones to get the exact 128k bitrate. For Layer I, a slot is 32 bits long, for Layer II and Layer III, it is 8 bits long. Padding is necessary with sampling frequencies of 11.025 kHz, 22.05 kHz and 44.1 kHz. Padding may also be required in free format.

private_bit: Bit for private use. This bit will not be used in the future by ISO/IEC.

mode: Indicates the mode according to Table 11. The joint stereo mode is intensity stereo and/or ms stereo.

Table 11 Stereo mode selection

Code	Stereo mode
00	Stereo
01	Joint stereo
10	Dual channel
11	Mono

mode_extension: These bits are used in joint stereo mode. For Layer-3 stream they indicate which type of joint stereo coding method is applied. The frequency ranges, over which the intensity_stereo and ms_stereo modes are applied, are implicit in the algorithm. For Layer-1 and Layer-2 streams, these two bits determine the frequency range (sub-bands) where intensity stereo is applied. There are 32 sub-bands in all.

The types of mode extensions are listed in **Table 12**.

Table 12 Selection of mode extension (only for joint stereo mode)

Code	Layer-1 and 2 Joint stereo range	Layer-3	
		Intensity stereo	Ms stereo
00	bands 4 to 31	Off	Off
01	bands 8 to 31	On	Off
10	bands 12 to 31	On	On

11	bands 16 to 31	Off	On

copyright: If this bit equals ‘0’, there is no copyright on the Layer3 bitstream. ‘1’ means that these data are copyright protected.

original/copy: This bit equals ‘0’ if the bitstream is a copy and ‘1’ if it is an original.

emphasis: Indicates the used type of emphasis. The emphasis is a technique which is sometimes used for audio transmission over band-limited (analog) channels. It pre-equalizes the audio contents by amplifying the high-frequency part of the signal. The emphasis indication is here to tell the decoder that the file must be de-emphasized, i.e., the decoder must ‘re-equalize’ the sound after a Dolby-like noise suppression. It is rarely used.

2.6. Features and recommendations on encoder usage

2.6.1. Audio mass storage

Figure 4 demonstrates an example of the encoder software usage where MP3 encoder is used to compress digital audio data (AudioCD music or microphone-recorded speech). The good quality of the compressed audio signal along with high compression ratio provides efficient utilization of the storage space (data CD/DVD, HDD, flash card).

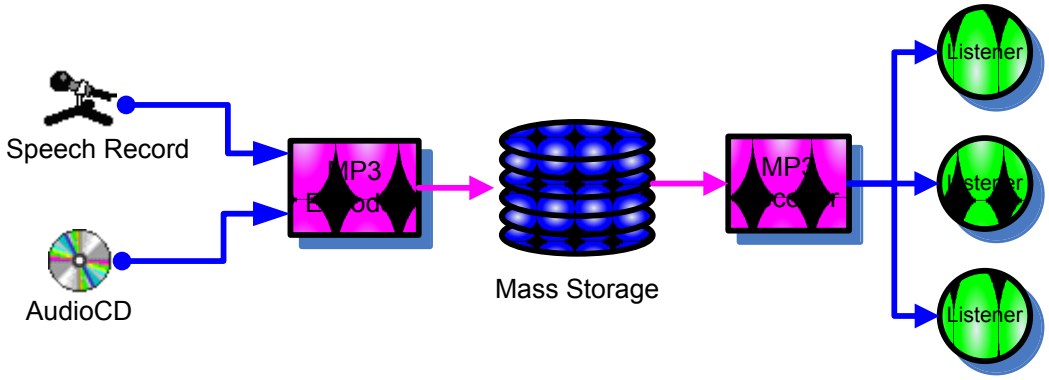

Figure 4. Audio mass storage using MP3 encoder

2.6.2. Mobile messaging with audio attachments

Figure 5 demonstrates example of the encoder software usage where encoded audio data (e.g. music tune, speech sample, recorded sound effect) is attached to MMS message. While the receiving person enjoys the good quality of attached audio sample, the small size of the encoded audio data provides efficient usage of the modile network bandwidth.

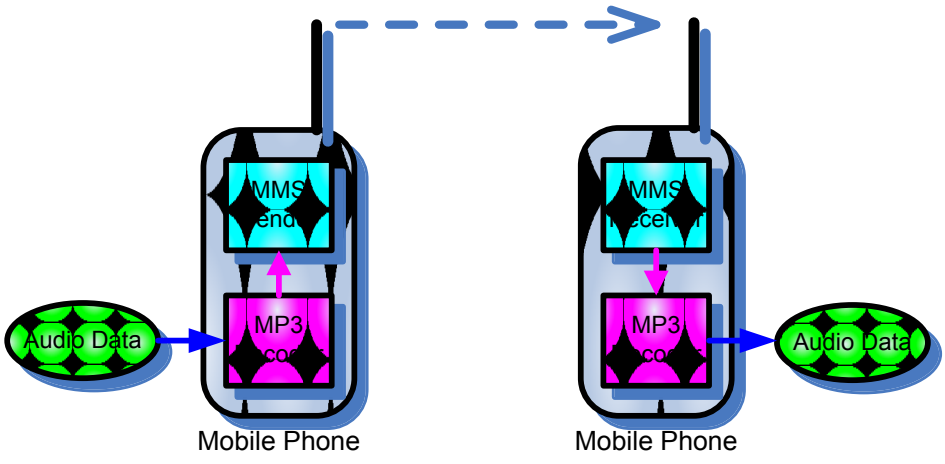

Figure 5. Mobile MMS messaging using MP3 compressed audio attachment

2.6.3. Wireless audio data delivery via Bluetooth™

Figure 6 demonstrates example of the encoder software usage where the audio data (e.g. music) intended to be delivered to the wireless headphones is compressed by MP3 encoder and transferred via Bluetooth™. The high compression ratio of MP3 encoder allows to fit good quality audio data into Bluetooth™ bandwidth. Note that the headphones have to contain MP3 decoder (not shown in the figure) which is easy to embed due to its low resource requirements.

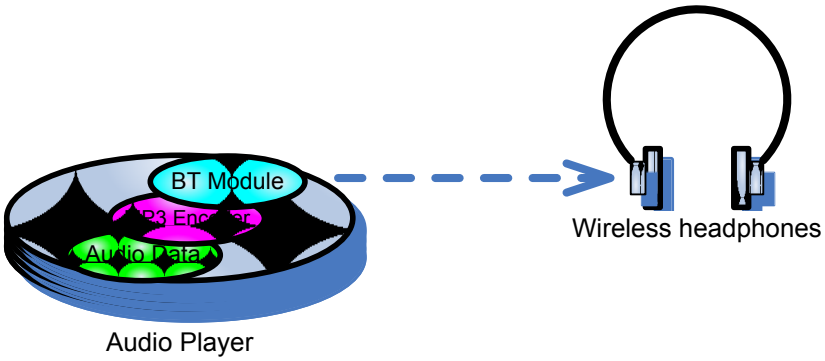

Figure 6. Wireless audio data delivery via Bluetooth™ using MP3 compression

3. Software implementation interface

3.1. Overview of SPIRIT's MP3 encoder interface

The encoder implemented using XDAIS parent-child interface. The child object implements algorithm functionality, while the parent object holds the constant data, shared among the instances of the child objects. The parent object can be considered as the Scratch RAM implementation for the constant data.

This encoder's implementation is developed in direct compliance with XDAIS rules described above. For this case, encoder software has a set of the following functions.

Table 13. Encoder interface functions

Name	Functionality	Section
`MP3E__Init`	MP3 Encoder Module Initialization	3.5.1
`MP3E__Encode576`	MP3 Data Encoding	0

The `MP3E__Init` function provides initial preparation of encoder for operation.

The `MP3E__Encode576` is a basic function, which provides audio encoding.

Prototypes of these functions are defined in "`SpiritMP3Enc.h`".

3.2. Data flow

The encoder uses the “push” model of dataflow. Application supplies fixed-size PCM data buffer to the encoder, and encoder returns to the application pointer to the coded data. The coded data buffer belongs to encoder, and must not be “de-allocated” by the application. The encoder does not change input PCM data buffer.

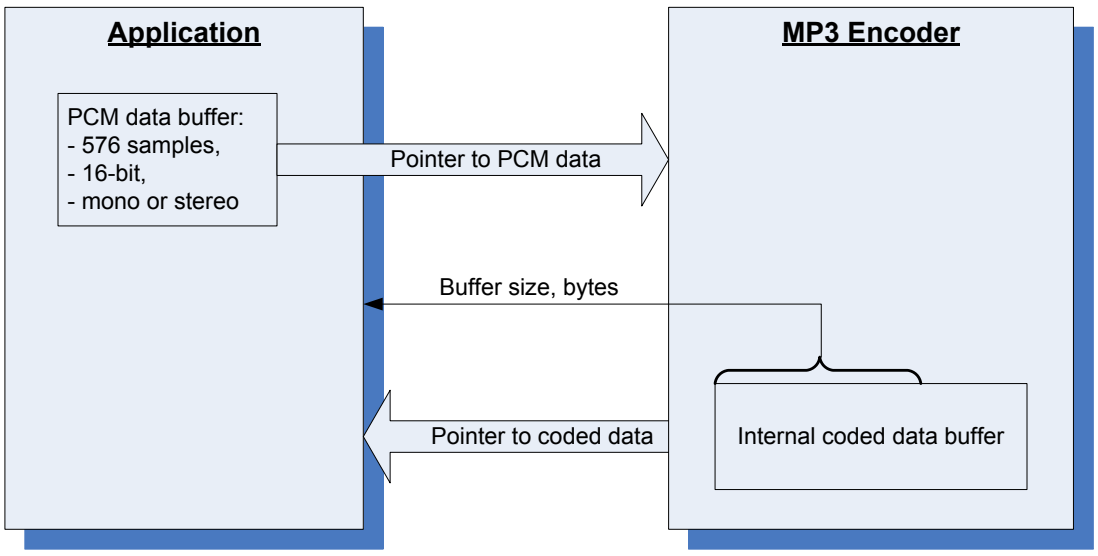

Figure 7 MP3 encoder data flow.

3.3. Integration flow

In order to integrate the MP3 encoder into a user's framework, the user should:

- initialize the encoder using `MP3E__Init` function,
- encode buffer (576 samples) using `MP3E__Encode576` function and send output data to its planned destination,

Example of the encoder's call sequence is represented in Appendix 5.1.

3.4. Structure description

This section describes the algorithm data structures. Structures content is not documented. The application should not access structures directly.

3.4.1. TMP3E_persist structure

Description

The MP3 encoder persistent data structure. The encoder does not use non-constant static or global variables so all functions are re-entrant. You can use several instances of the encoder by allocating and initializing one encoder structure per instance.

Types

The `TMP3E_persist` structure is defined in "`SpiritMP3Enc.h`" file.

3.4.2. TMP3E_scratch structure

Description

The MP3 encoder temporary data structure. The contents of this structure may be destroyed between calls to the MP3 encoder. This structure can be shared between several instances of the MP3 encoder, or may be used by the application.

Types

The `TMP3E_scratch` structure is defined in "`SpiritMP3Enc.h`" file.

3.5. Interface function description

This section describes interface functions.

3.5.1. MP3E__Init function

Description

Initialization function. It must be called before any other encoder functions. Also this function must be called before encoding of each new MP3 stream.

Call sequence

int	MP3E__Init (TMP3E_persist * pEncoder, TMP3E_scratch * pScratch, unsigned int uiSampleRateHz, int iBitrateKbps, int iCh);

Parameters

TMP3E_persist * pEncoder	Pointer to encoder state function
TMP3E_scratch * pScratch	Pointer to encoder scratch memory.
unsigned int uiSampleRateHz	PCM Sample rate, Hz See Table 10 for valid sampling rate values.
int iBitrateKbps	Coding bitrate, kbps. See Table 8 for standard bitrate values. If specified rate is not standard, free-format mode encoding is used.
int iCh	Number of audio channels: 1 for mono or 2 for stereo.
int	Returned value: non-zero if success, zero if initialization parameters are incorrect.

3.5.2. MP3E__Encode576 function

Description

Gets the frame of uniform PCM samples and encodes them into a bitstream frame.

Call sequence

unsigned char *	MP3E__Encode576 (TMP3E_persist pEncoder, const short * psPCM_left, const short * psPCM_right, int iPCM_interleave, unsigned int * puiSizeofCodedData,);

Parameters

TMP3E_persist pEncoder	Pointer to MP3 encoder object
const short * psPCM_left	Pointer to left channel input sample buffer
const short * psPCM_right	Pointer to right channel input sample buffer

`int iPCM_interleave`		Interleave factor for input buffers
`unsigned int puiSizeofCodedData`	*	Returned parameter. Number of bytes in the output buffer
`unsigned char *`		Returned parameter. Output bit stream buffer

Remarks

Function encodes 576 audio samples for 1 or two channels. Function returns pointer to the coded data and coded data size in bytes. Returned buffer belongs to encoder object, and must not be de-allocated by the application.

Function may accept stereo data in interleaved or plain format, by proper assignment of input parameters. Please see examples of proper parameters choice on Figure 8 and Figure 9

Function may not return coded data immediately after the call (due to bit reservoir usage). In this case coded buffer size will be set to 0, and value of returned pointer is not defined (in the current implementation function always returns valid pointer, but application should not rely on this assumption).

Note that for MPEG1 sampling rates 1 MP3 frame consists of two granules (576 samples in each granule), so this function will produce data every second call (normally). In case of MPEG2 and MPEG2.5 mode, function normally produces data after every call.

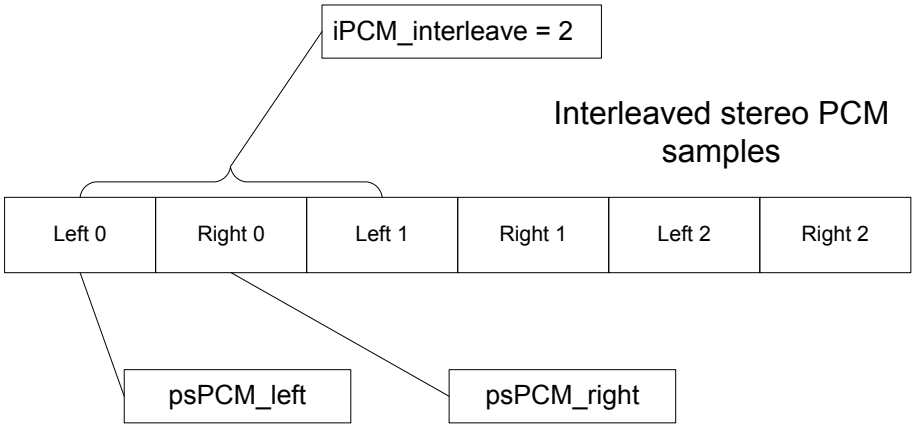

Figure 8 Input parameters for interleaved PCM data.

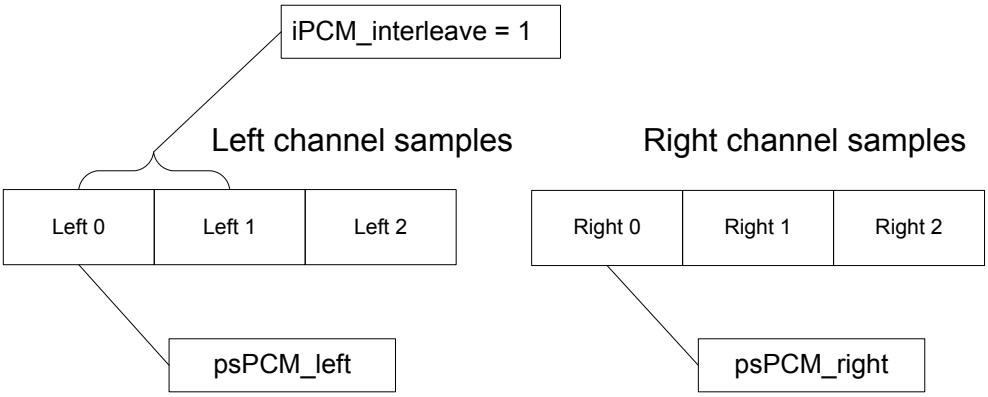

Figure 9 Input parameters for planar PCM data.

3.6. Testing and verification

3.6.1. Test strategy

Testing of the software is intended to make sure the software operates correctly under all operating modes and all specification parameters are valid.

In order to achieve this purpose, test environment is developed. The test environment includes test software and a set of test vectors along with reference vectors.

The test procedure includes

- Algorithm quality estimation
- Verification of bit-exactness of PC model with the code, running on the target platform
- Worst-case and average MIPS (MCPS) requirements estimation
- Code, constants, heap and stack memory size estimation

The complete test environment is not a part of the product. The product package includes simplified test environment, which can be referred as an integration example.

3.6.2. Coding Quality and Testing Procedure.

The Spirit MP3 encoder coding quality was estimated using objective quality measurement according to ITU-R BS.1387 recommendation, also known as PEAQ. The objective measurement method described in this Recommendation measures audio quality and outputs a value intended to correspond to perceived audio quality. The measurement method models fundamental properties of the auditory system.

The input to BS.1387 is two audio files, one is original “reference” file, and second is the same file with introduced distortion.

The BS.1387 provides several intermediate outputs, which can be used to characterize artifacts, and also provides integral quality metric, called Objective Difference Grade (ODG). The ODG normally has a negative value. It roughly corresponds to 5 levels of impairment:

- 0 - Imperceptible
- -1 - Perceptible but not annoying
- -2 - Slightly annoying
- -3 - Annoying
- -4 - Very Annoying

The BS.1387 have several serious limitations, most significant is that it was calibrated only for 48 KHz sample rate, and pretty low level of distortions (near-CD quality). However, in our opinion it is the only available tool for objective estimation of the audio quality.

For MP3 encoder testing, following procedure was used:

- Test set was chosen (“Lame” test suite). Test set includes 20 stereo files.
- The test suite was encoded and decoded using all MP3 bitrates starting from 112 kbps. The lower bitrates was not tested, since it was assumed that BS.1387 does not provide precious results for low-quality audio.
- The ODG score for all files for all tested bitrates (totally 140 measurements) is averaged to provide integral characteristic of the coding quality.

The quality of the Spirit MP3 encoder was tuned using quality procedure above. Since Spirit MP3 encoder designed for real-time embedded applications, it does not use full set of available MP3 coding tools.

The Spirit MP3 encoder uses following MP3 coding tools:

- Block switching (short blocks)
- Mid-side stereo

- Semi-adaptive Huffman coding
- Noise Shaping

Besides, the Spirit MP3 encoder implements efficient bit-allocation algorithm, which is missed in known implementations of the MP3 algorithm: the MPEG specification assumes that coder implements noise-shaping, which automatically provides proper bit-allocation, but Spirit MP3 encoder uses dedicated bit-allocation algorithm.

The following MP3 coding tools is not implemented or not used

- Mixed blocks
- Intensity stereo.

The mixed blocks are not used, since it does not provide quality gain at all.

The Intensity stereo tool was not implemented, since it mostly usable for very low bitrates, which was not tested due to limitations of the BS.1387.

The test shows that Spirit MP3 encoder provides better quality, than known state-of-the-art encoders.

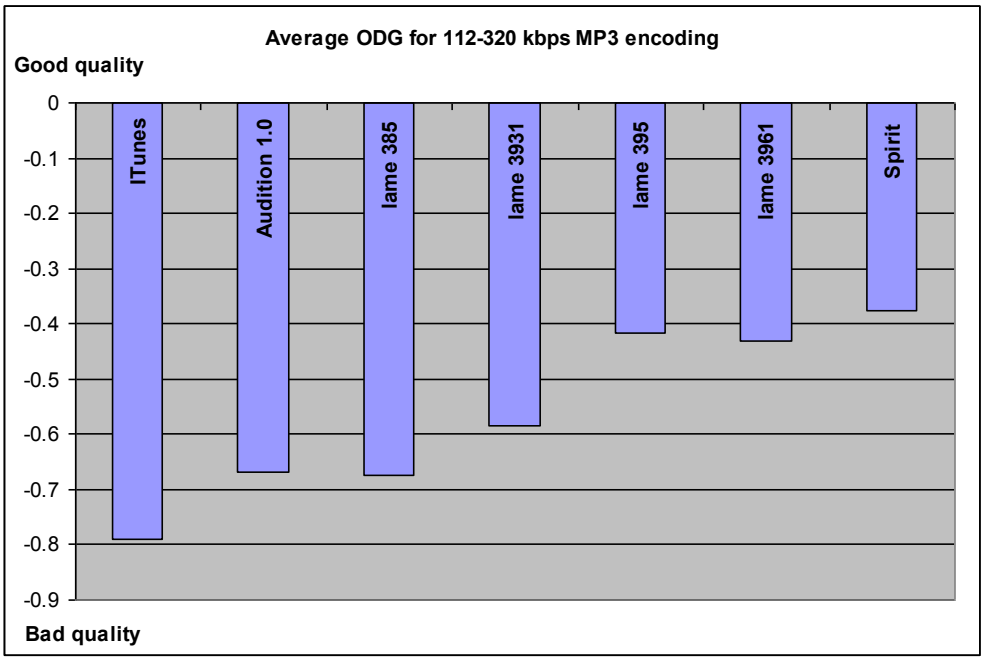

Figure 10. Encoder quality comparison.

3.6.3. Coding delay.

The coding delay is the total delay of the encoding-decoding system. The coding delay for MP3 encoder estimated analytically. The minimum coding delay is 2367 samples for MPEG-1 sampling rates and 1791 samples for MPEG-2 sampling rates. Due to bit-reservoir usage, the coding delay increases, but this delay depends both on coding bitrate and coded audio contents. The bit-reservoir delay is limited in the Spirit implementation by 8 frames.

Table 14 Coding delay estimation

Algorithm	Delay, PCM samples	
	MPEG-1	MPEG-2 & MPEG-3
Transient detector look-ahead†	128	128

† Spirit Implementation specific

PQMF filterbank	511	511
Framing delay	1152	576
MDCT delay	576	576
Bit-reservoir delay (bit-rate and contents dependent)	0-9216	0-4608
Total:	2367 – 11583	1791 – 6399

3.6.4. Technical characteristics estimation.

The MIPS requirements estimated by measuring number of cycles, required for encoding of 576 samples. To obtain reproducible results, the measurements performed with the simulator. The peak MIPS requirements defined as:

$$MIPS = \frac{NCycles}{FramseSize} \cdot SampleRateHz,$$

where *Ncycles* is the maximum observed number of cycles and *FrameSize* equals to 576 samples. The average MIPS estimated using same equation, but for the average number of cycles.

Stack and heap memory size requirements estimated when running the test application. The code and constants size requirements estimated using the map, generated by the linker.

4. Delivery package description

4.1. Directory tree description

The MP3 software package includes library, header file, test application source files and corresponding input and reference test vectors (see Table 15).

The package content is divided into several subdirectories:

Doc	The documentation files
Lib	The MP3 encoder libraries
Inc	Header files
Test	The test application source files, test projects and test files

Table 15. The package files

Directory\File	Description
Doc\MP3_Encoder_User_Guide.pdf	This document
Test*.c	Test application source files
Test\run.bat	Batch file to build and run sample application with the functional simulator
Test*.wav	Input file(s)
Test*.ref	Reference output file(s)

4.2. Test application source files

The test application source code consists of following files: `Sample.c WavReader.c WavReader.h.`

The module `Sample.c` implements WAV file encoding application.

The module `WavReader.c` is intended for low-level WAV file input operations. This code is independent of the encoder interface.

4.3. Scenario of testing

Step	Description
1. Set-up test scenario.	Specify a name of the input .wav file in the main() function of the `Sample.c`
2. Set-up test environment	Modify paths to the functional simulator in the run.bat
2. Run application with the functional simulator.	Run "run.bat" file. Specified WAV file will be encoded to MP3 file.
3. MCPS Check	Verify that number of cycles, taken by MP3ENC_apply() function do not exceed specified MIPS.
4. Memory check	To check the memory requirements of the encoder, the QualiTI tool can be used. Load the supplied algorithm definition file and run the compliance test. Compliance test generates the compliance documentation which contains detailed information of the encoder memory usage

5. Appendixes

5.1. Call sequence example for the encoder integration

The C-code shown below illustrates the typical call sequence for the encoder.

```
#include "SpiritMP3Enc.h"
#include <stdio.h>

#define MP3_FRAME_SIZE 576

int main()
{
    TMP3E_persist Enc;                  // Declare encoder instance
    TMP3E_scratch Scratch;              // Declare encoder scratch pad
    short asPCM[2*MP3_FRAME_SIZE];      // PCM buffer for 576 samples for 2 channels
    FILE *filePCM, *fileMP3;            // file I/O variables

    // coding parameters
    unsigned long ulSampleRateHz = 44100;
    int iBitrateKbps = 128;
    int iChannels = 2;
    int status;

    // Initialize encoder.
    status = MP3E__Init(&Enc, &Scratch, ulSampleRateHz, iBitrateKbps, iChannels);

    if (!status) { // initialization failed
        printf("ERROR: Invalid coding parameters!\n");
        return 0;
    }

    // open input PCM and output MP3 files
    filePCM = fopen("music.pcm", "rb");
    fileMP3 = fopen("music.mp3", "wb");
    if (!filePCM || !fileMP3) {
        printf("ERROR: problems with input/output files!\n");
        return 0;
    }

    // read PCM data and encode to MP3 in a loop
    while (fread(asPCM, sizeof(short)*iChannels*MP3_FRAME_SIZE, 1, filePCM)) {
        unsigned int uiBitBufSize;
        unsigned char *pCodedData;

        // Encode PCM data.
        pCodedData = MP3E__Encode576(
             &Enc,           // encoder
             asPCM,          // left channel
             asPCM + 1,      // right channel
             iChannels,      // PCM interleave
             &uiBitBufSize   // [OUT] coded data size
             );

        if (uiBitBufSize) {  // write mp3 data to file
            fwrite(pCodedData, sizeof(char), uiBitBufSize, fileMP3);
        }
    }

    // cleanup: no need to de-initialize encoder object.
    fclose(filePCM);
    fclose(fileMP3);
    return 0;
}
```