Decoder Libraries Help

MPLAB Harmony Integrated Software Framework

Volume V: MPLAB Harmony Framework Reference

This volume provides API reference information for the framework libraries included in your installation of MPLAB Harmony.

Description

This volume is a programmer reference that details the interfaces to the libraries that comprise MPLAB Harmony and explains how to use the libraries individually to accomplish the tasks for which they were designed.

Decoder Libraries Help

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

AAC Decoder Library

This section describes the AAC Decoder Library.

Introduction

Advanced Audio Coding (AAC) is a proprietary audio coding standard for lossy digital audio compression. Designed to be the successor of the MP3 format, AAC generally achieves better sound quality than MP3 at the same bit rate. The AAC Decoder Library in MPLAB Harmony is a decoder for Low-Complexity Profile (AAC-LC), which is available for the PIC32MX and PIC32MZ 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.

Features

- Supports the MPEG-4 AAC-LC Profile
- Supports Audio Data Transport Stream (ADTS) input formats
- Supports Sampling Rates from 8 kHz to 96 kHz
- Supports bitrates from 8 kbps to 1152 kbps

Library Performance

The following matrix was measured on a PIC32MX family device.

	Performance Statistics (Average)	Memory Statistics	
AAC 44.1 kHz Test Vector	MIPS	Data RAM	Flash Memory
	73	26.72 KB	47.13 KB

Input buffer for one frame: 1536 bytes

Output Buffer: 4096 bytes for 16-bit audio sample, stereo output

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 AAC 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.

Please refer to Universal Audio Decoders (universal_audio_decoders) demonstration to see how to use the MPLAB Harmony AAC framework APIs to decode a AAC frame.

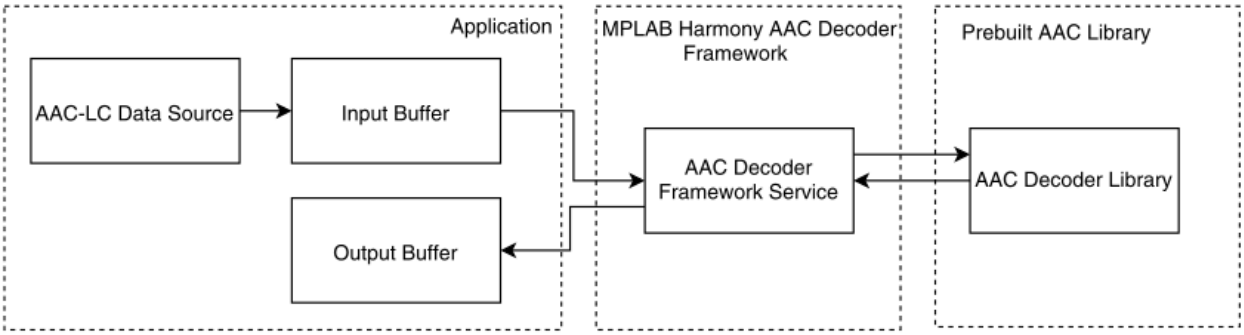
Abstraction Model

This library provides an abstraction of the MPEG4 AAC-LC Decoder.

Description

The following block diagram describes how the AAC Decoder Library interacts with an application. The AAC Decoder Framework Service is another layer of abstraction of the AAC decoder library, which provides application APIs to set and get decoder state information.

Abstraction Model



Library Overview

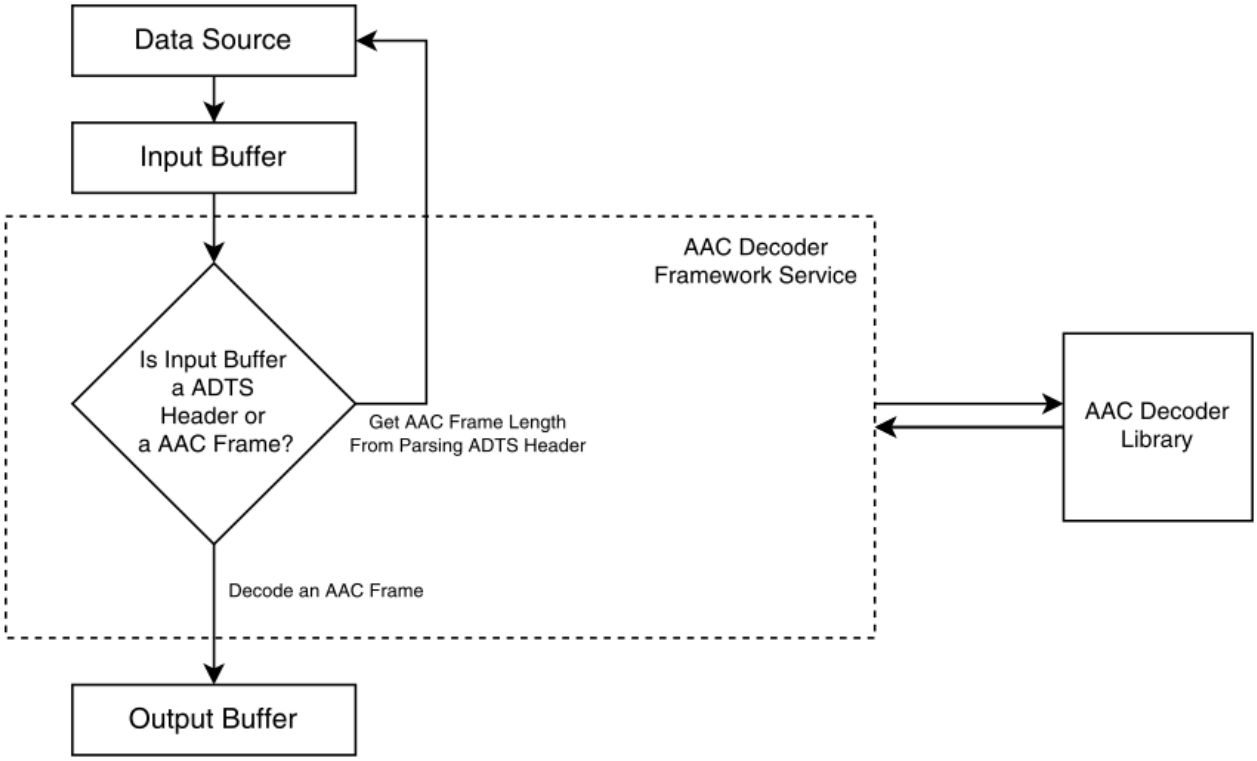
The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the AAC Decoder Library module.

How the Library Works

This topic provides information on how the library works.

Description

The following diagram describes the data flow between the AAC Decoder and an application.



Initializing the AAC Decoder

Initialize the AAC Decoder by calling function AAC_Initialize from the `aac_dec.h` file. This function initializes the AAC decoder state structure, and obtains streaming information from the first ADTS header. The application can then retrieve stream information by using the APIs in `aac_dec.h` after initialization.

Decoding a AAC Frame

The AAC_Decoder function in `aac_dec.h` is used to decode a single AAC frame, and depending on the current state of the AAC Decoder, this function either parses a ADTS header to get the next AAC frame size, or decodes a single AAC frame.

Code Example

```
void DECODER_Initialize(decoder_type){
       switch(decoder_type){
```

```
                case AAC_DECODER:
                     bool ret = AAC_Initialize(aacDecoderPtr,sizeOfAACDecoder, adtsPtr, aacFilehandle)
                     if(ret)
                     {
                         // setup other components,
                         // for example, audio CODEC, display, etc...
                     }
                }
        }
void App_Task()
{
        while(1){
                while(not end of audio data){
                     // Read aac frame to input buffer pointer
                     appData.nBytesRead = OPUS_DiskRead(input_ptr);
                     // Decode aac frame
                     bool ret = AAC_Decoder (input_ptr,inSize,read,output,written,outBufSize);
                     if(ret)
                     {
                         // continue decoding
                     }else{
                         // handle decoding errors;
                     }
                }
                break;
        }
}
```

Library Interface

a) General Functions

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

b) Data Types and Constants

	Name	Description
	AAC_DECODER_STATES	This is type AAC_DECODER_STATES.
	AAC_SAMPLING_FREQUENCY_INDEX	This is type AAC_SAMPLING_FREQUENCY_INDEX.
	AAC_ERROR_COUNT_MAX	This is macro AAC_ERROR_COUNT_MAX.
	SetReadBytesInAppData	This is type SetReadBytesInAppData.
	AAC_DEC_H	This is macro AAC_DEC_H.

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_dec.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.

isAACdecoder_enabled Function

File

aac_dec.h

C

```
bool isAACdecoder_enabled();
```

Description

This is function isAACdecoder_enabled.

AAC_Initialize Function

File

aac_dec.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_dec.h

C

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

Description

This is function AAC_RegisterDecoderEventHandlerCallback.

AAC_GetSamplingFrequency Function

File

aac_dec.h

C

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

Description

This is function AAC_GetSamplingFrequency.

AAC_GetChannels Function

File

aac_dec.h

C

```
uint8_t AAC_GetChannels();
```

Description

This is function AAC_GetChannels.

b) Data Types and Constants

AAC_DECODER_STATES Enumeration

File

aac_dec.h

C

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

Description

This is type AAC_DECODER_STATES.

AAC_SAMPLING_FREQUENCY_INDEX Enumeration

File

aac_dec.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_dec.h

C

```
#define AAC_ERROR_COUNT_MAX 1
```

Description

This is macro AAC_ERROR_COUNT_MAX.

SetReadBytesInAppData Type

File

aac_dec.h

C

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

Description

This is type SetReadBytesInAppData.

AAC_DEC_H Macro

File

aac_dec.h

C

```
#define AAC_DEC_H
```

Description

This is macro AAC_DEC_H.

Files

Files

Name	Description
aac_dec.h	AAC decoder interface header file.

Description

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

aac_dec.h

AAC decoder interface header file.

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_GetChannels	This is function AAC_GetChannels.
	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_DEC_H	This is macro AAC_DEC_H.
	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 describes the interface functions of AAC decoder.

File Name

aac_dec.h

Company

Microchip Technology Inc.

FLAC Decoder Library

This section describes the FLAC Decoder Library.

Introduction

FLAC stands for Free Lossless Audio Codec, which is an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality. This Library is Based on xiph.org's FLAC 1.3.1 source code, and is under xiph.org's BSD license. This source code of FLAC 1.3.1 is modified to use input and output buffer from application instead of dynamic allocating in library.

Description

FLAC (Free Lossless Audio Codec) is an audio coding format for lossless compression of digital audio, and is also the name of the reference codec implementation. Digital audio compressed by the FLAC algorithm can typically be reduced to 50–60% of its original size and decompresses to an identical copy of the original audio data.

Features

- Lossless decoder
- Supports native FLAC format streaming
- No theoretical limitation on maximum Sample rate and bitrate

Note: Lossless decoding will consume large buffer size in runtime, therefore, refer to the FLAC performance table to determine whether it fits in your application.

Library Performance

The following matrix was measured on a PIC32MX family device.

FLAC Type	Performance Statistics (Average)	Memory Statistics	
	MIPS	Data RAM	Flash Memory
96kHz_24bits Native FLAC	54	32 KB (always the same size with output buffer)	113 KB
44.1kHz_24bits Native FLAC	42	9K	113 KB

Input buffer for one FLAC Frame: 16 KB
(to be practical it should be at least 1 KB, 16 KB works for most native FLAC frames)
Output Buffer: at least 32 KB + 32 bytes
(padding bytes because of FLAC algorithm for a 4096 samples FLAC frame)

Using the Library

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

Description

Interface Header File: `flac.h`

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

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

Please refer to Universal Audio Decoders (univeral_audio_decoders) demonstration to see how to use the MPLAB Harmony FLAC framework APIs to decode a FLAC Frame.

Abstraction Model

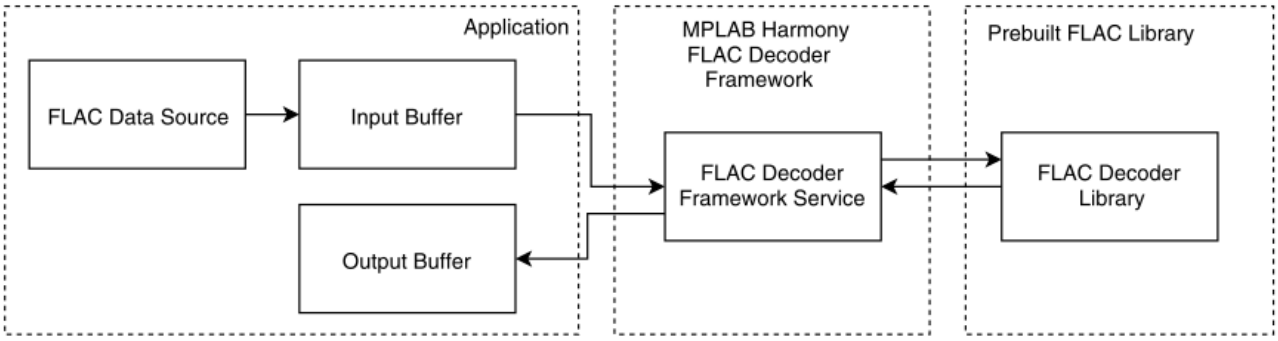
Describes the abstraction model for the FLAC Decoder Library.

Description

The following block diagram describes how the FLAC decoder library interacts with application.

The FLAC Decoder Framework Service is another layer of abstraction of the FLAC Decoder Library, which provides application APIs to set and get decoder state information.

Abstraction Model



Library Overview

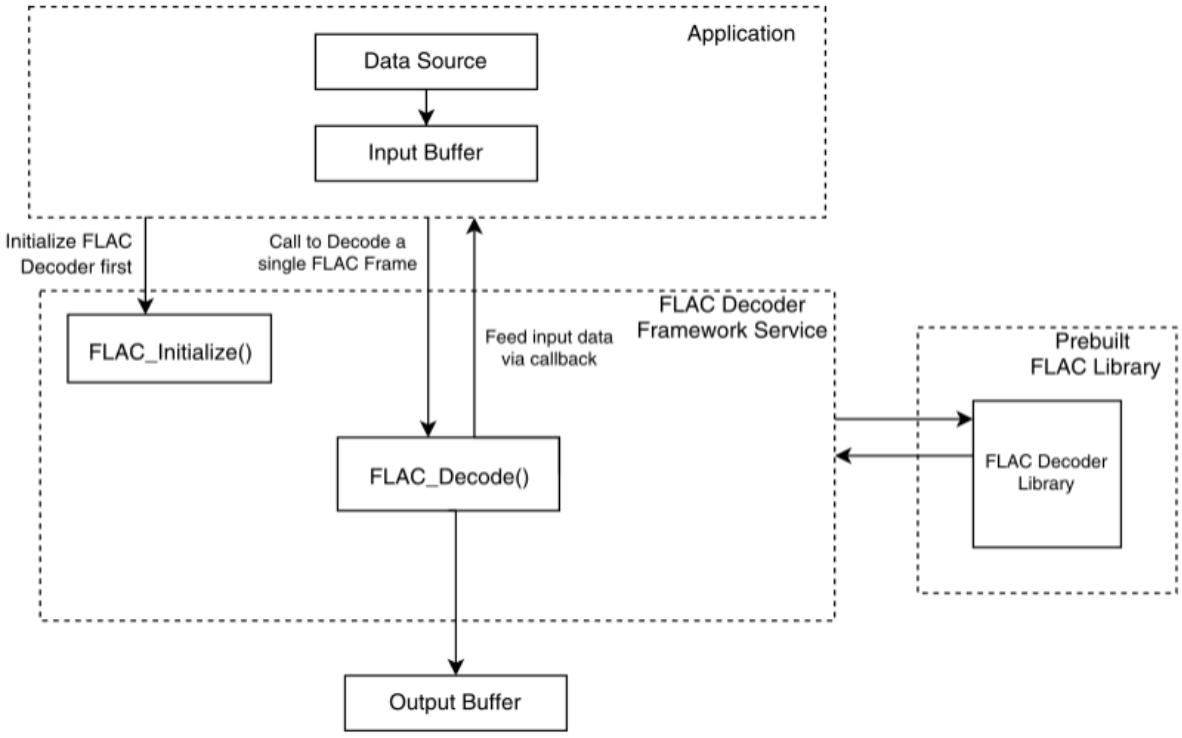
The Library Interface routines are divided into various sub-sections, each of which addresses one of the blocks or the overall operation of the FLAC Decoder Library module.

How the Library Works

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

Description

The following diagram describes the data flow between the FLAC Decoder and an application.



Initializing the FLAC Decoder

Initialize the FLAC Decoder by calling the function FLAC_Initialize from the `flac_dec.h` file. This function initializes the FLAC Decoder state structure, and sets callback functions for reading FLAC data. The application can start to decode FLAC frame after initialization.

Decoding a FLAC Frame

The FLAC_Decoder function in `flac_dec.h` is used to decode a single FLAC frame, and write back decoded data in an output buffer.

Code Example

This FLAC decoder framework service provides general APIs for decoder application use. The following code example shows typical usage of this FLAC framework service.

```
[Code Example]

void DECODER_Initialize(decoder_type){
        switch(decoder_type){
            case FLAC:
       if(FLAC_Initialize (appDataPtr->fileHandle, appDataPtr->fileStatus.lfname) == true)
       {
        // setup other components,
        // for example, audio CODEC, display, etc...
          }
        }
}

void App_Task()
{
         while(audio is not end){
    // Read one Opus packet
    appData.nBytesRead = SYS_File_Read(file_ptr, input_ptr, blocksize);
    // Decode one packet
           ret = FLAC_Decoder (input_ptr,inSize,read,output,written);
    if(ret == success)
    {
         // continue decoding
    }else{
        // handle decoding errors;
    }
         }
         // Clean up
         FLAC_Cleanup();
     }
}
```

Library Interface

a) Functions

	Name	Description
	FLAC_Cleanup	A clean up function for deallocating memory resources of a FLAC decoder.
	FLAC_Decoder	An abstraction function on FLAC decoder library, this function decodes a chuck of input data and returns the decoded data.
	FLAC_GetBitRate	Returns bit rate of the FLAC audio file.
	FLAC_GetBlockSize	Returns size of next packet to be decoded.
	FLAC_GetChannels	Returns number of channel of the FLAC file.
	FLAC_GetSamplingRate	Returns sample rate of the FLAC audio file.
	FLAC_RegisterDecoderEventHandlerCallback	Register a decoder event handler function to FLAC decoder.
	isFLACdecoder_enabled	Return a boolean if the FLAC decoder is included or not.
	FLAC_GetBitdepth	Returns bitdepth of the FLAC audio file.
	FLAC_GetDuration	Returns track length of the FLAC audio file.
	FLAC_Initialize	An abstraction function on FLAC decoder library, initialize necessary FLAC decoder state variables.

b) Data Types and Constants

	Name	Description
	FLAC_DEC_H	This is macro FLAC_DEC_H.

Description

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

Refer to each section for a detailed description.

a) Functions

FLAC_Cleanup Function

A clean up function for deallocating memory resources of a FLAC decoder.

File

flac_dec.h

C

```
void FLAC_Cleanup();
```

Returns

Size of next packet to be decoded.

Description

Function FLAC_Cleanup:

void FLAC_Cleanup();

This function must be called after a FLAC audio file is decoded to free the memory resources.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
blocksize = FLAC_GetBlockSize();
SYS_FS_FileRead(fptr, input, blocksize);
```

FLAC_Decoder Function

An abstraction function on FLAC decoder library, this function decodes a chuck of input data and returns the decoded data.

File

flac_dec.h

C

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

Returns

output buffer pointer which holds decoded data. written - size of decoded data.

This function returns a boolean value.

- 0 - FLAC decoder decodes failed.
- 1 - FLAC decoder decodes succeed.

Description

Function FLAC_Decoder:

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

This function decodes input buffer then returns decoded data, it provides a abstraction interface to use FLAC decode functions from library.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
SYS_FS_FileRead(fptr, input, inSize);
FLAC_Decoder(input, inSize, read, output, written);
```

Parameters

Parameters	Description
inSize	input buffer size
read	size of input buffer that is read by this function.

FLAC_GetBitRate Function

Returns bit rate of the FLAC audio file.

File

flac_dec.h

C

```
int32_t FLAC_GetBitRate();
```

Returns

Bit rate of the FLAC audio file.

Description

Function FLAC_GetBitRate:

int32_t FLAC_GetBitRate();

This function returns bit rate of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
bitrate = FLAC_GetBitRate();
```

FLAC_GetBlockSize Function

Returns size of next packet to be decoded.

File

flac_dec.h

C

```
int32_t FLAC_GetBlockSize();
```

Returns

Size of next packet to be decoded.

Description

Function FLAC_GetBlockSize:

int32_t FLAC_GetBlockSize();

This function returns size of next packet to be decoded.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
blocksize = FLAC_GetBlockSize();
SYS_FS_FileRead(fptr, input, blocksize);
```

FLAC_GetChannels Function

Returns number of channel of the FLAC file.

File

flac_dec.h

C

```
uint8_t FLAC_GetChannels();
```

Returns

Number of audio channels

Description

Function FLAC_GetChannels:

uint8_t FLAC_GetChannels();

This function returns audio channel number of the FLAC file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
num_channel = FLAC_GetChannels();
```

FLAC_GetSamplingRate Function

Returns sample rate of the FLAC audio file.

File

flac_dec.h

C

```
int32_t FLAC_GetSamplingRate();
```

Returns

Sample rate of the FLAC audio file.

Description

Function FLAC_GetSamplingRate:

int32_t FLAC_GetSamplingRate();

This function returns sample rate of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
samplerate = FLAC_GetSamplingRate();
```

FLAC_RegisterDecoderEventHandlerCallback Function

Register a decoder event handler function to FLAC decoder.

File

flac_dec.h

C

```
void FLAC_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);
```

Returns

None.

Description

Function FLAC_RegisterDecoderEventHandlerCallback:

void FLAC_RegisterDecoderEventHandlerCallback(DecoderEventHandlerCB fptr);

This function registers a event handler function for propagating FLAC decoding information to a upper level.

Remarks

None.

Preconditions

None.

Example

```
FLAC_RegisterDecoderEventHandlerCallback(fptr);
```

Parameters

Parameters	Description
fptr	event handler function pointer.

isFLACdecoder_enabled Function

Return a boolean if the FLAC decoder is included or not.

File

flac_dec.h

C

```
bool isFLACdecoder_enabled();
```

Returns

This function returns a boolean value.

- 0 - FLAC decoder is not enabled.
- 1 - FLAC decoder is enabled.

Description

Function isFLACdecoder_enabled:

bool isFLACdecoder_enabled();

Return a boolean if the FLAC decoder is included or not.

Remarks

None.

Preconditions

None.

Example

```
bool flac_enabled = isFLACdecoder_enabled();
```

FLAC_GetBitdepth Function

Returns bitdepth of the FLAC audio file.

File

flac_dec.h

C

```
uint8_t FLAC_GetBitdepth();
```

Returns

Bitdepth of the FLAC audio file.

Description

Function FLAC_GetBitdepth:

uint8_t FLAC_GetBitdepth();

This function returns bitdepth of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
bitdepth = FLAC_GetBitdepth();
```

FLAC_GetDuration Function

Returns track length of the FLAC audio file.

File

flac_dec.h

C

```
uint32_t FLAC_GetDuration();
```

Returns

track length of the FLAC audio file.

Description

Function FLAC_GetDuration:

uint32_t FLAC_GetDuration();

This function returns track length of the FLAC audio file.

Remarks

None.

Preconditions

FLAC_Initialize function must be called before this function.

Example

```
length_in_seconds = FLAC_GetDuration();
```

FLAC_Initialize Function

An abstraction function on FLAC decoder library, initialize necessary FLAC decoder state variables.

File

flac_dec.h

C

```
bool FLAC_Initialize(SYS_FS_HANDLE fhandle, void * input_buffer);
```

Returns

This function returns a boolean value.

- 0 - FLAC decoder initialization failed.
- 1 - FLAC decoder initialization succeed.

Description

Function FLAC_Initialize:

bool FLAC_Initialize (SYS_FS_HANDLE fhandle, char *file);

This function provides decoder initialize function to audio applications.

Remarks

None.

Preconditions

A FLAC audio file.

Example

```
bool flac_init_ret = bool FLAC_Initialize (flacHandle, input_buffer);
```

Parameters

Parameters	Description
fhandle	file handle to a FLAC audio file
input_buffer	input buffer pointer, size must be larger enough to hold maximum framesize of the flac file.

b) Data Types and Constants

FLAC_DEC_H Macro

File

flac_dec.h

C

```
#define FLAC_DEC_H
```

Description

This is macro FLAC_DEC_H.

Files

Files

Name	Description
flac_dec.h	This header file consists of flac library interface function declarations.

Description

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

flac_dec.h

This header file consists of flac library interface function declarations.

Functions

	Name	Description
	FLAC_Cleanup	A clean up function for deallocating memory resources of a FLAC decoder.

	FLAC_Decoder	An abstraction function on FLAC decoder library, this function decodes a chuck of input data and returns the decoded data.
	FLAC_GetBitdepth	Returns bitdepth of the FLAC audio file.
	FLAC_GetBitRate	Returns bit rate of the FLAC audio file.
	FLAC_GetBlockSize	Returns size of next packet to be decoded.
	FLAC_GetChannels	Returns number of channel of the FLAC file.
	FLAC_GetDuration	Returns track length of the FLAC audio file.
	FLAC_GetSamplingRate	Returns sample rate of the FLAC audio file.
	FLAC_Initialize	An abstraction function on FLAC decoder library, initialize necessary FLAC decoder state variables.
	FLAC_RegisterDecoderEventHandlerCallback	Register a decoder event handler function to FLAC decoder.
	isFLACdecoder_enabled	Return a boolean if the FLAC decoder is included or not.

Macros

	Name	Description
	FLAC_DEC_H	This is macro FLAC_DEC_H.

Description

FLAC Decoder Library Interface File

FLAC decoder interface function declarations, it is not library source code of flac, it is interface functions for easily use in Harmony audio framework.

File Name

flac_dec.h

Company

Microchip Technology Inc.

MP3 Decoder Library

This section describes the MP3 Decoder Library.

Introduction

MP3 is a lossy data compression coding format. MP3 was designed by the Moving Picture Experts Group (MPEG) as part of its MPEG-1 standard and later extended in the MPEG-2 standard. The MPLAB Harmony MP3 Decoder Library provides not only MPEG-1 standard support, but also MPEG-2 and MPEG-2.5 profile support for decoding low sampling frequency.

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.

Features

- Support for decoding MPEG-1, MPEG-2 and MPEG-2.5 encoded files
- Supports variable bit rate (VBR)
- Supports sampling frequencies from 8 kHz to 48 kHz
- Supports bit rates from 8 kbps to 320 kbps.

Library Performance

The following matrix was measured on a PIC32MX family device.

	Performance Statistics (Average)	Memory Statistics	
MP3 44.1 kHz Test Vector	MIPS	Data RAM	Flash Memory
	30	19.85 KB	47 KB

Input buffer for one MP3 Frame: 1538 byte for most cases
(for high sample frequency and bit rate audio data, 6144 byte is recommended)
Output buffer: 4608 byte for an 1152 samples, stereo 16-bit audio

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.

Abstraction Model

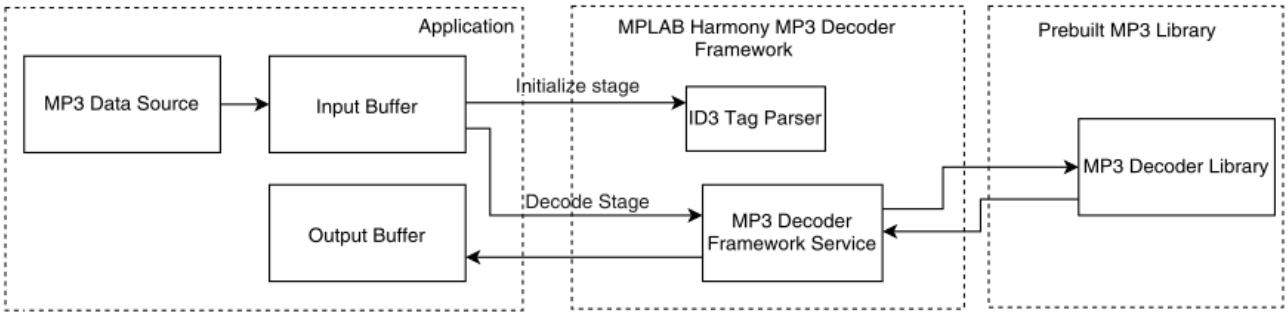
Describes the abstraction model for the MP3 Decoder Library.

Description

The following block diagram describes how the MP3 Decoder Library interacts with an application.

The MP3 Decoder Framework Service is another layer of abstraction of MP3 decoder library, it not only decodes MP3 frame, but also has a ID3 tag parser engine.

Abstraction Model



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.
	MP3_EventHandler	This is function MP3_EventHandler.
	MP3_ParseVBR	This is function MP3_ParseVBR.
	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.
	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_IN_FRAME_SIZE	1538
	MP3_OUT_FRAME_SIZE	This is macro MP3_OUT_FRAME_SIZE.
	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_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.
	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.
	MP3_DEC_H	This is macro MP3_DEC_H.

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_dec.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.

MP3_EventHandler Function

File

mp3_dec.h

C

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

Description

This is function MP3_EventHandler.

MP3_ParseVBR Function

File

mp3_dec.h

C

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

Description

This is function MP3_ParseVBR.

isMP3decoder_enabled Function

File

mp3_dec.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.

MP3_GetAudioSize Function

File

mp3_dec.h

C

```
uint32_t MP3_GetAudioSize();
```

Description

This is function MP3_GetAudioSize.

MP3_UpdatePlaytime Function

File

mp3_dec.h

C

```
uint32_t MP3_UpdatePlaytime();
```

Description

This is function MP3_UpdatePlaytime.

MP3_Initialize Function

File

mp3_dec.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_dec.h

C

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

Description

This is function MP3_RegisterDecoderEventHandlerCallback.

MP3_GetChannels Function

File

mp3_dec.h

C

```
uint8_t MP3_GetChannels();
```

Description

This is function MP3_GetChannels.

b) Data Types and Constants

MP3_IN_FRAME_SIZE Macro

File

mp3_dec.h

C

```
#define MP3_IN_FRAME_SIZE 6144//1538
```

Description

1538

MP3_OUT_FRAME_SIZE Macro

File

mp3_dec.h

C

```
#define MP3_OUT_FRAME_SIZE 1152 * 4
```

Description

This is macro MP3_OUT_FRAME_SIZE.

MP3_STATE_SIZE Macro

File

mp3_dec.h

C

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

Description

11024

MP3_EVENT Enumeration

File

mp3_dec.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_dec.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_dec.h

C

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

Description

This is type MP3_STATE.

MP3_XING_HEADER Structure

File

mp3_dec.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_dec.h

C

```
#define MP3_ERROR_COUNT_MAX 1
```

Description

This is macro MP3_ERROR_COUNT_MAX.

MP3_HEADER_CHANNELS_DUAL Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_DUAL 0b10
```

Description

This is macro MP3_HEADER_CHANNELS_DUAL.

MP3_HEADER_CHANNELS_JOINT Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_JOINT 0b01
```

Description

This is macro MP3_HEADER_CHANNELS_JOINT.

MP3_HEADER_CHANNELS_MONO Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_MONO 0b11
```

Description

This is macro MP3_HEADER_CHANNELS_MONO.

MP3_HEADER_CHANNELS_STEREO Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_CHANNELS_STEREO 0b00
```

Description

This is macro MP3_HEADER_CHANNELS_STEREO.

MP3_HEADER_SAMPLERATE_32000 Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_32000 0b10
```

Description

This is macro MP3_HEADER_SAMPLERATE_32000.

MP3_HEADER_SAMPLERATE_44100 Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_44100 0b00
```

Description

This is macro MP3_HEADER_SAMPLERATE_44100.

MP3_HEADER_SAMPLERATE_48000 Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_48000 0b01
```

Description

This is macro MP3_HEADER_SAMPLERATE_48000.

MP3_HEADER_SAMPLERATE_RESV Macro

File

mp3_dec.h

C

```
#define MP3_HEADER_SAMPLERATE_RESV 0b11
```

Description

This is macro MP3_HEADER_SAMPLERATE_RESV.

MP3_XING_HEADER_START_MONO Macro

File

mp3_dec.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_dec.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.

MP3MPEG1L3_SAMPLES_PER_FRAME Macro

File

mp3_dec.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_dec.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_dec.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_dec.h

C

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

Description

This is type DecoderEventHandlerCB.

MP3_DEC_H Macro

File

mp3_dec.h

C

```
#define MP3_DEC_H
```

Description

This is macro MP3_DEC_H.

Files

Files

Name	Description
id3.h	MP3 Decoder ID3 parser header API.
mp3_dec.h	MP3 Decoder support API.

Description

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

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 ID3 Tag Parser Header File

This file provides MP3 Decoder ID3 parser header APIs.

File Name

id3.h

Company

Microchip Technology Inc.

mp3_dec.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_DEC_H	This is macro MP3_DEC_H.
	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_IN_FRAME_SIZE	1538
	MP3_OUT_FRAME_SIZE	This is macro MP3_OUT_FRAME_SIZE.
	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.

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.

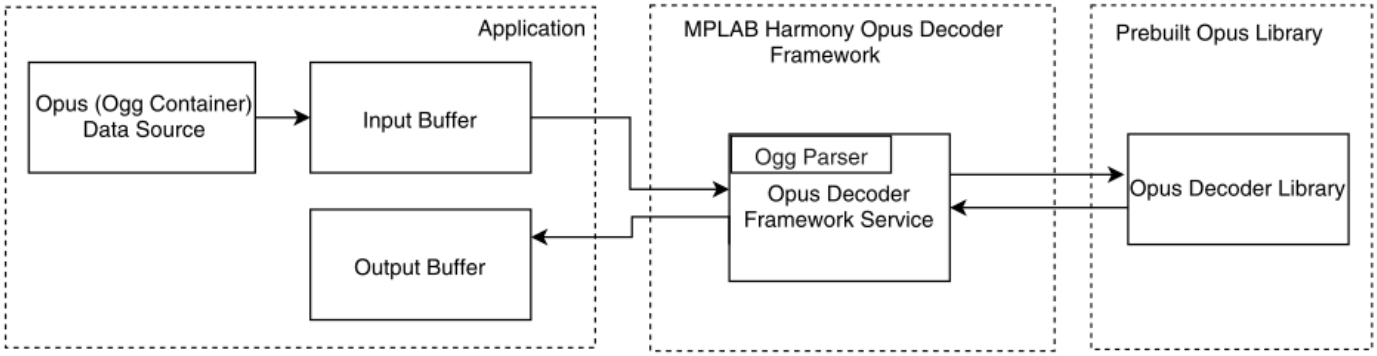
Abstraction Model

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



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.

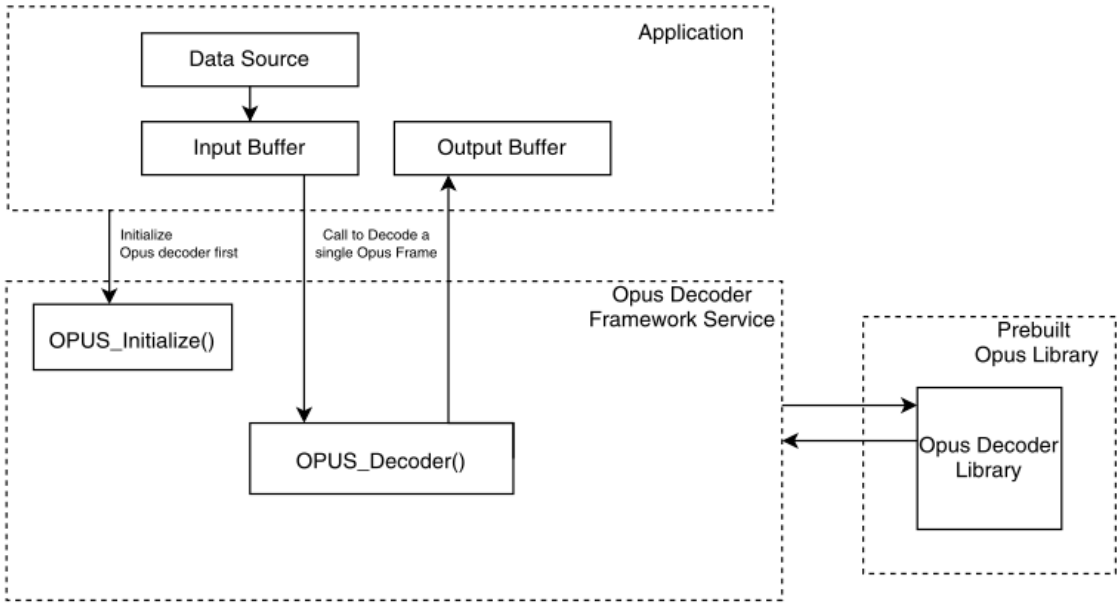
How the Library Works

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.

The following diagram describes the data flow between the Opus Decoder and an application.



Initializing the Opus Decoder

Initialize the Opus Decoder by calling the function OPUS_Initialize from the `opus_dec.h` file. This function initializes the Opus Decoder state structure. The application can start to decode Opus frame after initialization.

Decoding an Opus Frame

The OPUS_Decoder function in opus_dec.h is used to decode a single Opus frame, and write back decoded data into an output buffer.

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
	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
	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
	OPUS_INPUT_BUFFER_SIZE	MACROS
	OPUS_MAX_FRAME_SIZE	120ms @ 48Khz

	OPUS_OUTPUT_BUFFER_SIZE	This is macro OPUS_OUTPUT_BUFFER_SIZE.
	DECODER_OPUS_DEC_SUPPORT_H	This is macro DECODER_OPUS_DEC_SUPPORT_H.

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

isOPUSdecoder_enabled Function

Checks if Opus decoder is enabled by MHC configuration.

File

opus_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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

OPUS_ERROR_MSG Enumeration

File

opus_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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

OPUS_INPUT_BUFFER_SIZE Macro

File

opus_dec.h

C

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

Description

MACROS

OPUS_MAX_FRAME_SIZE Macro

File

opus_dec.h

C

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

Description

120ms @ 48Khz

OPUS_OUTPUT_BUFFER_SIZE Macro

File

opus_dec.h

C

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

Description

This is macro OPUS_OUTPUT_BUFFER_SIZE.

DECODER_OPUS_DEC_SUPPORT_H Macro

File

opus_dec.h

C

```
#define DECODER_OPUS_DEC_SUPPORT_H
```

Description

This is macro DECODER_OPUS_DEC_SUPPORT_H.

Files

Files

Name	Description
opus_dec.h	Contains the Opus decoder specific defintions and function prototypes.

Description

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

opus_dec.h

Contains the Opus decoder specific defintions and function prototypes.

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_DEC_SUPPORT_H	This is macro DECODER_OPUS_DEC_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

Ogg-Opus Decoder Interface File

This file contains the opus decoder specific defintions and function prototypes.

File Name

opus_dec.h

Company

Microchip Technology Inc.

Speex Decoder Library

This section describes the Speex Decoder Library.

Introduction

Speex is an Open Source/Free Software patent-free audio compression format designed for speech. It is designed by the Xiph.Org Foundation, and has been obsoleted by Opus.

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.

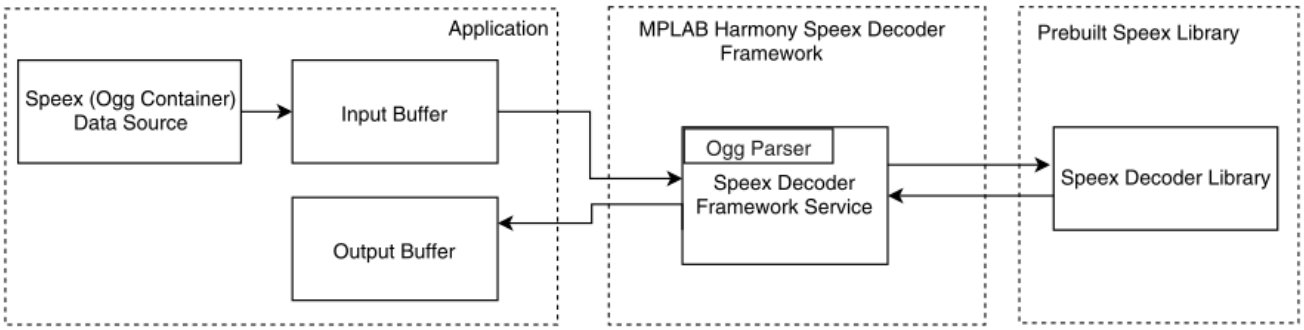
Abstraction Model

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



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.

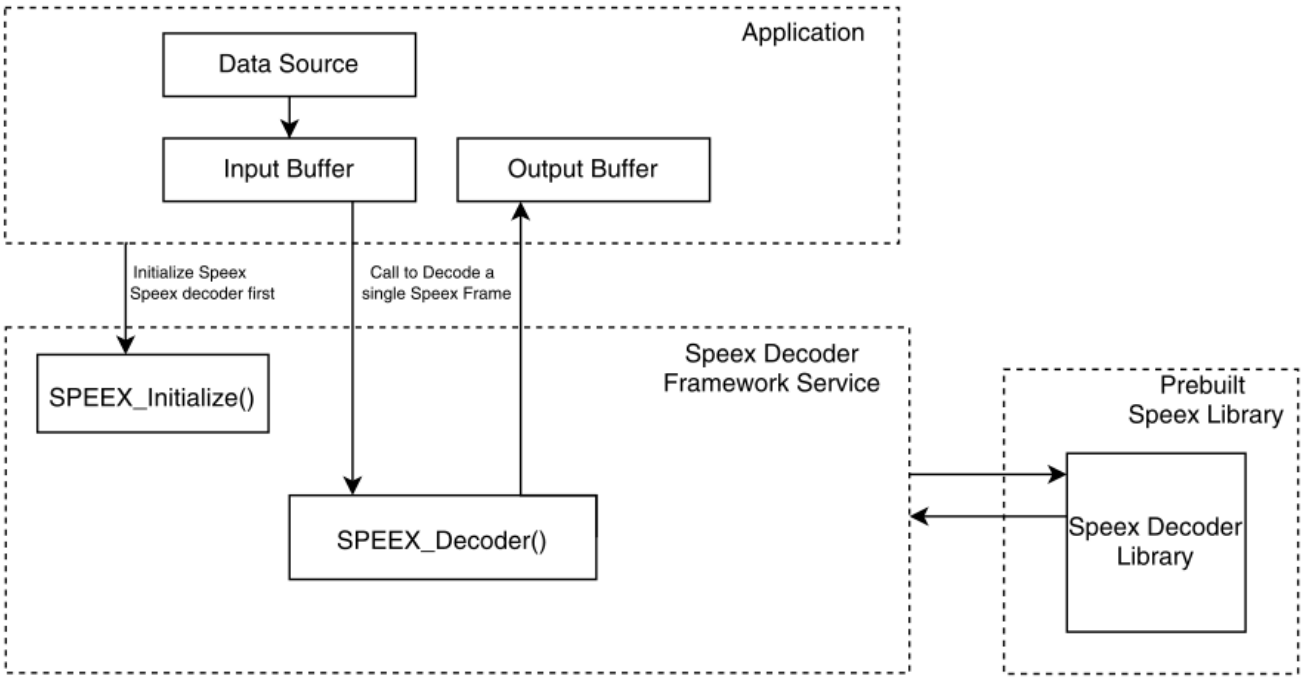
How the Library Works

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.

The following diagram describes the data flow between the Speex Decoder and an application.



Initializing the Speex Decoder

Initialize the Speex Decoder by calling the function SPEEX_Initialize from the `speex_dec.h` file. This function initializes the Speex Decoder state structure. The application can start to decode a Speex frame after initialization.

Decoding a Speex Frame

The SPEEX_Decoder function in `speex_dec.h` decodes a single Speex frame, and writes back decoded data into an output buffer.

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.
	SPEEX_GetChannels	Returns channel number of Ogg-Speex audio.

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_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_dec.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_dec.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_dec.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_dec.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_dec.h

C

```
int32_t SPEEX_GetBitrate();
```

Returns

This function returns the bit rate.

Description

This function returns the bit 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_dec.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_dec.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);

SPEEX_GetChannels Function

Returns channel number of Ogg-Speex audio.

File

speex_dec.h

C

```
uint8_t SPEEX_GetChannels();
```

Returns

This function returns channel number

Description

This function returns the channel number of Ogg-Speex audio.

Preconditions

This function can only be called after the SPEEX_Initialize function.

Example

```
channelNumber = SPEEX_GetChannels();
```

Function

uint8_t SPEEX_GetChannels()

b) Data Types and Constants

eSpxFlags Enumeration

File

speex_dec.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_dec.h

C

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

Description

progress display function

progressDcpt Structure

File

speex_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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_dec.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_dec.h

C

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

Description

The Speex packet ID

SPEEX_INPUT_BUFFER_SIZE Macro

File

speex_dec.h

C

```
#define SPEEX_INPUT_BUFFER_SIZE 1024
```

Description

This is macro SPEEX_INPUT_BUFFER_SIZE.

SPEEX_OUTPUT_BUFFER_SIZE Macro

File

speex_dec.h

C

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

Description

This is macro SPEEX_OUTPUT_BUFFER_SIZE.

SPEEX_STRING_LENGTH Macro

File

speex_dec.h

C

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

Description

The size of the Speex string

SPEEX_VENDOR_STR Macro

File

speex_dec.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_dec.h

C

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

Description

The Speex version string

SPEEX_VERSION_ID Macro

File

speex_dec.h

C

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

Description

Version identifier

SPEEX_VERSION_LENGTH Macro

File

speex_dec.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_dec.h

C

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

Description

20% is sufficient

SPX_CODEC_BUFF_MUL Macro

File

speex_dec.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_dec.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_dec.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_GetChannels	Returns channel number 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_DEC_SUPPORT_H	This is macro SPEEX_DEC_SUPPORT_H.
	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_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_dec.h

Company

Microchip Technology Inc.

WMA Decoder Library

This section describes the WMA Decoder Library.

Introduction

Windows Media Audio (WMA) is the name of a series of Audio Codecs and their corresponding audio coding formats developed by Microsoft. It is a proprietary technology that forms part of the Windows Media framework.

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.

Features

- Supports both variable bit rate (VBR) and constant bit rate (CBR)
- Supports WMA version v9.2, v9.1, v9, v8, v7, v4.1, v4.0
- Supports sampling frequency range from 8 kHz to 48 kHz
- Supports bit rate range from 128 bps to 384 kbps

Library Performance

	Performance Statistics (Average)	Memory Statistics	
WMA 44.1 kHz, 192 kbps Test Vector	MIPS	Data RAM	Flash Memory
	35	32.23KB	102.72KB

Input buffer for one WMA frame: 20 Kbytes

Output buffer: 12 Kbytes for stereo 16-bit audio data

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 WMA 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.

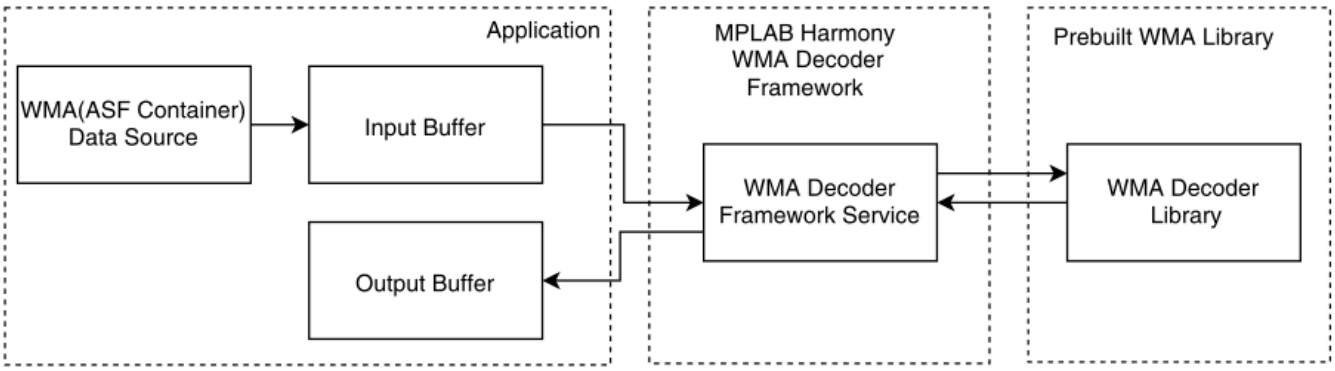
Abstraction Model

Describes the abstraction model for the WMA Decoder Library.

Description

The following figure describes the abstraction model for the WMA Decoder Library.

Abstraction Model



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
	WMA_Decoder	This is function WMA_Decoder.
	WMA_SamplingFrequency_Get	This is function WMA_SamplingFrequency_Get.
	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.
	GetReadBytesInAppData	This is type GetReadBytesInAppData.
	SetReadBytesReadFlagInAppData	This is type SetReadBytesReadFlagInAppData.
	SPEEX_DEC_SUPPORT_H	This is macro SPEEX_DEC_SUPPORT_H.

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

WMA_Decoder Function

File

wma_dec.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_dec.h

C

```
int32_t WMA_SamplingFrequency_Get();
```

Description

This is function WMA_SamplingFrequency_Get.

WMA_FreeMemory Function

File

wma_dec.h

C

```
void WMA_FreeMemory();
```

Description

This is function WMA_FreeMemory.

isWMAdecoder_enabled Function

File

wma_dec.h

C

```
bool isWMAdecoder_enabled();
```

Description

This is function isWMAdecoder_enabled.

WMA_BitRate_Get Function

File

wma_dec.h

C

```
int32_t WMA_BitRate_Get();
```

Description

This is function WMA_BitRate_Get.

WMA_GetHeaderPacketOffset Function

File

wma_dec.h

C

```
int32_t WMA_GetHeaderPacketOffset();
```

Description

This is function WMA_GetHeaderPacketOffset.

WMA_Initialize Function

File

wma_dec.h

C

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

Description

This is function WMA_Initialize.

WMA_RegisterAppCallback Function

File

wma_dec.h

C

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

Description

This is function WMA_RegisterAppCallback.

WMA_GetChannels Function

File

wma_dec.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_dec.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_dec.h

C

```
#define WMA_ERROR_COUNT_MAX 1
```

Description

This is macro WMA_ERROR_COUNT_MAX.

WMA_H Macro

File

wma_dec.h

C

```
#define WMA_H
```

Description

This is macro WMA_H.

GetReadBytesInAppData Type

File

wma_dec.h

C

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

Description

This is type GetReadBytesInAppData.

SetReadBytesReadFlagInAppData Type

File

wma_dec.h

C

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

Description

This is type SetReadBytesReadFlagInAppData.

SPEEX_DEC_SUPPORT_H Macro

File

speex_dec.h

C

```
#define SPEEX_DEC_SUPPORT_H
```

Description

This is macro SPEEX_DEC_SUPPORT_H.

Files

Files

Name	Description
wma_dec.h	WMA Decoder support API.

Description

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

wma_dec.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.

Index

O

P

S

U

V

W