MPLAB® Harmony Help - Decoder Libraries

MPLAB Harmony Integrated Software Framework v1.11

Decoder Libraries Help

This section provides descriptions of the Decoder libraries that are available in MPLAB Harmony.

AAC Decoder Library

This section describes the AAC Decoder Library.

Introduction

The AAC Decoder Library is available for the PIC32MX family of microcontrollers.

Description

The AAC Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

This AAC algorithm Non-Modifiable Binary Code is designed for 80 MHz or greater PIC32MX MCUs. Keep in mind, this code requires 62 MIPS peak 34 MIPS average performance, 61 KB Flash and 12 KB RAM without frame buffer memory for operation on the PIC32MX MCU. This product is the non-modifiable binary. A modifiable source-code version of this code is also available on Microchip Direct (SW320013-2). Users are responsible for licensing for their products through Via Licensing.

Assembly Language Issue

There is an assembly language issue in the AAC Decoder Library that misaligns the stack pointer. Contrary to the MIPS programming specification, this error is benign except when the application is built for a device with a hardware Floating Point Unit (FPU), such as the PIC32MZ EF family of devices. In this case, the sub-routine context-saving operations will produce an exception and stop the application. A work around exists for this issue, which involves disabling FPU context-saving for Interrupt Service Routines (ISRs).

The work around is to convert all ISRs from:

```
void __ISR(vector,ipl) MyInterruptHandler(void)
```

and change them to:

```
void __attribute__((vector),interrupt(ipl),nomips16,no_fpu)) MyInterruptHandler(void)
```

where, `vector` and `ipl` are replaced with the actual interrupt vector and interrupt priority level.

An example of this work around can be found in the universal_audio_decoders demonstration.

So as long as no FPU operations occur in ISRs and the routines are called by ISRs, this work around will prevent the misaligned stack pointer from causing an exception. If FPU operations result from an ISR with no_fpu enabled, a general exception will result.

Using the Library

This topic describes the basic architecture of the AAC Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_aac.h`

The interface to the AAC Decoder Library is defined in the `decoder_aac.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_aac.h`.

Please refer to the What is MPLAB Harmony? section for how the AAC Decoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the AAC Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	AAC_Decoder	This is function AAC_Decoder.
	AAC_WriteWavHeader	Generates the .wav file header, if applicable.
	AACDecoder_GetFrameSize	Used to get the AAC Decoder frame size.
	AACDecoder_GetStateSize	This function allocates memory for each channel being used.
	AACDecoder_Init	Views the received point and clears the structure.
	AACDecoder_InterleaveSamples	This function interleaves left and right channel data.
	isAACdecoder_enabled	This is function isAACdecoder_enabled.

	AAC_Decode	This function decodes AAC data.
	AAC_Initialize	This is function AAC_Initialize.
	AAC_RegisterDecoderEventHandlerCallback	This is function AAC_RegisterDecoderEventHandlerCallback.
	AAC_GetSamplingFrequency	This is function AAC_GetSamplingFrequency.

b) Data Types and Constants

	Name	Description
	INTPCM	
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	WAVE	This is type WAVE.
	AAC_PROFILE	This is macro AAC_PROFILE.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.
	AACDECODER_STATE_SIZE	in Bytes
	FRAME_SIZE	in Samples
	INPUT_BUF_SIZE	in Bytes
	MAX_STACK	in Bytes
	STACK	This is macro STACK.
	AAC_FRAME_HEADER_SIZE	in Bytes
	SetReadBytesInAppData	This is type SetReadBytesInAppData.

Description

This section describes the Application Programming Interface (API) functions of the AAC Decoder Library.

Refer to each section for a detailed description.

a) General Functions

AAC_Decoder Function

File

aac.h

C

```
int16_t AAC_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t *
written);
```

Description

This is function AAC_Decoder.

AAC_WriteWavHeader Function

Generates the .wav file header, if applicable.

File

decoder_aac.h

C

```
void AAC_WriteWavHeader(void * haacDecoderInfo, WAVE * wav);
```

Returns

None.

Description

This function is called only if the output needs to be written into a .wav file. If the output is for play out or for writing into a .pcm file, this function shall not be called. In case of wave file generation, the application shall leave 44 bytes space free at the start of the file and begin writing decoded samples from the 45th byte location onwards. Once the entire data had been decoded, this function is called. The function generates the .wav file header using the information available in the state memory and fills the WaveBuf array. The application copies the content of this buffer into the file in the reserved place available at the start of the file.

Preconditions

None.

Example

```
AAC_WriteWavHeader(haacDecoderInfo, wav);
```

Parameters

Parameters	Description
*haacDecoderInfo	The pointer for the structure that stores decoder information
*adts_ptr	pointer to WAVE header buffer

Function

void AAC_WriteWavHeader(void *haacDecoderInfo, WAVE *wav);

AACDecoder_GetFrameSize Function

Used to get the AAC Decoder frame size.

File

decoder_aac.h

C

```
int32_t AACDecoder_GetFrameSize(void * haacDecoderInfo, uint8_t * adts_ptr);
```

Returns

This function returns the size (in bytes) of the next frame to be decoded.

Description

This function is called every time before calling the AAC decode function. The frame size is extracted from the relevant field of the ADTS header for returning and also the frame size (in bits) is updated in the channel state.

Preconditions

None.

Parameters

Parameters	Description
*haacDecoderInfo	The pointer to the ADTS Header is passed. The application extracts the next available seven bytes from the file and passes them as ADTS header.
*adts_ptr	The pointer to the state memory is passed.

Function

int32_t AACDecoder_GetFrameSize(void *haacDecoderInfo, uint8_t *adts_ptr);

AACDecoder_GetStateSize Function

This function allocates memory for each channel being used.

File

decoder_aac.h

C

```
int32_t AACDecoder_GetStateSize();
```

Returns

This function returns the size of the AAC Decoder state structure in bytes.

Description

Based on the return value, the application allocates the required memory for the AAC Decoder state. This function is called only once for each channel before the channel is initialized.

If dynamic memory allocation (malloc) is not to be used, this function need not be called and the AAC Decoder state buffer shall be statically

defined at compile time with the size of the AAC Decoder state structure (constant).

Preconditions

None.

Example

```
int32_t aac_State_Size;
aac_State_Size = AACDecoder_GetStateSize();
```

Function

int32_t AACDecoder_GetStateSize(void)

AACDecoder_Init Function

Views the received point and clears the structure.

File

decoder_aac.h

C

```
void AACDecoder_Init(void * haacDecoderInfo, uint8_t * adts_ptr);
```

Returns

None.

Description

This function is called only once for each channel before starting the decoding of the first frame. This function views the received pointer as structure pointer and clears the entire structure. Subsequently it initializes the selective variables with the default values. Also, the information on sampling frequency and number of channels are extracted from the header and are stored in the channel state structure.

After returning from this function, the application resets the file pointer to origin as these seven bytes will be fetched again to extract the frame size through the function mentioned next.

Preconditions

None.

Example

```
int32_t aac_State_Size;
AACDecoder_Init(haacDecoderInfo, appData.data);
```

Parameters

Parameters	Description
*haacDecoderInfo	The pointer for the structure that stores decoder information
*adts_ptr	pointer to AAC header structure

Function

void AACDecoder_Init(void *haacDecoderInfo, uint8_t *adts_ptr);

AACDecoder_InterleaveSamples Function

This function interleaves left and right channel data.

File

decoder_aac.h

C

```
int16_t AACDecoder_InterleaveSamples(INTPCM * pTimeCh0, void * state);
```

Returns

This function returns the number of samples in the I/O buffer:

- 1024 - for mono
- 2048 - for stereo

Description

For stereo audio, the left and right channels are interleaved here and stored in the same I/O buffer. For mono channels, nothing is done.

Preconditions

None.

Example

```
outSize = AACDecoder_InterleaveSamples(outBuf, haacDecoderInfo);
```

Parameters

Parameters	Description
*pState	The pointer to the state memory is passed. The details about the number of channels, mono or stereo, is used.
*outBuf	Pointer to the I/O buffer.

Function

int32_t AACDecoder_InterleaveSamples (void *pState, int16_t *outBuf);

isAACdecoder_enabled Function

File

aac.h

C

```
bool isAACdecoder_enabled();
```

Description

This is function isAACdecoder_enabled.

AAC_Decode Function

This function decodes AAC data.

File

decoder_aac.h

C

```
int16_t AAC_Decode(void * DecoderState, uint8_t * inBufferptr, INTPCM * pTimeData, int16_t * outFlag,
int32_t * restBytes);
```

Returns

This function returns either the value '0' or '1' to indicate the status of the decode operation.

- 0 - indicates success
- 1 - indicates failure or end of available encoded data. Upon receiving the value '1', the application aborts processing the channel.

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of frame is detected or the decode function returns the value '1'.

Preconditions

None.

Example

```
errorStatus = AAC_Decode(haacDecoderInfo, inBuffer, outBuf, &outFlag, &restbit);
```

Parameters

Parameters	Description
*DecoderState	The pointer to the state memory is passed.

*inBufferptr	Pointer to the input buffer, where the encoded bit stream is available. Based on the return value from AACDecoder_GetFrameSize(), the application fetches the needed bytes from the bit stream file and fills in this buffer.
*pTimeData	Pointer to the buffer where the output samples are to be stored
*outFlag	The pointer to a variable for output flag: 0 for no valid data (for initial few frames) 1 for valid data present (for all subsequent frames) After decoding the frame, the function indicates whether valid data is present in the output buffer. • restBytes - Pointer to the variable that indicates how many bytes have not decoded. • Normally the number is 0.

Function

int32_t AAC_Decode(void *DecoderState,uint8_t *inBufferptr,short *pTimeData, int16_t *outFlag)

AAC_Initialize Function

File

aac.h

C

```
bool AAC_Initialize(void * heap, uint16_t size, uint8_t * ptr, SYS_FS_HANDLE aacFilehandle);
```

Description

This is function AAC_Initialize.

AAC_RegisterDecoderEventHandlerCallback Function

File

aac.h

C

```
void AAC_RegisterDecoderEventHandlerCallback(SetReadBytesInAppData fptr);
```

Description

This is function AAC_RegisterDecoderEventHandlerCallback.

AAC_GetSamplingFrequency Function

File

aac.h

C

```
int32_t AAC_GetSamplingFrequency(uint8_t * ptr);
```

Description

This is function AAC_GetSamplingFrequency.

b) Data Types and Constants

INTPCM Type

File

decoder_aac.h

C

```
typedef short INTPCM;
```

Section

Middleware & Other Library Configuration

AAC_DECODER_STATES Enumeration

File

aac.h

C

```
typedef enum {
  AAC_GET_FRAME_SIZE,
  AAC_DECODE_FRAME
} AAC_DECODER_STATES;
```

Description

This is type AAC_DECODER_STATES.

WAVE Structure

File

decoder_aac.h

C

```
typedef struct {
  int32_t riff;
  int32_t len;
  int32_t wave;
  int32_t fmt;
  int32_t sublen;
  int16_t mode;
  int16_t num_chnls;
  int32_t samp_freq;
  int32_t bytes_per_sec;
  int16_t bytes_per_sample;
  int16_t bits_per_sample;
  int32_t data;
  int32_t data_len;
} WAVE;
```

Members

Members	Description
int16_t mode;	1 for uncompressed PCM
int16_t num_chnls;	1 for mono and 2 for stereo
int32_t bytes_per_sec;	bytes_per_sample * samp_freq
int16_t bytes_per_sample;	bits_per_sample/8 * num_channels : bytes per set of sample
int16_t bits_per_sample;	bits per word (left or right - not both)

Description

This is type WAVE.

AAC_PROFILE Macro

File

decoder_aac.h

C

```
#define AAC_PROFILE 1
```

Description

This is macro AAC_PROFILE.

AAC_SAMPLING_FREQUENCY_INDEX Enumeration

File

aac.h

C

```
typedef enum {
  SAMPLING_FREQUENCY_IDX0 = 0,
  SAMPLING_FREQUENCY_IDX1,
  SAMPLING_FREQUENCY_IDX2,
  SAMPLING_FREQUENCY_IDX3,
  SAMPLING_FREQUENCY_IDX4,
  SAMPLING_FREQUENCY_IDX5,
  SAMPLING_FREQUENCY_IDX6,
  SAMPLING_FREQUENCY_IDX7,
  SAMPLING_FREQUENCY_IDX8,
  SAMPLING_FREQUENCY_IDX9,
  SAMPLING_FREQUENCY_IDX10,
  SAMPLING_FREQUENCY_IDX11,
  SAMPLING_FREQUENCY_IDX12,
  SAMPLING_FREQUENCY_IDX13,
  SAMPLING_FREQUENCY_IDX14,
  SAMPLING_FREQUENCY_IDX15
} AAC_SAMPLING_FREQUENCY_INDEX;
```

Description

This is type AAC_SAMPLING_FREQUENCY_INDEX.

AAC_ERROR_COUNT_MAX Macro

File

aac.h

C

```
#define AAC_ERROR_COUNT_MAX 1
```

Description

This is macro AAC_ERROR_COUNT_MAX.

AACDECODER_STATE_SIZE Macro

File

decoder_aac.h

C

```
#define AACDECODER_STATE_SIZE 11032                // in Bytes
```

Description

in Bytes

FRAME_SIZE Macro

File

decoder_aac.h

C

```
#define FRAME_SIZE 1024                 // in Samples
```

Description

in Samples

INPUT_BUF_SIZE Macro

File

decoder_aac.h

C

```
#define INPUT_BUF_SIZE (6144*2/8)           // in Bytes
```

Description

in Bytes

MAX_STACK Macro

File

decoder_aac.h

C

```
#define MAX_STACK 0x5000                // in Bytes
```

Description

in Bytes

STACK Macro

File

decoder_aac.h

C

```
#define STACK 1
```

Description

This is macro STACK.

AAC_FRAME_HEADER_SIZE Macro

File

decoder_aac.h

C

```
#define AAC_FRAME_HEADER_SIZE 7                    // in Bytes
```

Description

in Bytes

SetReadBytesInAppData Type

File

aac.h

C

```
typedef void (* SetReadBytesInAppData)(int32_t val);
```

Description

This is type SetReadBytesInAppData.

Files

Files

Name	Description
aac.h	Frame enumerators.
decoder_aac.h	This file consists of the abstract function and input output buffer size declaration for decoding purposes.

Description

This section lists the source and header files used by the MP3 Decoder Library.

aac.h

Frame enumerators.

Enumerations

	Name	Description
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.

Functions

	Name	Description
	AAC_Decoder	This is function AAC_Decoder.
	AAC_GetSamplingFrequency	This is function AAC_GetSamplingFrequency.
	AAC_Initialize	This is function AAC_Initialize.
	AAC_RegisterDecoderEventHandlerCallback	This is function AAC_RegisterDecoderEventHandlerCallback.
	isAACdecoder_enabled	This is function isAACdecoder_enabled.

Macros

	Name	Description
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.

Types

	Name	Description
	SetReadBytesInAppData	This is type SetReadBytesInAppData.

Description

AAC Decoder Library Support File

This header file consists of frame enumerators.

File Name

aac.h

Company

Microchip Technology Inc.

decoder_aac.h

This file consists of the abstract function and input output buffer size declaration for decoding purposes.

Functions

	Name	Description
	AAC_Decode	This function decodes AAC data.
	AAC_WriteWavHeader	Generates the .wav file header, if applicable.
	AACDecoder_GetFrameSize	Used to get the AAC Decoder frame size.
	AACDecoder_GetStateSize	This function allocates memory for each channel being used.

	AACDecoder_Init	Views the received point and clears the structure.
	AACDecoder_InterleaveSamples	This function interleaves left and right channel data.

Macros

	Name	Description
	AAC_FRAME_HEADER_SIZE	in Bytes
	AAC_PROFILE	This is macro AAC_PROFILE.
	AACDECODER_STATE_SIZE	in Bytes
	FRAME_SIZE	in Samples
	INPUT_BUF_SIZE	in Bytes
	MAX_STACK	in Bytes
	STACK	This is macro STACK.

Structures

	Name	Description
	WAVE	This is type WAVE.

Types

	Name	Description
	INTPCM	

Description

AAC Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer. This configuration header must not define any prototypes or data definitions (or include any files that do). It only provides macro definitions for build-time configuration options that are not instantiated until used by another MPLAB Harmony module or application.

File Name

decoder_aac.h

Company

Microchip Technology Inc.

MP3 Decoder Library

This section describes the MP3 Decoder Library.

Introduction

The MP3 Decoder Library is available for the PIC32 family of microcontrollers.

Description

The MP3 Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

The Compact MP3 algorithm is designed to fit in small memory footprint PIC32MX Microcontrollers. This code requires only 28 MIPS of performance (CD Quality audio), 42 KB Flash and 11 KB RAM memory for operation on the PIC32MX. This part number is for Non-modifiable binary code. Source code is also available (refer to the previous web link). Users are responsible for patent-only licensing through Technicolor.

Assembly Language Issue

There is an assembly language issue in the MP3 Decoder Library that misaligns the stack pointer. Contrary to the MIPS programming specification, this error is benign except when the application is built for a device with a hardware Floating Point Unit (FPU), such as the PIC32MZ EF family of devices. In this case, the sub-routine context-saving operations will produce an exception and stop the application. A work around exists for this issue, which involves disabling FPU context-saving for Interrupt Service Routines (ISRs).

The work around is to convert all ISRs from:

```
void __ISR(vector,ipl) MyInterruptHandler(void)
```

and change them to:

```
void __attribute__((vector),interrupt(ipl),nomips16,no_fpu)) MyInterruptHandler(void)
```

where, `vector` and `ipl` are replaced with the actual interrupt vector and interrupt priority level.

An example of this work around can be found in the universal_audio_decoders demonstration.

So as long as no FPU operations occur in ISRs and the routines are called by ISRs, this work around will prevent the misaligned stack pointer from causing an exception. If FPU operations result from an ISR with no_fpu enabled, a general exception will result.

Using the Library

This topic describes the basic architecture of the MP3 Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_mp3.h`

The interface to the MP3 Decoder Library is defined in the `decoder_mp3.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_mp3.h`.

Please refer to the What is MPLAB Harmony? section for how the MP3 Decoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the MP3 Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	MP3_Decode	This is function MP3_Decode.
	MP3Initialize	The function views the received pointer as a structure pointer and clears the entire structure.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3Decode	This function is called once to decode one frame.
	MP3DecoderGetStateSize	This function allocates memory used for decoding.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	MP3DecoderInit	The function views the received pointer as a structure pointer and clears the entire structure.

	isMP3decoder_enabled	This is function isMP3decoder_enabled.
	ID3_Initialize	void ID3_Initialize (void);
	ID3_EventHandler	This is function ID3_EventHandler.
	ID3_Parse	This is function ID3_Parse.
	ID3_ParseFrameV22	This is function ID3_ParseFrameV22.
	ID3_ParseFrameV23	This is function ID3_ParseFrameV23.
	ID3_Parse_Frame	This is function ID3_Parse_Frame.
	MP3Decode	This function is called once to decode one frame.
	MP3_GetAudioSize	This is function MP3_GetAudioSize.
	MP3_UpdatePlaytime	This is function MP3_UpdatePlaytime.
	MP3_Initialize	This is function MP3_Initialize.
	MP3_RegisterDecoderEventHandlerCallback	This is function MP3_RegisterDecoderEventHandlerCallback.
	MP3_GetChannels	This is function MP3_GetChannels.

b) Data Types and Constants

	Name	Description
	MP3_FORMAT	
	MP3_STATE_SIZE	11024
	MP3_EVENT	This is type MP3_EVENT.
	MP3_FRAME_HEADER	This is type MP3_FRAME_HEADER.
	MP3_STATE	This is type MP3_STATE.
	MP3_XING_HEADER	This is type MP3_XING_HEADER.
	MP3_ERROR_COUNT_MAX	This is macro MP3_ERROR_COUNT_MAX.
	MP3_HEADER_CHANNELS_DUAL	This is macro MP3_HEADER_CHANNELS_DUAL.
	MP3_HEADER_CHANNELS_JOINT	This is macro MP3_HEADER_CHANNELS_JOINT.
	MP3_HEADER_CHANNELS_MONO	This is macro MP3_HEADER_CHANNELS_MONO.
	MP3_HEADER_CHANNELS_STEREO	This is macro MP3_HEADER_CHANNELS_STEREO.
	MP3_HEADER_SAMPLERATE_32000	This is macro MP3_HEADER_SAMPLERATE_32000.
	MP3_HEADER_SAMPLERATE_44100	This is macro MP3_HEADER_SAMPLERATE_44100.
	MP3_HEADER_SAMPLERATE_48000	This is macro MP3_HEADER_SAMPLERATE_48000.
	MP3_HEADER_SAMPLERATE_RESV	This is macro MP3_HEADER_SAMPLERATE_RESV.
	MP3_PLAYTIME_H	This is macro MP3_PLAYTIME_H.
	MP3_XING_HEADER_START_MONO	This is macro MP3_XING_HEADER_START_MONO.
	MP3_XING_HEADER_START_STEREO	This is macro MP3_XING_HEADER_START_STEREO.
	ID3_EVENT	This is type ID3_EVENT.
	ID3_STATE	This is type ID3_STATE.
	ID3V1_EXTENDED_TAG	This is type ID3V1_EXTENDED_TAG.
	ID3V1_TAG	This is type ID3V1_TAG.
	ID3V2_TAG_HEADER	This is type ID3V2_TAG_HEADER.
	ID3V22_FRAME	This is type ID3V22_FRAME.
	ID3V22_FRAME_HEADER	This is type ID3V22_FRAME_HEADER.
	ID3V23_FRAME	This is type ID3V23_FRAME.
	ID3V23_FRAME_HEADER	This is type ID3V23_FRAME_HEADER.
	ID3_H	This is macro ID3_H.
	ID3_STRING_SIZE	This is macro ID3_STRING_SIZE.
	ID3V22_ALBUM	This is macro ID3V22_ALBUM.
	ID3V22_ARTIST	This is macro ID3V22_ARTIST.
	ID3V22_TITLE	This is macro ID3V22_TITLE.
	ID3V22_ZERO	This is macro ID3V22_ZERO.
	ID3V23_ALBUM	This is macro ID3V23_ALBUM.
	ID3V23_ARTIST	This is macro ID3V23_ARTIST.
	ID3V23_TITLE	This is macro ID3V23_TITLE.
	ID3V23_ZERO	This is macro ID3V23_ZERO.
	MP3_CHANNEL_INFO	
	MP3MPEG1L3_SAMPLES_PER_FRAME	MPEG1 LayerIII, samples per frame

	MP3MPEG2L3_SAMPLES_PER_FRAME	MPEG2/2.5 LayerIII, samples per frame
	MP3_DEC	This is type MP3_DEC.
	DecoderEventHandlerCB	This is type DecoderEventHandlerCB.

Description

This section describes the Application Programming Interface (API) functions of the MP3 Decoder Library.

Refer to each section for a detailed description.

a) General Functions

MP3_Decode Function

File

mp3.h

C

```
bool MP3_Decode(uint8_t * input, uint16_t inSize, uint16_t * read, uint8_t * output, uint16_t * written);
```

Description

This is function MP3_Decode.

MP3Initialize Function

The function views the received pointer as a structure pointer and clears the entire structure.

File

decoder_mp3_microaptiv.h

C

```
bool MP3Initialize(void * state);
```

Returns

This function returns a boolean value.

- 0 - indicates unsuccessful initialization
- 1 - indicates successful initialization

Description

The function views the received pointer as a structure pointer and clears the entire structure. Subsequently, it initializes the selective variable with the default values. The function is called only once before initiating the decoding process.

Preconditions

None.

Example

```
MP3Initialize ( mp3.decoderVars );
```

Parameters

Parameters	Description
*state	The pointer to the allocated state memory is passed.

Function

bool MP3Initialize(void *state);

MP3_EventHandler Function

File

mp3.h

C

```
bool MP3_EventHandler(MP3_EVENT event, uint32_t data);
```

Description

This is function MP3_EventHandler.

MP3Decode Function

This function is called once to decode one frame.

File

decoder_mp3_microaptiv.h

C

```
uint16_t MP3Decode(void * state, uint8_t * in, uint16_t inSize, uint16_t * used, MP3_FORMAT * format, void
* outBuffer, uint16_t * outSize);
```

Returns

This function returns either the value '0' or '1' to indicate the status of the decode operation.

- 1 - indicates success
- 0 - indicates failure or end of available encoded data. Upon receiving the value '0' the application aborts processing the channel.

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns the value '0'. Unlike most speech applications where the encoded data for each frame is deterministic, this is not the case for audio applications. Therefore, the application provides a portion of input data (usually more than what is necessary for a single frame). The decoder parses the input and decodes it with a portion of the data. The decode function passes the information of how many bytes of data were used to the application so that the application saves the remaining data and presents to the decoder for the subsequent frame.

Preconditions

None.

Example

```
MP3Decode ( mp3.decoderVars, input, inSize, read, &mp3.format, output, written )
```

Parameters

Parameters	Description
*state	The pointer to the state memory is passed.
*in	The pointer to the input buffer , where the encoded bit stream is available.
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer.
*used	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.
*format	The pointer to a channel info structure is passed to the function the decider updates this structure after every call
*outBuffer	The pointer to the output sample buffer where the decided pcm samples are to be written,
*outSize	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

uint16_t MP3Decode (void *state, uint8_t *in, uint16_t inSize, uint16_t *used,
MP3_FORMAT *format, void *outBuffer, uint16_t *outSize);

MP3DecoderGetStateSize Function

This function allocates memory used for decoding.

File

decoder_mp3_microaptiv.h

C

```
uint32_t MP3DecoderGetStateSize();
```

Returns

This function returns the size of the MP3 Decoder state structure in bytes.

Description

Based on the return value, the application allocates the required memory for the MP3 Decoder state. This function is called only once before initializing the decoder state. If dynamic memory allocation (malloc) is not needed, it is not necessary to call this function and the MP3 Decoder state buffer will be statically defined at the compile time with the size of the MP3 decoder state structure.

Preconditions

None.

Example

```
Word mp3StateSize;
mp3StateSize = MP3DecoderGetStateSize();
```

Function

uint32_t MP3DecoderGetStateSize(void);

MP3_ParseVBR Function

File

mp3.h

C

```
bool MP3_ParseVBR(uint8_t* data);
```

Description

This is function MP3_ParseVBR.

MP3DecoderInit Function

The function views the received pointer as a structure pointer and clears the entire structure.

File

decoder_mp3_advanced.h

C

```
void MP3DecoderInit(void * State);
```

Returns

None.

Description

The function views the received pointer as a structure pointer and clears the entire structure. Subsequently, it initializes the selective variable with the default values. The function is called only once before initiating the decoding process.

Preconditions

None.

Example

```
MP3Initialize ( mp3.decoderVars );
```

Parameters

Parameters	Description
*state	The pointer to the allocated state memory is passed.

Function

void MP3DecoderInit(void *state);

isMP3decoder_enabled Function

File

mp3.h

C

```
bool isMP3decoder_enabled();
```

Description

This is function isMP3decoder_enabled.

ID3_Initialize Function

File

id3.h

C

```
int32_t ID3_Initialize(uint8_t * buffer);
```

Description

void ID3_Initialize (void);

ID3_EventHandler Function

File

id3.h

C

```
bool ID3_EventHandler(ID3_EVENT event, uint32_t data);
```

Description

This is function ID3_EventHandler.

ID3_Parse Function

File

id3.h

C

```
uint32_t ID3_Parse(uint8_t * buff, uint16_t size);
```

Description

This is function ID3_Parse.

ID3_ParseFrameV22 Function

File

id3.h

C

```
void ID3_ParseFrameV22(ID3V22_FRAME * frame, uint32_t frameSize, char * id, uint16_t event);
```

Description

This is function ID3_ParseFrameV22.

ID3_ParseFrameV23 Function

File

id3.h

C

```
void ID3_ParseFrameV23(ID3V23_FRAME * frame, uint32_t frameSize, char * id, uint16_t event);
```

Description

This is function ID3_ParseFrameV23.

ID3_Parse_Frame Function

File

id3.h

C

```
uint32_t ID3_Parse_Frame(uint8_t * buffer, size_t left, int8_t * ret);
```

Description

This is function ID3_Parse_Frame.

MP3Decode Function

This function is called once to decode one frame.

File

decoder_mp3_advanced.h

C

```
int16_t MP3Decode(void * state, uint8_t * In, uint16_t inSize, uint16_t * used, MP3_CHANNEL_INFO * format,
void * outBuffer, uint16_t * outSize);
```

Returns

This function returns either the value '0' or '1' or '2' to indicate the status of the decode operation.

- 0 - indicates success
- 1 - indicates that no data is available (thus marking end of file)
- 2 - indicates failure or invalid or corrupted data

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns the value '0'. Unlike most speech applications where the encoded data for each frame is deterministic, this is not the case for audio applications. Therefore, the application provides a portion of input data (usually more than what is necessary for a single frame). The decoder parses the input and decodes it with a portion of the data. The decode function passes the information of how many bytes of data were used to the application so that the application saves the remaining data and presents to the decoder for the subsequent frame.

Preconditions

None.

Example

```
MP3Decode ( mp3.decoderVars, input, inSize, read, &mp3.format, output, written )
```

Parameters

Parameters	Description
*state	The pointer to the state memory is passed.
*in	The pointer to the input buffer , where the encoded bit stream is available.
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer.
*used	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.

*format	The pointer to a channel info structure is passed to the function. The decoder updates this structure after every call. Additional member 'MPEG_Type' to indicate the MPEG version is added in this structure.
*outBuffer	The pointer to the output sample buffer where the decided pcm samples are to be written,
*outSize	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

int16_t MP3Decode (void *state, uint8_t *In, uint16_t inSize, uint16_t *used,
MP3_CHANNEL_INFO *format, void *outBuffer, uint16_t *outSize);

MP3_GetAudioSize Function

File

mp3.h

C

```
uint32_t MP3_GetAudioSize();
```

Description

This is function MP3_GetAudioSize.

MP3_UpdatePlaytime Function

File

mp3.h

C

```
uint32_t MP3_UpdatePlaytime();
```

Description

This is function MP3_UpdatePlaytime.

MP3_Initialize Function

File

mp3.h

C

```
bool MP3_Initialize(void * heap, uint16_t size, SYS_FS_HANDLE mp3Filehandle);
```

Description

This is function MP3_Initialize.

MP3_RegisterDecoderEventHandlerCallback Function

File

mp3.h

C

```
void MP3_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);
```

Description

This is function MP3_RegisterDecoderEventHandlerCallback.

MP3_GetChannels Function

File

mp3.h

C

```
uint8_t MP3_GetChannels();
```

Description

This is function MP3_GetChannels.

b) Data Types and Constants

MP3_FORMAT Structure

File

decoder_mp3_microaptiv.h

C

```
typedef struct {
  uint16_t Kbps;
  uint16_t sampleRate;
  uint8_t bitsPerSample;
  uint8_t channels;
} MP3_FORMAT;
```

Section

Middleware & Other Library Configuration

MP3_STATE_SIZE Macro

File

mp3.h

C

```
#define MP3_STATE_SIZE 16876//11024
```

Description

11024

MP3_EVENT Enumeration

File

mp3.h

C

```
typedef enum {
  MP3_EVENT_TAG_ARTIST,
  MP3_EVENT_TAG_ALBUM,
  MP3_EVENT_TAG_TITLE,
  MP3_EVENT_STREAM_START,
  MP3_EVENT_SAMPLERATE,
  MP3_EVENT_BITRATE,
  MP3_EVENT_TRACK_TIME
} MP3_EVENT;
```

Members

Members	Description
MP3_EVENT_TAG_ARTIST	Align TAG_ARTIST, TAG_ALBUM and TAG_TITLE with ID3 EVENT_TAGS

Description

This is type MP3_EVENT.

MP3_FRAME_HEADER Union

File

mp3.h

C

```
typedef union {
  uint32_t data;
  struct {
    uint8_t sync0 : 8;
    uint8_t CRC : 1;
    uint8_t layer : 2;
    uint8_t mpeg : 2;
    uint8_t sync1 : 3;
    uint8_t padding : 1;
    uint8_t samprate : 2;
    uint8_t bitrate : 4;
    uint8_t nc : 4;
    uint8_t extend : 2;
    uint8_t stereo : 2;
  }
} MP3_FRAME_HEADER;
```

Description

This is type MP3_FRAME_HEADER.

MP3_STATE Enumeration

File

mp3.h

C

```
typedef enum {
  MP3_STATE_STREAM
} MP3_STATE;
```

Description

This is type MP3_STATE.

MP3_XING_HEADER Structure

File

mp3.h

C

```
typedef struct {
  int8_t tag[4];
  union {
    uint32_t flags;
    struct {
      uint32_t frameInfo : 1;
      uint32_t sizeInfo : 1;
      uint32_t tocInfo : 1;
      uint32_t qualityInfo : 1;
    }
  }
  uint8_t frames[4];
  uint8_t size[4];
  uint8_t toc[100];
  uint8_t quality;
} MP3_XING_HEADER;
```

Description

This is type MP3_XING_HEADER.

MP3_ERROR_COUNT_MAX Macro

File

mp3.h

C

```
#define MP3_ERROR_COUNT_MAX 1
```

Description

This is macro MP3_ERROR_COUNT_MAX.

MP3_HEADER_CHANNELS_DUAL Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_DUAL 0b10
```

Description

This is macro MP3_HEADER_CHANNELS_DUAL.

MP3_HEADER_CHANNELS_JOINT Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_JOINT 0b01
```

Description

This is macro MP3_HEADER_CHANNELS_JOINT.

MP3_HEADER_CHANNELS_MONO Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_MONO 0b11
```

Description

This is macro MP3_HEADER_CHANNELS_MONO.

MP3_HEADER_CHANNELS_STEREO Macro

File

mp3.h

C

```
#define MP3_HEADER_CHANNELS_STEREO 0b00
```

Description

This is macro MP3_HEADER_CHANNELS_STEREO.

MP3_HEADER_SAMPLERATE_32000 Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_32000 0b10
```

Description

This is macro MP3_HEADER_SAMPLERATE_32000.

MP3_HEADER_SAMPLERATE_44100 Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_44100 0b00
```

Description

This is macro MP3_HEADER_SAMPLERATE_44100.

MP3_HEADER_SAMPLERATE_48000 Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_48000 0b01
```

Description

This is macro MP3_HEADER_SAMPLERATE_48000.

MP3_HEADER_SAMPLERATE_RESV Macro

File

mp3.h

C

```
#define MP3_HEADER_SAMPLERATE_RESV 0b11
```

Description

This is macro MP3_HEADER_SAMPLERATE_RESV.

MP3_PLAYTIME_H Macro

File

mp3.h

C

```
#define MP3_PLAYTIME_H
```

Description

This is macro MP3_PLAYTIME_H.

MP3_XING_HEADER_START_MONO Macro

File

mp3.h

C

```
#define MP3_XING_HEADER_START_MONO 17
```

Description

This is macro MP3_XING_HEADER_START_MONO.

MP3_XING_HEADER_START_STEREO Macro

File

mp3.h

C

```
#define MP3_XING_HEADER_START_STEREO 32
```

Description

This is macro MP3_XING_HEADER_START_STEREO.

ID3_EVENT Enumeration

File

id3.h

C

```
typedef enum {
  ID3_EVENT_TAG_ARTIST,
  ID3_EVENT_TAG_ALBUM,
  ID3_EVENT_TAG_TITLE
} ID3_EVENT;
```

Description

This is type ID3_EVENT.

ID3_STATE Enumeration

File

id3.h

C

```
typedef enum {
  ID3_STATE_INIT,
  ID3_STATE_READ_V1,
  ID3_STATE_READ_V2_HEADER,
  ID3_STATE_READ_V2_FRAME,
  ID3_STATE_FINISHED
} ID3_STATE;
```

Description

This is type ID3_STATE.

ID3V1_EXTENDED_TAG Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[4];
  int8_t title[60];
  int8_t artist[60];
  int8_t album[60];
  uint8_t speed;
  int8_t genre[30];
  int8_t startTime[6];
  int8_t endTime[6];
} ID3V1_EXTENDED_TAG;
```

Description

This is type ID3V1_EXTENDED_TAG.

ID3V1_TAG Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[3];
  int8_t title[30];
  int8_t artist[30];
  int8_t album[30];
  int8_t year[4];
  union {
    int8_t comment[30];
    struct {
      int8_t commentShort[28];
      uint8_t zero;
      uint8_t track;
    }
  }
  uint8_t genre;
} ID3V1_TAG;
```

Description

This is type ID3V1_TAG.

ID3V2_TAG_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t tag[3];
  uint8_t version;
  uint8_t empty;
  uint8_t flag;
  uint8_t size[4];
} ID3V2_TAG_HEADER;
```

Description

This is type ID3V2_TAG_HEADER.

ID3V22_FRAME Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[3];
  uint8_t size[3];
  uint8_t encoding;
  uint8_t contents[1];
} ID3V22_FRAME;
```

Description

This is type ID3V22_FRAME.

ID3V22_FRAME_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[3];
  uint8_t size[3];
} ID3V22_FRAME_HEADER;
```

Description

This is type ID3V22_FRAME_HEADER.

ID3V23_FRAME Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[4];
  uint8_t size[4];
  uint8_t flags[2];
  uint8_t encoding;
  uint8_t contents[1];
} ID3V23_FRAME;
```

Description

This is type ID3V23_FRAME.

ID3V23_FRAME_HEADER Structure

File

id3.h

C

```
typedef struct {
  uint8_t id[4];
  uint8_t size[4];
  uint8_t flags[2];
} ID3V23_FRAME_HEADER;
```

Description

This is type ID3V23_FRAME_HEADER.

ID3_H Macro

File

id3.h

C

```
#define ID3_H
```

Description

This is macro ID3_H.

ID3_STRING_SIZE Macro

File

id3.h

C

```
#define ID3_STRING_SIZE 128
```

Description

This is macro ID3_STRING_SIZE.

ID3V22_ALBUM Macro

File

id3.h

C

```
#define ID3V22_ALBUM "TAL"
```

Description

This is macro ID3V22_ALBUM.

ID3V22_ARTIST Macro

File

id3.h

C

```
#define ID3V22_ARTIST "TP1"
```

Description

This is macro ID3V22_ARTIST.

ID3V22_TITLE Macro

File

id3.h

C

```
#define ID3V22_TITLE "TT2"
```

Description

This is macro ID3V22_TITLE.

ID3V22_ZERO Macro

File

id3.h

C

```
#define ID3V22_ZERO "\0\0\0"
```

Description

This is macro ID3V22_ZERO.

ID3V23_ALBUM Macro

File

id3.h

C

```
#define ID3V23_ALBUM "TALB"
```

Description

This is macro ID3V23_ALBUM.

ID3V23_ARTIST Macro

File

id3.h

C

```
#define ID3V23_ARTIST "TPE1"
```

Description

This is macro ID3V23_ARTIST.

ID3V23_TITLE Macro

File

id3.h

C

```
#define ID3V23_TITLE "TIT2"
```

Description

This is macro ID3V23_TITLE.

ID3V23_ZERO Macro

File

id3.h

C

```
#define ID3V23_ZERO "\0\0\0\0"
```

Description

This is macro ID3V23_ZERO.

MP3_CHANNEL_INFO Structure

File

decoder_mp3_advanced.h

C

```
typedef struct {
  int16_t bitsPerWord;
  int16_t channels;
  int32_t samplingRate;
  int32_t bitRate;
  int32_t MPEG_Type;
```

```
} MP3_CHANNEL_INFO;
```

Section

Middleware & Other Library Configuration

MP3MPEG1L3_SAMPLES_PER_FRAME Macro

File

mp3.h

C

```
#define MP3MPEG1L3_SAMPLES_PER_FRAME 1152 // MPEG1 LayerIII, samples per frame
```

Description

MPEG1 LayerIII, samples per frame

MP3MPEG2L3_SAMPLES_PER_FRAME Macro

File

mp3.h

C

```
#define MP3MPEG2L3_SAMPLES_PER_FRAME 576  // MPEG2/2.5 LayerIII, samples per frame
```

Description

MPEG2/2.5 LayerIII, samples per frame

MP3_DEC Structure

File

mp3.h

C

```
typedef struct {
  uint16_t mp3SampleRate;
  uint32_t mp3BitRate;
  uint32_t mp3Duration;
  uint32_t firstFramePos;
  bool isVBR;
  uint32_t mp3ValidBytes;
  uint8_t stereo;
} MP3_DEC;
```

Members

Members	Description
uint32_t firstFramePos;	first frame position in file
uint32_t mp3ValidBytes;	for VBR MP3, mp3ValidBytes is the number of bytes in file is given by XING header, for CBR MP3, mp3ValidBytes is MP3 audio file size - first frame position

Description

This is type MP3_DEC.

DecoderEventHandlerCB Type

File

mp3.h

C

```
typedef bool (* DecoderEventHandlerCB)(uint32_t event, uint32_t data);
```

Description

This is type DecoderEventHandlerCB.

Files

Files

Name	Description
mp3.h	MP3 Decoder support API.
id3.h	MP3 Decoder ID3 parser header API.
decoder_mp3_advanced.h	This file consists of the abstract function and input output buffer size declaration for decoding purpose
decoder_mp3_microaptiv.h	This file consists of the abstract function and input output buffer size declaration for decoding purpose

Description

This section lists the source and header files used by the MP3 Decoder Library.

mp3.h

MP3 Decoder support API.

Enumerations

	Name	Description
	MP3_EVENT	This is type MP3_EVENT.
	MP3_STATE	This is type MP3_STATE.

Functions

	Name	Description
	isMP3decoder_enabled	This is function isMP3decoder_enabled.
	MP3_Decode	This is function MP3_Decode.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3_GetAudioSize	This is function MP3_GetAudioSize.
	MP3_GetChannels	This is function MP3_GetChannels.
	MP3_Initialize	This is function MP3_Initialize.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	MP3_RegisterDecoderEventHandlerCallback	This is function MP3_RegisterDecoderEventHandlerCallback.
	MP3_UpdatePlaytime	This is function MP3_UpdatePlaytime.

Macros

	Name	Description
	MP3_ERROR_COUNT_MAX	This is macro MP3_ERROR_COUNT_MAX.
	MP3_HEADER_CHANNELS_DUAL	This is macro MP3_HEADER_CHANNELS_DUAL.
	MP3_HEADER_CHANNELS_JOINT	This is macro MP3_HEADER_CHANNELS_JOINT.
	MP3_HEADER_CHANNELS_MONO	This is macro MP3_HEADER_CHANNELS_MONO.
	MP3_HEADER_CHANNELS_STEREO	This is macro MP3_HEADER_CHANNELS_STEREO.
	MP3_HEADER_SAMPLERATE_32000	This is macro MP3_HEADER_SAMPLERATE_32000.
	MP3_HEADER_SAMPLERATE_44100	This is macro MP3_HEADER_SAMPLERATE_44100.
	MP3_HEADER_SAMPLERATE_48000	This is macro MP3_HEADER_SAMPLERATE_48000.
	MP3_HEADER_SAMPLERATE_RESV	This is macro MP3_HEADER_SAMPLERATE_RESV.
	MP3_PLAYTIME_H	This is macro MP3_PLAYTIME_H.
	MP3_STATE_SIZE	11024
	MP3_XING_HEADER_START_MONO	This is macro MP3_XING_HEADER_START_MONO.
	MP3_XING_HEADER_START_STEREO	This is macro MP3_XING_HEADER_START_STEREO.
	MP3MPEG1L3_SAMPLES_PER_FRAME	MPEG1 LayerIII, samples per frame
	MP3MPEG2L3_SAMPLES_PER_FRAME	MPEG2/2.5 LayerIII, samples per frame

Structures

	Name	Description
	MP3_DEC	This is type MP3_DEC.
	MP3_XING_HEADER	This is type MP3_XING_HEADER.

Types

	Name	Description
	DecoderEventHandlerCB	This is type DecoderEventHandlerCB.

Unions

	Name	Description
	MP3_FRAME_HEADER	This is type MP3_FRAME_HEADER.

Description

MP3 Decoder Library Interface File

This header file consists of support function declarations.

File Name

mp3.h

Company

Microchip Technology Inc.

id3.h

MP3 Decoder ID3 parser header API.

Enumerations

	Name	Description
	ID3_EVENT	This is type ID3_EVENT.
	ID3_STATE	This is type ID3_STATE.

Functions

	Name	Description
	ID3_EventHandler	This is function ID3_EventHandler.
	ID3_Initialize	void ID3_Initialize (void);
	ID3_Parse	This is function ID3_Parse.
	ID3_Parse_Frame	This is function ID3_Parse_Frame.
	ID3_ParseFrameV22	This is function ID3_ParseFrameV22.
	ID3_ParseFrameV23	This is function ID3_ParseFrameV23.

Macros

	Name	Description
	ID3_H	This is macro ID3_H.
	ID3_STRING_SIZE	This is macro ID3_STRING_SIZE.
	ID3V22_ALBUM	This is macro ID3V22_ALBUM.
	ID3V22_ARTIST	This is macro ID3V22_ARTIST.
	ID3V22_TITLE	This is macro ID3V22_TITLE.
	ID3V22_ZERO	This is macro ID3V22_ZERO.
	ID3V23_ALBUM	This is macro ID3V23_ALBUM.
	ID3V23_ARTIST	This is macro ID3V23_ARTIST.
	ID3V23_TITLE	This is macro ID3V23_TITLE.
	ID3V23_ZERO	This is macro ID3V23_ZERO.

Structures

	Name	Description
	ID3V1_EXTENDED_TAG	This is type ID3V1_EXTENDED_TAG.
	ID3V1_TAG	This is type ID3V1_TAG.
	ID3V2_TAG_HEADER	This is type ID3V2_TAG_HEADER.
	ID3V22_FRAME	This is type ID3V22_FRAME.
	ID3V22_FRAME_HEADER	This is type ID3V22_FRAME_HEADER.
	ID3V23_FRAME	This is type ID3V23_FRAME.
	ID3V23_FRAME_HEADER	This is type ID3V23_FRAME_HEADER.

Description

MP3 Decoder Timekeeping and Delay Routines Definitions Header File

This file provides MP3 Decoder ID3 parser header APIs.

File Name

id3.h

Company

Microchip Technology Inc.

decoder_mp3_advanced.h

This file consists of the abstract function and input output buffer size declaration for decoding purpose

Functions

	Name	Description
	MP3Decode	This function is called once to decode one frame.
	MP3DecoderInit	The function views the received pointer as a structure pointer and clears the entire structure.

Structures

	Name	Description
	MP3_CHANNEL_INFO	

Description

MP3 Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer.

File Name

decoder_mp3_advanced.h

Company

Microchip Technology Inc.

decoder_mp3_microaptiv.h

This file consists of the abstract function and input output buffer size declaration for decoding purpose

Functions

	Name	Description
	MP3Decode	This function is called once to decode one frame.
	MP3DecoderGetStateSize	This function allocates memory used for decoding.
	MP3Initialize	The function views the received pointer as a structure pointer and clears the entire structure.

Structures

	Name	Description
	MP3_FORMAT	

Description

MP3 Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer.

File Name

decoder_mp3_microaptiv.h

Company

Microchip Technology Inc.

Opus Decoder Library

This section describes the Opus Decoder Library.

Introduction

Introduces the Opus Decoder Library.

Description

Opus is an open, royalty-free, highly versatile audio codec. Opus is unmatched for interactive speech and music transmission over the Internet, but is also intended for storage and streaming applications. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716, which incorporated technology from Skype's SILK codec and Xiph.Org's CELT codec.

Opus Features

Supported features are:

- Bit rates from 6 kb/s to 510 kb/s
- Sampling rates from 8 kHz (narrowband) to 48 kHz (fullband)
- Frame sizes from 2.5 ms to 60 ms
- Support for both constant bit rate (CBR) and variable bit rate (VBR)
- Audio bandwidth from narrowband to fullband
- Support for speech and music
- Support for mono and stereo
- Support for up to 255 channels (mult-istream frames)
- Dynamically adjustable bit rate, audio bandwidth, and frame size
- Good loss robustness and packet loss concealment (PLC)
- Floating point and fixed-point implementation

The full specification, RFC 6716, including the reference implementation is available from http://www.opus-codec.org. An up-to-date implementation of the Opus standard is available from the Opus Codec downloads page by visiting: https://www.opus-codec.org/downloads/

Typical Applications

- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies
- Toys
- Robots
- Any application using message playback

Resources

Feature	Option	Usage
Opus Library Flash Size	Release Build	< 143 KB
Opus Decoder RAM	Heap Size (Mono mode) Heap Size (Stereo mode)	17 KB 25 KB
Opus Decoding Performance (measured using a PIC32MX270F512L device)	16 kbps Voice 12 kbps Voice	6.8 MCPS/13.6 MIPS 4.8 MCPS/9.6 MIPS

Using the Library

This topic describes the basic architecture of the Opus Decoder Library and provides information and examples on its use.

Description

Interface Header File: `opus.h`

The interface to the Opus Decoder Library is defined in the `opus.h` header file. Any C language source (`.c`) file that uses the Opus Decoder Library should include `opus.h`.

Please refer to the What is MPLAB Harmony? section for how the Opus Decoder Library interacts with the framework.

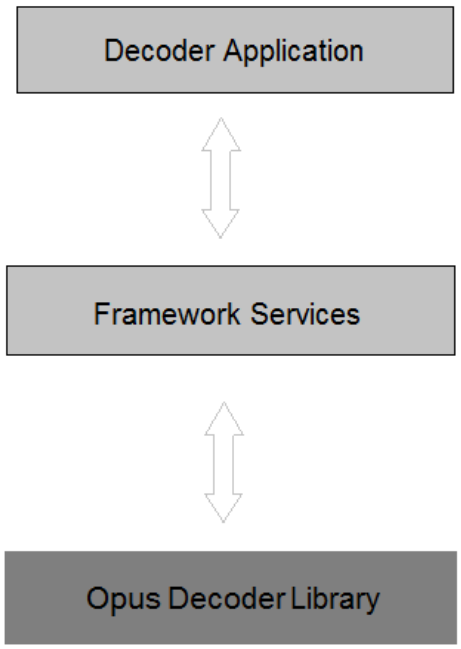
Abstraction Model

Describes the abstraction model for the Opus Decoder Library.

Description

This Opus Library is an Open Source/Free Software patent-free audio compression format designed for interactive speech and music transmission over the Internet. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Opus Decoder Library module.

How the Library Works

This section describes how to work with the MPLAB Harmony Opus abstraction layer interface, which provides simplifier APIs for using the Opus Decoder Library.

Description

For complete documentation and how to implement Opus, visit https://www.opus-codec.org.

Code Example

The following code provides an example for initializing the Opus Decoder and decoding a packet.

```
void DECODER_Initialize(decoder_type){
  switch(decoder_type){
    case OPUS:
      APP_ERROR_MSG ret = OPUS_Initialize(appDataPtr->fileHandle);
      if(ret == APP_SUCCESS)
      {
        // setup other components,
        // for example, audio CODEC, display, etc...
      }
  }
}

void App_Task()
```

```
{
  while(1){
    while(audio is not end){
      // Read one Opus packet
      appData.nBytesRead = OPUS_DiskRead(input_ptr);
     // Decode one packet
     ret = OPUS_Decoder (input_ptr,inSize,read,output,written,outBufSize);
     if(ret == success)
     {
       // continue decoding
     }else{
      // handle decoding errors;
     }
  }
    // Clean up
    OPUS_Cleanup();
    break;
  }
}
```

Library Interface

a) General Functions

	Name	Description
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	opus_decoder_create	Allocates and initializes an decoder state.
	opus_decoder_destroy	Frees an OpusDecoder allocated by opus_decoder_create.
	opus_decoder_get_size	Gets the size of an OpusDecoder structure.
	opus_encoder_create	Allocates and initializes an encoder state.
	opus_encoder_destroy	Frees an OpusEncoder allocated by opus_encoder_create.
	opus_encoder_get_size	Gets the size of an OpusEncoder structure.
	isOPUSdecoder_enabled	Checks if Opus decoder is enabled by MHC configuration.
	OPUS_Cleanup	Frees the memory allocated by the Opus Decoder.
	OPUS_Decoder	Called once to decode one Opus packet.
	OPUS_DiskRead	Called once to read one Opus packet.
	OPUS_GetChannels	Returns the number of channels for this Opus audio.
	OPUS_GetSamplingRate	Returns the sampling rate of Opus audio.
	OPUS_Initialize	Initializes the Opus decoder and creates an Opus decoder state structure.

b) Data Types and Constants

	Name	Description
	OpusDecoder	Opus decoder state.
	OpusEncoder	This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.
	OPUS_H	Opus reference implementation API
	OPUS_ERROR_MSG	Ogg Container Structures
	opusDecDcpt	This is type opusDecDcpt.
	sOggPageHdr	header of an Ogg page, full segment info included
	sOpusHeader	This is type sOpusHeader.
	sOpusPktDcpt	decoder data packet descriptor
	sOpusStreamDcpt	info needed by the stream at run-time
	DECODER_OPUS_SUPPORT_H	This is macro DECODER_OPUS_SUPPORT_H.
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz
	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.

Description

This section describes the Application Programming Interface (API) functions of the Opus Decoder Library.

Refer to each section for a detailed description.

a) General Functions

OPUS_ARG_NONNULL Function

File

opus.h

C

```
OPUS_ARG_NONNULL(1);
```

Description

This is function OPUS_ARG_NONNULL.

OPUS_ARG_NONNULL Function

File

opus.h

C

```
OPUS_ARG_NONNULL(4);
```

Description

This is function OPUS_ARG_NONNULL.

opus_decoder_create Function

Allocates and initializes an decoder state.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusDecoder * opus_decoder_create(opus_int32 Fs, int channels, int *
error);
```

Returns

- OpusEncoder - Opus encoder state structure.

Description

This function allocates and initializes an decoder state.

Remarks

Internally, Opus stores data at 48000 Hz, so that should be the default value for Fs. However, the decoder can efficiently decode to buffers at 8, 12, 16, and 24 kHz so if for some reason the caller cannot use data at the full sample rate, or knows the compressed data does not use the full frequency range, it can request decoding at a reduced rate. Likewise, the decoder is capable of filling in either mono or interleaved stereo pcm buffers, at the caller's request.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
Fs	Sampling rate of input signal (Hz). This must be one of 8000, 12000, 16000, 24000, or 48000.
channels	Number of channels (1 or 2) in input signal.
error	(out) OPUS_OK Success or opus_errorcodes.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusDecoder *opus_decoder_create(opus_int32 Fs,
int channels, int *error)

opus_decoder_destroy Function

Frees an OpusDecoder allocated by opus_decoder_create.

File

opus.h

C

```
OPUS_EXPORT void opus_decoder_destroy(OpusDecoder * st);
```

Returns

None.

Description

This function frees an OpusDecoder allocated by opus_decoder_create.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
st	State to be freed.

Function

OPUS_EXPORT void opus_decoder_destroy(OpusDecoder *st)

opus_decoder_get_size Function

Gets the size of an OpusDecoder structure.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decoder_get_size(int channels);
```

Returns

- int - The size in bytes.

Description

This function gets the size of an OpusDecoder structure.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
channels	Number of channels. This value must be 1 or 2.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_decoder_get_size(int channels)

opus_encoder_create Function

Allocates and initializes an encoder state.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusEncoder * opus_encoder_create(opus_int32 Fs, int channels, int application, int * error);
```

Returns

- OpusEncoder - Opus encoder state structure.

Description

This function allocates and initializes an encoder state. There are three coding modes:

OPUS_APPLICATION_VOIP provides the best quality at a given bit rate for voice signals. It enhances the input signal by high-pass filtering and emphasizing formants and harmonics. Optionally, t includes in-band forward error correction to protect against packet loss. Use this mode for typical VoIP applications. Because of the enhancement, even at high bit rates the output may sound different from the input.

OPUS_APPLICATION_AUDIO provides the best quality at a given bit rate for most non-voice signals like music. Use this mode for music and mixed (music/voice) content, broadcast, and applications requiring less than 15 ms of coding delay.

OPUS_APPLICATION_RESTRICTED_LOWDELAY configures a low-delay mode that disables the speech-optimized mode in exchange for slightly reduced delay. This mode can only be set on a newly initialized or freshly reset encoder because it changes the codec delay.

Remarks

Regardless of the sampling rate and the number channels selected, the Opus encoder can switch to a lower audio bandwidth or number of channels if the bit rate selected is too low. This also means that it is safe to always use 48 kHz stereo input and let the encoder optimize the encoding.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
Fs	Sampling rate of input signal (Hz). This value must be one of 8000, 12000, 16000, 24000, or 48000.
channels	Number of channels (1 or 2) in input signal.
application	Coding mode (OPUS_APPLICATION_VOIP/ OPUS_APPLICATION_AUDIO/ OPUS_APPLICATION_RESTRICTED_LOWDELAY).
error	opus_errorcodes of this function.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT OpusEncoder *opus_encoder_create(opus_int32 Fs, int channels, int application, int *error)

opus_encoder_destroy Function

Frees an OpusEncoder allocated by opus_encoder_create.

File

opus.h

C

```
OPUS_EXPORT void opus_encoder_destroy(OpusEncoder * st);
```

Returns

None.

Description

This function frees an OpusEncoder allocated by opus_encoder_create.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
st	State to be freed.

Function

OPUS_EXPORT void opus_encoder_destroy(OpusEncoder *st)

opus_encoder_get_size Function

Gets the size of an OpusEncoder structure.

File

opus.h

C

```
OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_encoder_get_size(int channels);
```

Returns

- int - The size in bytes.

Description

This function gets the size of an OpusEncoder structure.

Remarks

None.

Preconditions

None.

Example

None.

Parameters

Parameters	Description
channels	Number of channels. This value must be 1 or 2.

Function

OPUS_EXPORT OPUS_WARN_UNUSED_RESULT int opus_encoder_get_size(int channels)

isOPUSdecoder_enabled Function

Checks if Opus decoder is enabled by MHC configuration.

File

opus_support.h

C

```
bool isOPUSdecoder_enabled();
```

Returns

None.

Description

The function checks if Opus decoder is enabled by MHC configuration.

Preconditions

None.

Example

```
bool opusEnabled = isOPUSdecoder_enabled ();
```

Function

bool isOPUSdecoder_enabled()

OPUS_Cleanup Function

Frees the memory allocated by the Opus Decoder.

File

opus_support.h

C

```
void OPUS_Cleanup();
```

Returns

None.

Description

This function frees the memory that was allocated by the Opus Decoder and is called when the Opus Decoder ends.

Preconditions

None.

Example

```
OPUS_Cleanup();
```

Function

void OPUS_Cleanup()

OPUS_Decoder Function

Called once to decode one Opus packet.

File

opus_support.h

C

```
OPUS_ERROR_MSG OPUS_Decoder(const uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output,
uint16_t * written, uint16_t outSize);
```

Returns

The status code of this function.

Description

This function is called once to decode one packet, this function will be called in an infinite while loop until the end of the packet is detected or the decode function returns failure value.

Preconditions

None.

Example

```
OPUS_ERROR_MSG ret = OPUS_Decoder(input, inSize, read, output, written, outSize);
```

Parameters

Parameters	Description
*input	The pointer to the input buffer, where the encoded bit stream is available
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer
*read	The pointer to a variable is passed to the function. After decoding the packet the function writes the number of bytes consumed from the current frame decoder operation
*output	The pointer to the output sample buffer where the decided PCM samples are to be written
*written	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer
outSize	A variable that indicates the size of output buffer, used for avoiding buffer overwritten.

Function

OPUS_ERROR_MSG OPUS_Decoder(const uint8_t *input, uint16_t inSize, uint16_t *read,
int16_t *output, uint16_t *written, uint16_t outSize)

OPUS_DiskRead Function

Called once to read one Opus packet.

File

opus_support.h

C

```
int32_t OPUS_DiskRead(uint8_t * inBuff);
```

Returns

This function returns the size of this Opus packet.

Description

This function is called once to read one Opus packet.

Preconditions

None.

Example

```
int32_t bytesRead = OPUS_DiskRead(inputBuffer);
```

Parameters

Parameters	Description
*inBuff	The pointer to the input buffer, where the Opus encoded data are to be written.

Function

int32_t OPUS_DiskRead(uint8_t *inBuff)

OPUS_GetChannels Function

Returns the number of channels for this Opus audio.

File

opus_support.h

C

```
uint8_t OPUS_GetChannels();
```

Returns

- -1 - 1 channel, mono mode
- -2 - 2 channels, stereo mode

Description

This function returns the number of channels for this Opus audio.

Preconditions

None.

Example

```
uint8_t channelNumber = OPUS_GetChannels();
```

Function

uint8_t OPUS_GetChannels()

OPUS_GetSamplingRate Function

Returns the sampling rate of Opus audio.

File

opus_support.h

C

```
int32_t OPUS_GetSamplingRate();
```

Returns

The sampling rate of Opus audio.

Description

This function returns the sampling rate of this Opus file, return value is always 48000, Opus playback sample rate should be 48000 as described in the OggOpus spec.

Preconditions

None.

Example

```
int32_t sampleRate = OPUS_GetSamplingRate();
```

Function

int32_t OPUS_GetSamplingRate()

OPUS_Initialize Function

Initializes the Opus decoder and creates an Opus decoder state structure.

File

opus_support.h

C

```
OPUS_ERROR_MSG OPUS_Initialize(const SYS_FS_HANDLE opus_file_handler);
```

Returns

The status code of this function.

Description

This function initializes the Opus decoder and creates an Opus decoder state structure.

Preconditions

None.

Example

```
OPUS_ERROR_MSG ret = OPUS_Initialize(fileHandler);
```

Parameters

Parameters	Description
opus_file_handler	The file handler of current Opus file which will be decoded.

Function

OPUS_ERROR_MSG OPUS_Initialize(const SYS_FS_HANDLE opus_file_handler)

b) Data Types and Constants

OpusDecoder Type

Opus decoder state.

File

opus.h

C

```
typedef struct OpusDecoder OpusDecoder;
```

Description

Opus decoder state.

This structure contains the complete state of an Opus decoder. It is position independent and can be freely copied. See opus_decoder_create,opus_decoder_init.

Remarks

None.

OpusEncoder Type

This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.

File

opus.h

C

```
typedef struct OpusEncoder OpusEncoder;
```

Description

Opus encoder state

This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.

Remarks

None.

OPUS_H Macro

Opus reference implementation API

File

opus.h

C

```
#define OPUS_H
```

File Name

opus.h

OPUS_ERROR_MSG Enumeration

File

opus_support.h

C

```
typedef enum {
  OPUS_SUCCESS = 1,
  OPUS_READ_ERROR,
  OPUS_STREAM_ERROR,
  OPUS_BUFF_ERROR,
  OPUS_STREAM_END,
  OPUS_PLAYBACK_ERROR,
  OPUS_OUT_OF_MEM_ERROR,
  OPUS_DISK_ERROR,
  OPUS_GENERAL_ERROR
} OPUS_ERROR_MSG;
```

Description

Ogg Container Structures

opusDecDcpt Structure

File

opus_support.h

C

```
typedef struct {
  int processedPktNo;
  int currPktNo;
  int nInBytes;
} opusDecDcpt;
```

Members

Members	Description
int processedPktNo;	counter of processed packets
int currPktNo;	number of the currently received packet from the stream
int nInBytes;	bytes available in the input buffer

Description

This is type opusDecDcpt.

sOggPageHdr Structure

File

opus_support.h

C

```
typedef struct {
  int32_t pageCapture;
  int8_t struct_ver;
  int8_t headerFlags;
  int64_t granulePos;
  int32_t streamNo;
  int32_t pageNo;
  int32_t pageCrc;
  uint8_t pageSegments;
  uint8_t segmentTbl[255];
} sOggPageHdr;
```

Members

Members	Description
int32_t pageCapture;	should be OGG_ID_MAGIC
int8_t struct_ver;	version of the Ogg file format. Should be 0 (RFC3533)
int8_t headerFlags;	an eOggHeaderFlags value
int64_t granulePos;	stream dependent position info
int32_t streamNo;	logical bit stream identifier
int32_t pageNo;	page sequence number
int32_t pageCrc;	CRC32 checksum of the page
uint8_t pageSegments;	number of page segments to follow
uint8_t segmentTbl[255];	actually segmentTbl[pageSegments]; contains the lace values for all segments in the page

Description

header of an Ogg page, full segment info included

sOpusHeader Structure

File

opus_support.h

C

```
typedef struct {
  char signature[8];
  uint8_t version;
  uint8_t channels;
  uint16_t preskip;
  uint32_t input_sample_rate;
  uint16_t gain;
  uint8_t channel_mapping;
  int8_t nb_streams;
  int8_t nb_coupled;
  unsigned char stream_map[255];
} sOpusHeader;
```

Members

Members	Description
char signature[8];	Magic signature: "OpusHead
uint8_t version;	Version number: 0x01 for this spec
uint8_t channels;	Number of channels: 1..255

uint16_t preskip;	This is the number of samples (at 48kHz) to discard from the decoder output when starting playback
uint32_t input_sample_rate;	Original input sample rate in Hz, this is not the sample rate to use for playback of the encoded data
uint16_t gain;	output gain in dB
uint8_t channel_mapping;	This byte indicates the order and semantic meaning of various channels encoded in each Opus packet The rest is only used if channel_mapping != 0
int8_t nb_streams;	This field indicates the total number of streams so the decoder can correctly parse the packed Opus packets inside the Ogg packet
int8_t nb_coupled;	Describes the number of streams whose decoders should be configured to produce two channels

Description

This is type sOpusHeader.

sOpusPktDcpt Structure

File

opus_support.h

C

```
typedef struct {
  int pktBytes;
  int pktSeqNo;
} sOpusPktDcpt;
```

Members

Members	Description
int pktBytes;	how many bytes in this packet
int pktSeqNo;	packet sequence number

Description

decoder data packet descriptor

sOpusStreamDcpt Structure

File

opus_support.h

C

```
typedef struct {
  sOggPageHdr pageHdr;
  int segIx;
  int pktIx;
  int prevBytes;
} sOpusStreamDcpt;
```

Members

Members	Description
sOggPageHdr pageHdr;	current page header
int segIx;	current packet segment index in the current page
int pktIx;	current packet index, 0 -> ...
int prevBytes;	previous value of the bytes in the encoded output buffer

Description

info needed by the stream at run-time

DECODER_OPUS_SUPPORT_H Macro

File

opus_support.h

C

```
#define DECODER_OPUS_SUPPORT_H
```

Description

This is macro DECODER_OPUS_SUPPORT_H.

OPUS_INPUT_BUFFER_SIZE Macro

File

opus_support.h

C

```
#define OPUS_INPUT_BUFFER_SIZE (1024*2)
```

Description

MACROS

OPUS_MAX_FRAME_SIZE Macro

File

opus_support.h

C

```
#define OPUS_MAX_FRAME_SIZE (960*6) // 120ms @ 48Khz
```

Description

120ms @ 48Khz

OPUS_OUTPUT_BUFFER_SIZE Macro

File

opus_support.h

C

```
#define OPUS_OUTPUT_BUFFER_SIZE (1024*7)
```

Description

This is macro OPUS_OUTPUT_BUFFER_SIZE.

Files

Files

Name	Description
opus.h	Opus reference implementation API.
opus_support.h	This is file opus_support.h.

Description

This section lists the source and header files used by the Opus Decoder Library.

opus.h

Opus reference implementation API.

Functions

	Name	Description
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	OPUS_ARG_NONNULL	This is function OPUS_ARG_NONNULL.
	opus_decoder_create	Allocates and initializes an decoder state.
	opus_decoder_destroy	Frees an OpusDecoder allocated by opus_decoder_create.
	opus_decoder_get_size	Gets the size of an OpusDecoder structure.
	opus_encoder_create	Allocates and initializes an encoder state.
	opus_encoder_destroy	Frees an OpusEncoder allocated by opus_encoder_create.
	opus_encoder_get_size	Gets the size of an OpusEncoder structure.

Macros

	Name	Description
	OPUS_H	Opus reference implementation API

Types

	Name	Description
	OpusDecoder	Opus decoder state.
	OpusEncoder	This contains the complete state of an Opus encoder. It is position independent and can be freely copied. See opus_encoder_create,opus_encoder_init.

Description

Opus codec Library

Opus reference implementation API.

File Name

opus.h

Company

Microchip Technology Inc.

opus_support.h

Enumerations

	Name	Description
	OPUS_ERROR_MSG	Ogg Container Structures

Functions

	Name	Description
	isOPUSdecoder_enabled	Checks if Opus decoder is enabled by MHC configuration.
	OPUS_Cleanup	Frees the memory allocated by the Opus Decoder.
	OPUS_Decoder	Called once to decode one Opus packet.
	OPUS_DiskRead	Called once to read one Opus packet.
	OPUS_GetChannels	Returns the number of channels for this Opus audio.
	OPUS_GetSamplingRate	Returns the sampling rate of Opus audio.
	OPUS_Initialize	Initializes the Opus decoder and creates an Opus decoder state structure.

Macros

	Name	Description
	DECODER_OPUS_SUPPORT_H	This is macro DECODER_OPUS_SUPPORT_H.
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz
	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.

Structures

	Name	Description
	opusDecDcpt	This is type opusDecDcpt.
	sOggPageHdr	header of an Ogg page, full segment info included
	sOpusHeader	This is type sOpusHeader.
	sOpusPktDcpt	decoder data packet descriptor
	sOpusStreamDcpt	info needed by the stream at run-time

Description

This is file opus_support.h.

Speex Decoder Library

This section describes the Speex Decoder Library.

Introduction

Introduces the Speex Decoder Library.

Description

Speex is a Code Excited Linear Prediction (CELP) based open source patent-free audio compression format designed for speech.

Speex Features

- Fixed-point implementation
- Narrowband (8 kHz) and wideband (16 kHz) bit-streams
- Wide range of bit-rates available (from 2.15 kbps to 44 kbps)
- Perceptual enhancement (attempts to reduce the perception of the noise/distortion produced by the encoding/decoding process. In most cases, perceptual enhancement brings the sound further from the original objectively (e.g. considering only SNR), but in the end it still sounds better (subjective improvement)

Visit www.speex.org for further details and complete documentation

Typical Applications

- Answering machines; Voice recorders
- Building and home safety systems; Intercoms
- Smart appliances
- Walkie-talkies; Toys and robots
- Any application using message playback

Resources

Feature	Option	Usage
Speex Library Flash Size	Release Build	< 60 KB
	Debug Build	< 92 KB
Speex Decoder RAM	Dynamic	4 KB
	Stack	2 KB
Speex Decoding Performance	Narrow Mode	280 KB/sec

Using the Library

This topic describes the basic architecture of the Speex Decoder Library and provides information and examples on its use.

Description

Interface Header File: `speex.h`

The interface to the Speex Decoder Library is defined in the `speex.h` header file. Any C language source (`.c`) file that uses the Speex Decoder Library should include `speex.h`.

Please refer to the What is MPLAB Harmony? section for how the Speex Decoder Library interacts with the framework.

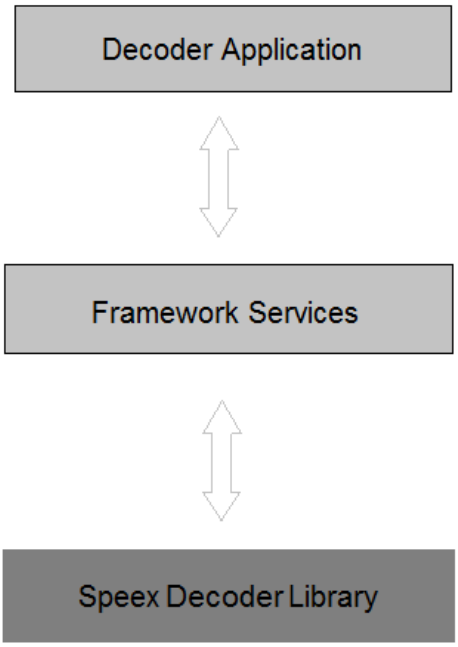
Abstraction Model

Describes the abstraction model for the Speex Decoder Library.

Description

This Speex Library is an Open Source/Free Software patent-free audio compression format designed for speech. The MPLAB Harmony framework provides an abstraction layer to easily interact with this library. Refer to the universal_audio_decoders demonstration for reference.

Abstraction Model Diagram



Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the Speex Decoder Library module.

How the Library Works

This section describes how to work with the MPLAB Harmony Speex abstraction layer interface, which provides simplifier APIs for using the Speex Decoder Library.

Description

For complete documentation and how to implement Speex, visit www.speex.org.

Code Example

The following code provides an example for initializing the Speex Decoder Library and reading and decoding one Speex packet.

```
void Decoder_Initialize(decoder_type){
    switch (decoder_type){
        case SPEEX:
            ret = SPEEX_Initialize();
            if(ret == success)
            {
                // start decoding
            }
    }
}

void App_Task()
{
    while(1){
        while(audio is not end){
            // Read one Speex packet
            appData.nBytesRead = SPEEX_DiskRead(input_ptr);

            // Decode one packet
            ret = SPEEX_Decoder (input_ptr,inSize,read,output,written);
            if(ret == success)
            {
                // continue decoding
            }else{
                // handle decoding errors;
```

```
            }
        }
        // Clean up
        SPEEX_Cleanup();
    }

}
```

Library Interface

a) General Functions

	Name	Description
	isSPEEXdecoder_enabled	Indicates whether the Speex Decoder is enabled.
	SPEEX_Cleanup	Frees the memory allocated by the Speex Decoder.
	SPEEX_Decoder	Called once to decode one packet.
	SPEEX_DiskRead	Reads one packet of Ogg-Speex audio.
	SPEEX_GetBitrate	Returns the bit rate of Ogg-Speex audio.
	SPEEX_GetSamplingRate	Returns the sampling rate of Ogg-Speex audio.
	SPEEX_Initialize	Reads the Speex header page and initializes the Speex Decoder state.

b) Data Types and Constants

	Name	Description
	eSpxFlags	Speex flags
	pProgressFnc	progress display function
	progressDcpt	encoder/decoder progress activity descriptor
	sOggPageSegHdr	header of an Ogg page, full segment info included
	SPEEX_ERROR_MSG	This is type SPEEX_ERROR_MSG.
	spxCdcDcpt	Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part
	spxDecDcpt	This is type spxDecDcpt.
	sSpeexHeader	the Speex header, the first page in the Speex stream
	sSpxPktDcpt	decoder data packet descriptor
	sSpxRunDcpt	run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()
	sSpxStreamDcpt	info needed by the stream at run-time
	OGG_ID_SPEEX	The Speex packet ID
	SPEEX_INPUT_BUFFER_SIZE	This is macro SPEEX_INPUT_BUFFER_SIZE.
	SPEEX_OUTPUT_BUFFER_SIZE	This is macro SPEEX_OUTPUT_BUFFER_SIZE.
	SPEEX_STRING_LENGTH	The size of the Speex string
	SPEEX_SUPPORT_H	This is macro SPEEX_SUPPORT_H.
	SPEEX_VENDOR_STR	comment in Ogg header
	SPEEX_VERSION	The Speex version string
	SPEEX_VERSION_ID	Version identifier
	SPEEX_VERSION_LENGTH	The size of the Speex version string
	SPX_CODEC_BUFF_DIV	20% is sufficient
	SPX_CODEC_BUFF_MUL	Values to adjust the average decoder buffer size

Description

This section describes the Application Programming Interface (API) functions of the Speex Decoder Library.

Refer to each section for a detailed description.

a) General Functions

isSPEEXdecoder_enabled Function

Indicates whether the Speex Decoder is enabled.

File

speex.h

C

```
bool isSPEEXdecoder_enabled();
```

Returns

This function returns a Boolean value.

- true - Indicates that the Speex Decoder is enabled
- false - Indicates that the Speex Decoder is disabled

Description

This function indicates whether the Speex Decoder is enabled.

Preconditions

None.

Example

```
appData.SPEEX_decoder_enabled   = isSPEEXdecoder_enabled();
```

Function

bool isSPEEXdecoder_enabled()

SPEEX_Cleanup Function

Frees the memory allocated by the Speex Decoder.

File

speex.h

C

```
void SPEEX_Cleanup();
```

Returns

None.

Description

This function frees the memory that was allocated by the Speex Decoder and is called when the Speex Decoder ends.

Preconditions

None.

Example

```
SPEEX_Cleanup();
```

Function

void SPEEX_Cleanup()

SPEEX_Decoder Function

Called once to decode one packet.

File

speex.h

C

```
SPEEX_ERROR_MSG SPEEX_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t
* written);
```

Returns

This function returns an enum value.

- SPEEX_SUCCESS - Indicates success
- SPEEX_READ_ERROR - Indicates read failure
- SPEEX_STREAM_ERROR - Indicates Ogg-Speex parsing error
- SPEEX_STREAM_END - Indicates end of the stream

Description

This function is called once to decode one frame. This function will be called in an infinite while loop until the end of the frame is detected or the decode function returns failure value.

Preconditions

None.

Example

```
res = SPEEX_Decoder (input,inSize,read,output,written);
```

Parameters

Parameters	Description
*input	The pointer to the input buffer , where the encoded bit stream is available
inSize	A variable of data type uint16_t indicating the number of valid bytes available in the input buffer
*read	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of bytes consumed from the current frame decoder operation.
*output	The pointer to the output sample buffer where the decided PCM samples are to be written,
*written	The pointer to a variable is passed to the function. After decoding the frame the function writes the number of valid bytes available in the output buffer.

Function

SPEEX_ERROR_MSG SPEEX_Decoder(uint8_t *input, uint16_t inSize, uint16_t *read,
int16_t *output, uint16_t *written);

SPEEX_DiskRead Function

Reads one packet of Ogg-Speex audio.

File

speex.h

C

```
int32_t SPEEX_DiskRead(uint8_t * inBuff);
```

Returns

This function returns the size of reading bytes.

Description

This function is called once to read one packet.

Preconditions

None.

Example

```
appData.nBytesRead = SPEEX_DiskRead(ptr);
```

Parameters

Parameters	Description
*inBuff	The pointer to the input buffer, where one packet data is to to be written

Function

int32_t SPEEX_DiskRead(uint8_t *inBuff)

SPEEX_GetBitrate Function

Returns the bit rate of Ogg-Speex audio.

File

speex.h

C

```
int32_t SPEEX_GetBitrate();
```

Returns

This function returns the bit rate.

Description

This function returns the bit rate rate of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
decoderBitrate = SPEEX_GetBitrate()/1000;
```

Function

int32_t SPEEX_GetBitrate()

SPEEX_GetSamplingRate Function

Returns the sampling rate of Ogg-Speex audio.

File

speex.h

C

```
int32_t SPEEX_GetSamplingRate();
```

Returns

This function returns the sampling rate.

Description

This function returns the sampling rate of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
sampling_frequency = SPEEX_GetSamplingRate();
```

Function

int32_t SPEEX_GetSamplingRate()

SPEEX_Initialize Function

Reads the Speex header page and initializes the Speex Decoder state.

File

speex.h

C

```
SPEEX_ERROR_MSG SPEEX_Initialize(uintptr_t spx_file_handle);
```

Returns

This function returns an enum value.

- SPEEX_SUCCESS - Indicates success
- SPEEX_READ_ERROR - Indicates read failure
- SPEEX_STREAM_ERROR - Indicates Ogg-Speex parsing error
- SPEEX_STREAM_END - Indicates end of the stream

Description

The function internally calls the Speex Codec Decoder initialization function. The function is called only once before initiating the decoding process.

Preconditions

Call SPEEX_Cleanup function before this function for safety.

Example

```
APP_ERROR_MSG res = SPEEX_Initialize(spx_file_handle);
```

Function

SPEEX_ERROR_MSG SPEEX_Initialize(uintptr_t spx_file_handle);

b) Data Types and Constants

eSpxFlags Enumeration

File

speex.h

C

```
typedef enum {
  SPX_FLAG_WB = 0x1,
  SPX_FLAG_VBR = 0x2,
  SPX_FLAG_BRATE_OVR = 0x4,
  SPX_FLAG_PRCPT_ENH = 0x8
} eSpxFlags;
```

Members

Members	Description
SPX_FLAG_WB = 0x1	wide/narrow band
SPX_FLAG_VBR = 0x2	VBR
SPX_FLAG_BRATE_OVR = 0x4	bit rate setting overrides the encoder quality
SPX_FLAG_PRCPT_ENH = 0x8	decoder perceptual enhancement on/off

Description

Speex flags

Remarks

at most 8 flags are supported right now!

pProgressFnc Type

File

speex.h

C

```
typedef void (* pProgressFnc)(int nBytes);
```

Description

progress display function

progressDcpt Structure

File

speex.h

C

```
typedef struct {
  int progressStep;
  int progressCnt;
  pProgressFnc progressFnc;
} progressDcpt;
```

Members

Members	Description
int progressStep;	no of bytes to process before calling the progress callback
int progressCnt;	current counter
pProgressFnc progressFnc;	progress callback

Description

encoder/decoder progress activity descriptor

sOggPageSegHdr Structure

File

speex.h

C

```
typedef struct {
  int32_t pageCapture;
  int8_t struct_ver;
  int8_t headerFlags;
  int64_t granulePos;
  int32_t streamNo;
  int32_t pageNo;
  int32_t pageCrc;
  int8_t pageSegments;
  uint8_t segmentTbl[255];
} sOggPageSegHdr;
```

Members

Members	Description
int32_t pageCapture;	should be OGG_ID_MAGIC
int8_t struct_ver;	version of the Ogg file format. Should be 0 (RFC3533)
int8_t headerFlags;	an eOggHeaderFlags value
int64_t granulePos;	stream dependent position info
int32_t streamNo;	logical bit stream identifier
int32_t pageNo;	page sequence number
int32_t pageCrc;	CRC32 checksum of the page
int8_t pageSegments;	number of page segments to follow
uint8_t segmentTbl[255];	actually segmentTbl[pageSegments]; contains the lace values for all segments in the page

Description

header of an Ogg page, full segment info included

SPEEX_ERROR_MSG Enumeration

File

speex.h

C

```
typedef enum {
  SPEEX_SUCCESS = 1,
  SPEEX_READ_ERROR,
  SPEEX_STREAM_ERROR,
  SPEEX_BUFF_ERROR,
  SPEEX_STREAM_END,
  SPEEX_PLAYBACK_ERROR,
  SPEEX_OUT_OF_MEM_ERROR,
  SPEEX_DISK_ERROR,
  SPEEX_GENERAL_ERROR
} SPEEX_ERROR_MSG;
```

Description

This is type SPEEX_ERROR_MSG.

spxCdcDcpt Structure

File

speex.h

C

```
typedef struct {
  SpeexBits spxBits;
  void* spxState;
  int inBuffSize;
  int outBuffSize;
  int nOutBytes;
} spxCdcDcpt;
```

Members

Members	Description
SpeexBits spxBits;	codec control structure
void* spxState;	codec state
int inBuffSize;	input buffer size
int nOutBytes;	bytes available in the output buffer

Description

Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part

spxDecDcpt Structure

File

speex.h

C

```
typedef struct {
  spxCdcDcpt cdc;
  int framesPerPacket;
  int frameSize;
  int processedPktNo;
  int currPktNo;
  int nInBytes;
  int outFrameSize;
} spxDecDcpt;
```

Members

Members	Description
spxCdcDcpt cdc;	common part
int framesPerPacket;	frames per Ogg packet
int frameSize;	size of the frames
int processedPktNo;	counter of processed packets

int currPktNo;	number of the currently received packet from the stream
int nInBytes;	bytes available in the input buffer

Description

This is type spxDecDcpt.

sSpeexHeader Structure

File

speex.h

C

```
typedef struct {
  char speexString[SPEEX_STRING_LENGTH];
  char speexVer[SPEEX_VERSION_LENGTH];
  int32_t speexVerId;
  int32_t headerSize;
  int32_t sampleRate;
  int32_t wBand;
  int32_t modeBitsStreamVer;
  int32_t nChannels;
  int32_t bitRate;
  int32_t frameSamples;
  int32_t vbr;
  int32_t framesPerPacket;
  int32_t extraHeaders;
  int32_t reserved1;
  int32_t reserved2;
} sSpeexHeader;
```

Members

Members	Description
char speexString[SPEEX_STRING_LENGTH];	identify the Speex bit-stream: OGG_ID_SPEEX
char speexVer[SPEEX_VERSION_LENGTH];	Speex version that encoded the file
int32_t speexVerId;	Speex version (for checking compatibility)
int32_t headerSize;	sizeof(SpeexHeader)
int32_t sampleRate;	Sampling rate used
int32_t wBand;	0 for narrowband, 1 for wideband
int32_t modeBitsStreamVer;	Version ID of the bit-stream
int32_t nChannels;	Number of channels encoded
int32_t bitRate;	Bit-rate used
int32_t frameSamples;	Size of frames, samples
int32_t vbr;	1 for a VBR encoding, 0 otherwise
int32_t framesPerPacket;	Number of frames stored per Ogg packet
int32_t extraHeaders;	Number of additional headers after the comments
int32_t reserved1;	Reserved for future use, 0
int32_t reserved2;	Reserved for future use, 0

Description

the Speex header, the first page in the Speex stream

sSpxPktDcpt Structure

File

speex.h

C

```
typedef struct {
  int pktBytes;
  int pktSeqNo;
} sSpxPktDcpt;
```

Members

Members	Description
int pktBytes;	how many bytes in this packet
int pktSeqNo;	packet sequence number

Description

decoder data packet descriptor

sSpxRunDcpt Structure

File

speex.h

C

```
typedef struct {
  int streamNo;
  int streamVer;
  unsigned short frameSamples;
  unsigned short bitRate;
  unsigned char framesPerPacket;
  unsigned char packetsPerPage;
  unsigned char complexity;
  unsigned char qualFactor;
  char spxFlags;
  char reserved[3];
} sSpxRunDcpt;
```

Members

Members	Description
int streamNo;	stream number inside the container
int streamVer;	stream version
unsigned short frameSamples;	Size of frames, in samples
unsigned short bitRate;	encoder bit rate
unsigned char framesPerPacket;	Number of frames stored per Ogg packet, <10
unsigned char packetsPerPage;	number of packets in an Ogg page <= 255
unsigned char complexity;	encoder complexity 1-10
unsigned char qualFactor;	encoder quality factor 1-10
char spxFlags;	eSpxFlags: run time flags, wideband, VBR
char reserved[3];	future expansion/padding

Description

run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()

sSpxStreamDcpt Structure

File

speex.h

C

```
typedef struct {
  sSpxRunDcpt runDcpt;
  sOggPageSegHdr pageHdr;
  int segIx;
  int pktIx;
  int prevBytes;
} sSpxStreamDcpt;
```

Members

Members	Description
sSpxRunDcpt runDcpt;	global info

sOggPageSegHdr pageHdr;	current page header
int segIx;	current packet segment index in the current page
int pktIx;	current packet index, 0 -> ...
int prevBytes;	previous value of the bytes in the encoded output buffer

Description

info needed by the stream at run-time

OGG_ID_SPEEX Macro

File

speex.h

C

```
#define OGG_ID_SPEEX "Speex   "    // The Speex packet ID
```

Description

The Speex packet ID

SPEEX_INPUT_BUFFER_SIZE Macro

File

speex.h

C

```
#define SPEEX_INPUT_BUFFER_SIZE 1024
```

Description

This is macro SPEEX_INPUT_BUFFER_SIZE.

SPEEX_OUTPUT_BUFFER_SIZE Macro

File

speex.h

C

```
#define SPEEX_OUTPUT_BUFFER_SIZE 1024*7
```

Description

This is macro SPEEX_OUTPUT_BUFFER_SIZE.

SPEEX_STRING_LENGTH Macro

File

speex.h

C

```
#define SPEEX_STRING_LENGTH 8           // The size of the Speex string
```

Description

The size of the Speex string

SPEEX_SUPPORT_H Macro

File

speex.h

C

```
#define SPEEX_SUPPORT_H
```

Description

This is macro SPEEX_SUPPORT_H.

SPEEX_VENDOR_STR Macro

File

speex.h

C

```
#define SPEEX_VENDOR_STR "Encoded by Microchip Audio Library ver 1.0"    // comment in Ogg header
```

Description

comment in Ogg header

SPEEX_VERSION Macro

File

speex.h

C

```
#define SPEEX_VERSION "speex-1.2beta3"    // The Speex version string
```

Description

The Speex version string

SPEEX_VERSION_ID Macro

File

speex.h

C

```
#define SPEEX_VERSION_ID 1                    // Version identifier
```

Description

Version identifier

SPEEX_VERSION_LENGTH Macro

File

speex.h

C

```
#define SPEEX_VERSION_LENGTH 20               // The size of the Speex version string
```

Description

The size of the Speex version string

SPX_CODEC_BUFF_DIV Macro

File

speex.h

C

```
#define SPX_CODEC_BUFF_DIV 5    // 20% is sufficient
```

Description

20% is sufficient

SPX_CODEC_BUFF_MUL Macro

File

speex.h

C

```
#define SPX_CODEC_BUFF_MUL 6    // Values to adjust the average decoder buffer size
```

Description

Values to adjust the average decoder buffer size

Files

Files

Name	Description
speex.h	This file is an abstraction layer of the Speex Library.

Description

This section lists the source and header files used by the Speex Decoder Library.

speex.h

This file is an abstraction layer of the Speex Library.

Enumerations

	Name	Description
	eSpxFlags	Speex flags
	SPEEX_ERROR_MSG	This is type SPEEX_ERROR_MSG.

Functions

	Name	Description
	isSPEEXdecoder_enabled	Indicates whether the Speex Decoder is enabled.
	SPEEX_Cleanup	Frees the memory allocated by the Speex Decoder.
	SPEEX_Decoder	Called once to decode one packet.
	SPEEX_DiskRead	Reads one packet of Ogg-Speex audio.
	SPEEX_GetBitrate	Returns the bit rate of Ogg-Speex audio.
	SPEEX_GetSamplingRate	Returns the sampling rate of Ogg-Speex audio.
	SPEEX_Initialize	Reads the Speex header page and initializes the Speex Decoder state.

Macros

	Name	Description
	OGG_ID_SPEEX	The Speex packet ID
	SPEEX_INPUT_BUFFER_SIZE	This is macro SPEEX_INPUT_BUFFER_SIZE.
	SPEEX_OUTPUT_BUFFER_SIZE	This is macro SPEEX_OUTPUT_BUFFER_SIZE.
	SPEEX_STRING_LENGTH	The size of the Speex string
	SPEEX_SUPPORT_H	This is macro SPEEX_SUPPORT_H.
	SPEEX_VENDOR_STR	comment in Ogg header
	SPEEX_VERSION	The Speex version string
	SPEEX_VERSION_ID	Version identifier
	SPEEX_VERSION_LENGTH	The size of the Speex version string
	SPX_CODEC_BUFF_DIV	20% is sufficient
	SPX_CODEC_BUFF_MUL	Values to adjust the average decoder buffer size

Structures

	Name	Description
	progressDcpt	encoder/decoder progress activity descriptor
	sOggPageSegHdr	header of an Ogg page, full segment info included
	spxCdcDcpt	Speex codec descriptor common part of encoder/decoder since we don't take the trouble to have a type to distinguish between descriptors, we need to have a common part
	spxDecDcpt	This is type spxDecDcpt.
	sSpeexHeader	the Speex header, the first page in the Speex stream
	sSpxPktDcpt	decoder data packet descriptor
	sSpxRunDcpt	run-time Speex descriptor obtained from the stream with AudioStreamGetRunInfo()
	sSpxStreamDcpt	info needed by the stream at run-time

Types

	Name	Description
	pProgressFnc	progress display function

Description

Speex Decoder Library Interface Definition

This file contains the Speex Decoder-specific definitions and function prototypes.

File Name

speex.h

Company

Microchip Technology Inc.

WMA Decoder Library

This section describes the WMA Decoder Library.

Introduction

The WMA Decoder Library is available for the PIC32 family of microcontrollers.

Description

The WMA Decoder Library is a premium product and is not included in the standard release of MPLAB Harmony and must be purchased separately. The library is only available in binary format, and is only available to Microsoft Windows Media Component Licensees. For licensing information, please visit: http://windows.microsoft.com/en-us/windows/windowsmedia-components-licensing. Refer to the Microchip Premium MPLAB Harmony Audio web page (www.microchip.com/pic32harmonypremiumaudio) for additional information.

Windows Media Audio (WMA) developed by Microsoft, is a format enabling the storage of digital audio using the Lossy compression algorithm.

The Microchip WMA Decoder can decode audio signals sampled at up to 48 kHz with up to two discrete channels. The WMA Decoder also supports VBR and CBR encoded audio stream. In most circumstances, `.wma` files are contained in Advance Systems Format (ASF), which is supported by the WMA Decoder. The WMA Decoder library is optimized (C/ASM) and is available for all PIC32MX devices.

Assembly Language Issue

There is an assembly language issue in the WMA Decoder Library that misaligns the stack pointer. Contrary to the MIPS programming specification, this error is benign except when the application is built for a device with a hardware Floating Point Unit (FPU), such as the PIC32MZ EF family of devices. In this case, the sub-routine context-saving operations will produce an exception and stop the application. A work around exists for this issue, which involves disabling FPU context-saving for Interrupt Service Routines (ISRs).

The work around is to convert all ISRs from:

```
void __ISR(vector,ipl) MyInterruptHandler(void)
```

and change them to:

```
void __attribute__((vector),interrupt(ipl),nomips16,no_fpu)) MyInterruptHandler(void)
```

where, `vector` and `ipl` are replaced with the actual interrupt vector and interrupt priority level.

An example of this work around can be found in the universal_audio_decoders demonstration.

So as long as no FPU operations occur in ISRs and the routines are called by ISRs, this work around will prevent the misaligned stack pointer from causing an exception. If FPU operations result from an ISR with no_fpu enabled, a general exception will result.

Using the Library

This topic describes the basic architecture of the WMA Decoder Library and provides information and examples on its use.

Description

Interface Header File: `decoder_wma.h`

The interface to the WMA Decoder Library is defined in the `decoder_wma.h` header file. Any C language source (`.c`) file that uses the MP3 Decoder Library should include `decoder_wma.h`.

Please refer to the What is MPLAB Harmony? section for how the WMA Decoder Library interacts with the framework.

Library Overview

The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the WMA Decoder Library module.

Library Interface

a) General Functions

	Name	Description
	stpnr	This is function stpnr.
	WMA_Decode	This function is called once to decode one or multiple frames.
	WMA_Decoder_Init	This function reads the pointers and clears the structure.
	WMA_FileGetPCM	Extracts the 16-bit samples.
	WMA_GetEncodeData	Retrieves encoded data from the WAVE file header.
	WMA_GetStateSize	This function requires the required memory for the WMA Decoder state.

	WMA_PacketData	Extracts the encoded bitstream from the packet's payload.
	WMA_PacketDatasize	This function determines the packet size for decoding.
	WMA_Parser	Parses the WMA data.
	WMA_Decoder	This is function WMA_Decoder.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.
	WMA_Header_Current_Offset	This function returns the amount of data used and updates the pointer.
	WMA_WaveHdr	Generates the .wav file header and fills the WavBuf array.
	WMA_FileDecodeClr	Clears all of the allocated memory.
	WMA_FreeMemory	This is function WMA_FreeMemory.
	isWMAdecoder_enabled	This is function isWMAdecoder_enabled.
	WMA_BitRate_Get	This is function WMA_BitRate_Get.
	WMA_GetHeaderPacketOffset	This is function WMA_GetHeaderPacketOffset.
	WMA_Initialize	This is function WMA_Initialize.
	WMA_RegisterAppCallback	This is function WMA_RegisterAppCallback.
	WMA_GetChannels	This is function WMA_GetChannels.

b) Data Types and Constants

	Name	Description
	SYS_DEBUG_BUFFER_DMA_READY	This should be defined in system_config.h. It is added here as a build safe-guard.
	WMA_DECODER_STATES	This is type WMA_DECODER_STATES.
	WMA_ERROR_COUNT_MAX	This is macro WMA_ERROR_COUNT_MAX.
	WMA_H	This is macro WMA_H.
	Chnl_Info	This is type Chnl_Info.
	WMA_Status	This is type WMA_Status.
	BYTES_PER_SAMPLE	This is macro BYTES_PER_SAMPLE.
	MAX_SAMPLES	in Bytes #define WAVE_HEADER_SIZE 44 //in Bytes
	WMA_MAX_INPUT_BUFFER_SIZE	in Bytes
	GetReadBytesInAppData	This is type GetReadBytesInAppData.
	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.

Description

This section describes the Application Programming Interface (API) functions of the WMA Decoder Library.
Refer to each section for a detailed description.

a) General Functions

stpnr Function

File

decoder_wma.h

C

```
static __inline int stpnr(int x);
```

Description

This is function stpnr.

WMA_Decode Function

This function is called once to decode one or multiple frames.

File

decoder_wma.h

C

```
int32_t WMA_Decode(void* paudec, int* pcSamplesReady, int* paudecState);
```

Returns

This function returns the error type.

Description

This function is called once to decode one or multiple frames. This function will be called in an infinite while loop until the end-of-frame is detected or the decode function returns an error value. Unlike most speech applications, where the encoded data for each frame is deterministic, this is not the case in audio applications. Therefore, the application provides a chunk of input packet data (usually more than what is necessary for a single frame). The decoder parses them and decodes with a portion of the data. The decode function passes the information of how many bytes of data were used to the application so that the application saves the remaining data and presents to the decoder for the subsequent frame.

Remarks

None.

Preconditions

None.

Example

```
Int res;
res = WMA_Decode(WMA_State, &nDecodedSamples, &chnlInfo.WMAState);
```

Parameters

Parameters	Description
paudec	The pointer to the WMA Decoder handler
pcSamplesReady	The pointer to the number of decoded samples
paudecState	The pointer to the state of the decoder

Function

int32_t WMA_Decode (void* paudec, int* pcSamplesReady, int* paudecState);

WMA_Decoder_Init Function

This function reads the pointers and clears the structure.

File

decoder_wma.h

C

```
WMA_Status WMA_Decoder_Init(void* pState, Chnl_Info * chnl, char * WMADataBuf);
```

Returns

This function returns the error result.

Description

This function views the received pointer as structure pointer and clears the entire structure. Subsequently, it initializes the selective variables with the default values. This function is called only once for each file before starting the decoding of an entire packet.

Preconditions

None.

Example

```
void *WMA_State;
Chnl_Info chnlInfo;
WMA_Status WMA_status;
char WMA_InBuf[MAX_INPUT_BUF];

WMA_Status = WMA_Decoder_Init (WMA_State, &chnlInfo, WMA_InBuf);
```

Parameters

Parameters	Description
pState	The pointer to the allocated state memory.
*chnl	Holds the essential audio file information and state of the file

WMADataBuf	The pointer to the input buffer

Function

WMA_Status WMA_Decoder_Init (void* pState, Chnl_Info *chnl, char * WMADataBuf);

WMA_FileGetPCM Function

Extracts the 16-bit samples.

File

decoder_wma.h

C

```
uint32_t WMA_FileGetPCM(void * pInt, short * pCh, uint32_t max_nsamples);
```

Returns

This function returns the number of samples.

Description

This function is used to extract the 16-bit samples.

Preconditions

None.

Example

```
nDecodedSamples = WMA_FileGetPCM (WMA_State, WMA_OutBuf, nDecodedSamples);
```

Parameters

Parameters	Description
*pInt	The pointer to the WMA handler
*pCh	The pointer to the output data
max_nsamples	Total number of samples

Function

uint32_t WMA_FileGetPCM (void *pInt, short *pCh, uint32_t max_nsamples);

WMA_GetEncodeData Function

Retrieves encoded data from the WAVE file header.

File

decoder_wma.h

C

```
void WMA_GetEncodeData(void * fWMA, char * pData, int pDtSz);
```

Returns

None.

Description

The function is used to retrieve the encoded data from the WAVE file header.

Preconditions

None.

Parameters

Parameters	Description
*fWMA	The WMA file
*pData	The pointer to the data
pDtSz	The pointer to the data size

Function

void WMA_GetEncodeData(void *fWMA, char *pData, int pDtSz);

WMA_GetStateSize Function

This function requires the required memory for the WMA Decoder state.

File

decoder_wma.h

C

```
uint32_t WMA_GetStateSize();
```

Returns

This function returns the size of the WMA Decoder state structure in bytes.

Description

Based on the return value, the application allocates the required memory for the WMA Decoder state. This function is called only once for each channel before the channel is initialized. If dynamic memory allocation (malloc) is not needed, it is not necessary to call this function and the WMA Decoder state buffer will be statically defined at compile time with the size of the WMA Decoder state structure (constant).

Preconditions

None.

Example

```
unsigned long nDecodedSamples;
nDecodedSamples = WMA_GetStateSize();
```

Function

uint32_t WMA_GetStateSize(void);

WMA_PacketData Function

Extracts the encoded bitstream from the packet's payload.

File

decoder_wma.h

C

```
int WMA_PacketData(void * p, char * InBuf, int residue);
```

Returns

None.

Description

This function is used to extract the encoded bitstream from the packet’s payload. This function is repeated until there are no bytes to decode.

Preconditions

None.

Example

```
void *WMA_State;
Chnl_Info chnlInfo;
Int res, residue;
res = WMA_PacketData(WMA_State,WMA_InBuf,residue);
```

Parameters

Parameters	Description
*p	The pointer to the WMA handler
InBuf	The pointer to the input buffer
residue	The pointer to the input file

Function

int WMA_PacketData(void *p, char * InBuf, int residue);

WMA_PacketDatasize Function

This function determines the packet size for decoding.

File

decoder_wma.h

C

```
int32_t WMA_PacketDatasize(void * p);
```

Returns

This function returns the packet data size.

Description

This function determines the packet size for decoding and sends request to the main program to read the number of bytes data for decoding. This function is called repetitively until there are no bytes to decode.

Preconditions

None.

Example

```
void *WMA_State;
int packetdatasize;
packetdatasize = WMA_PacketDatasize(WMA_State);
```

Parameters

Parameters	Description
*p	The pointer to the WMA handler

Function

int32_t WMA_PacketDatasize(void *p);

WMA_Parser Function

Parses the WMA data.

File

decoder_wma.h

C

```
int WMA_Parser(void * pInt);
```

Returns

None.

Description

This function is used to parse the WMA data.

Preconditions

None.

Parameters

Parameters	Description
*pInt	The pointer to the WMA handler

Function

int WMA_Parser(void *pInt);

WMA_Decoder Function

File

wma.h

C

```
int16_t WMA_Decoder(uint8_t * input, uint16_t inSize, uint16_t * read, int16_t * output, uint16_t *
written);
```

Description

This is function WMA_Decoder.

WMA_SamplingFrequency_Get Function

File

wma.h

C

```
int32_t WMA_SamplingFrequency_Get();
```

Description

This is function WMA_SamplingFrequency_Get.

WMA_Header_Current_Offset Function

This function returns the amount of data used and updates the pointer.

File

decoder_wma.h

C

```
uint32_t WMA_Header_Current_Offset(void * pInt);
```

Returns

This function returns the amount of data used.

Description

This function returns the amount of data used and updates the file pointer location to read input data from the current offset after parsing the header information. This function is called only once for each file before starting the decoding of entire packets.

Preconditions

None.

Example

```
Uint32_t packet_offset;
packet_offset = WMA_Header_Current_Offset(WMA_State);
```

Parameters

Parameters	Description
*pInt	The pointer to the WMA Handler

Function

uint32_t WMA_Header_Current_Offset(tWMAFileStateInternal *pInt)

WMA_WaveHdr Function

Generates the .wav file header and fills the WavBuf array.

File

decoder_wma.h

C

```
void WMA_WaveHdr(char * WavHdr, void * hWMA, uint32_t DataLen);
```

Returns

None.

Description

For wave file generation, the application will leave 44 bytes of free space at the beginning of the file and begins writing decoded samples from the 45th byte location onwards. Once the entire data had been decoded, this function is called. The function generates the .wav file header and fills the WavBuf array. The application copies the contents of this buffer into the reserved place available at the beginning of the file.

Preconditions

None.

Example

```
WMA_WaveHdr(WavBuf, WMA_State, DataLen);
```

Parameters

Parameters	Description
*WavHdr	The pointer to the header buffer of a WAVE file. The buffer size is 44 bytes.
*hWMA	The pointer to the WMA Handler
DataLen	Integer variable indicating the total number of samples written into the file

Function

void WMA_WaveHdr(char *WavHdr, void *hWMA, uint32_t DataLen);

WMA_FileDecodeClr Function

Clears all of the allocated memory.

File

decoder_wma.h

C

```
void WMA_FileDecodeClr(void* WMA_State);
```

Returns

None.

Description

This function is used to clear all of the allocated WMA state memory.

Preconditions

None.

Function

void WMA_FileDecodeClr(void* WMA_State);

WMA_FreeMemory Function

File

wma.h

C

```
void WMA_FreeMemory();
```

Description

This is function WMA_FreeMemory.

isWMAdecoder_enabled Function

File

wma.h

C

```
bool isWMAdecoder_enabled();
```

Description

This is function isWMAdecoder_enabled.

WMA_BitRate_Get Function

File

wma.h

C

```
int32_t WMA_BitRate_Get();
```

Description

This is function WMA_BitRate_Get.

WMA_GetHeaderPacketOffset Function

File

wma.h

C

```
int32_t WMA_GetHeaderPacketOffset();
```

Description

This is function WMA_GetHeaderPacketOffset.

WMA_Initialize Function

File

wma.h

C

```
void WMA_Initialize(SYS_FS_HANDLE wmaFilehandle, uint32_t inputBufferSize);
```

Description

This is function WMA_Initialize.

WMA_RegisterAppCallback Function

File

wma.h

C

```
void WMA_RegisterAppCallback(SetReadBytesReadFlagInAppData fptr0, GetReadBytesInAppData fptr1);
```

Description

This is function WMA_RegisterAppCallback.

WMA_GetChannels Function

File

wma.h

C

```
uint8_t WMA_GetChannels();
```

Description

This is function WMA_GetChannels.

b) Data Types and Constants

SYS_DEBUG_BUFFER_DMA_READY Macro

File

sys_debug.h

C

```
#define SYS_DEBUG_BUFFER_DMA_READY
```

Description

This should be defined in system_config.h. It is added here as a build safe-guard.

WMA_DECODER_STATES Enumeration

File

wma.h

C

```
typedef enum {
  WMA_GET_FRAME_SIZE,
  WMA_DECODE_FRAME
} WMA_DECODER_STATES;
```

Description

This is type WMA_DECODER_STATES.

WMA_ERROR_COUNT_MAX Macro

File

wma.h

C

```
#define WMA_ERROR_COUNT_MAX 1
```

Description

This is macro WMA_ERROR_COUNT_MAX.

WMA_H Macro

File

wma.h

C

```
#define WMA_H
```

Description

This is macro WMA_H.

Chnl_Info Structure

File

decoder_wma.h

C

```
typedef struct {
  long SamplesPerSec;
  long Channels;
  long BitRate;
  long Version;
  long ByteUsed;
  int WMAState;
  long PacketSize;
} Chnl_Info;
```

Members

Members	Description
long SamplesPerSec;	Sampling Frequency
long Channels;	No of Channels
long BitRate;	Bit Rate
long Version;	WMA Version
long ByteUsed;	Used Input Data
int WMAState;	Input or Decode State
long PacketSize;	Size of Packet

Description

This is type Chnl_Info.

WMA_Status Enumeration

File

decoder_wma.h

C

```
typedef enum {
  WMA_NO_ERR = 0,
  WMA_NEED_DATA = 1,
  WMA_ERR = 2
} WMA_Status;
```

Description

This is type WMA_Status.

BYTES_PER_SAMPLE Macro

File

decoder_wma.h

C

```
#define BYTES_PER_SAMPLE 2
```

Description

This is macro BYTES_PER_SAMPLE.

MAX_SAMPLES Macro

File

decoder_wma.h

C

```
#define MAX_SAMPLES 2048                    //in Bytes
```

Description

in Bytes #define WAVE_HEADER_SIZE 44 //in Bytes

WMA_MAX_INPUT_BUFFER_SIZE Macro

File

decoder_wma.h

C

```
#define WMA_MAX_INPUT_BUFFER_SIZE 1024*20   //in Bytes
```

Description

in Bytes

GetReadBytesInAppData Type

File

wma.h

C

```
typedef int32_t (* GetReadBytesInAppData)();
```

Description

This is type GetReadBytesInAppData.

SetReadBytesReadFlagInAppData Type

File

wma.h

C

```
typedef void (* SetReadBytesReadFlagInAppData)(int32_t val, bool b);
```

Description

This is type SetReadBytesReadFlagInAppData.

Files

Files

Name	Description
wma.h	WMA Decoder support API.
decoder_wma.h	This file consists of the abstract function and input output buffer size declaration for decoding purpose

Description

This section lists the source and header files used by the MP3 Decoder Library.

wma.h

WMA Decoder support API.

Enumerations

	Name	Description
	WMA_DECODER_STATES	This is type WMA_DECODER_STATES.

Functions

	Name	Description
	isWMAdecoder_enabled	This is function isWMAdecoder_enabled.
	WMA_BitRate_Get	This is function WMA_BitRate_Get.
	WMA_Decoder	This is function WMA_Decoder.
	WMA_FreeMemory	This is function WMA_FreeMemory.
	WMA_GetChannels	This is function WMA_GetChannels.
	WMA_GetHeaderPacketOffset	This is function WMA_GetHeaderPacketOffset.
	WMA_Initialize	This is function WMA_Initialize.
	WMA_RegisterAppCallback	This is function WMA_RegisterAppCallback.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.

Macros

	Name	Description
	WMA_ERROR_COUNT_MAX	This is macro WMA_ERROR_COUNT_MAX.
	WMA_H	This is macro WMA_H.

Types

	Name	Description
	GetReadBytesInAppData	This is type GetReadBytesInAppData.
	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.

Description

MP3 Decoder Library Interface File

This header file consists of support function declarations.

File Name

wma.h

Company

Microchip Technology Inc.

decoder_wma.h

This file consists of the abstract function and input output buffer size declaration for decoding purpose

Enumerations

	Name	Description
	WMA_Status	This is type WMA_Status.

Functions

	Name	Description
	stpnr	This is function stpnr.
	WMA_Decode	This function is called once to decode one or multiple frames.
	WMA_Decoder_Init	This function reads the pointers and clears the structure.
	WMA_FileDecodeClr	Clears all of the allocated memory.
	WMA_FileGetPCM	Extracts the 16-bit samples.
	WMA_GetEncodeData	Retrieves encoded data from the WAVE file header.

	WMA_GetStateSize	This function requires the required memory for the WMA Decoder state.
	WMA_Header_Current_Offset	This function returns the amount of data used and updates the pointer.
	WMA_PacketData	Extracts the encoded bitstream from the packet's payload.
	WMA_PacketDatasize	This function determines the packet size for decoding.
	WMA_Parser	Parses the WMA data.
	WMA_WaveHdr	Generates the .wav file header and fills the WavBuf array.

Macros

	Name	Description
	BYTES_PER_SAMPLE	This is macro BYTES_PER_SAMPLE.
	MAX_SAMPLES	in Bytes #define WAVE_HEADER_SIZE 44 //in Bytes
	WMA_MAX_INPUT_BUFFER_SIZE	in Bytes

Structures

	Name	Description
	Chnl_Info	This is type Chnl_Info.

Description

WMA Decoder Library Interface File

The header file consists of function declaration for the abstract functions to invoke decoding . The header file also defines the size of input samples and i/p and o/p buffer.

File Name

decoder_wma.h

Company

Microchip Technology Inc.

Index

O

P

S

U

W