Fraunhofer
IIS

# Advanced Audio Coding Decoder Library

MPEG-2 and MPEG-4

AAC Low-Complexity (AAC-LC),

High-Efficiency AAC v2 (HE-AAC v2),

AAC Low-Delay (AAC-LD), and

AAC Enhanced Low-Delay (AAC-ELD)

decoder

**Fraunhofer Institut Integrierte Schaltungen IIS,**
**Fraunhofer Institute for Integrated Circuits IIS**
http://www.iis.fraunhofer.de/amm

Revision 2.5.7 , August 15, 2013

# Contents

# Chapter 1

# Introduction

## 1.1 Scope

This document describes the high-level interface and usage of the ISO/MPEG-2/4 AAC Decoder library developed by the Fraunhofer Institute for Integrated Circuits (IIS). Depending on the library configuration, it implements decoding of AAC-LC (Low-Complexity), HE-AAC (High-Efficiency AAC, v1 and v2), AAC-LD (Low-Delay) and AAC-ELD (Enhanced Low-Delay).

All references to SBR (Spectral Band Replication) are only applicable to HE-AAC and AAC-ELD versions of the library. All references to PS (Parametric Stereo) are only applicable to HE-AAC v2 versions of the library.

## 1.2 Decoder Basics

This document can only give a rough overview about the ISO/MPEG-2 and ISO/MPEG-4 AAC audio coding standard. To understand all the terms in this document, you are encouraged to read the following documents.

- ISO/IEC 13818-7 (MPEG-2 AAC), which defines the syntax of MPEG-2 AAC audio bitstreams.
- ISO/IEC 14496-3 (MPEG-4 AAC, subpart 1 and 4), which defines the syntax of MPEG-4 AAC audio bitstreams.
- Lutzky, Schuller, Gayer, Krämer, Wabnik, "A guideline to audio codec delay", 116th AES Convention, May 8, 2004

MPEG Advanced Audio Coding is based on a time-to-frequency mapping of the signal. The signal is partitioned into overlapping portions and transformed into frequency domain. The spectral components are then quantized and coded.

An MPEG2 or MPEG4 AAC audio bitstream is composed of frames. Contrary to MPEG-1/2 Layer-3 (mp3), the length of individual frames is not restricted to a fixed number of bytes, but can take on any length between 1 and 768 bytes.

# Chapter 2

# Library Usage

## 2.1 API Description

All API header files are located in the folder /include of the release package. They are described in detail in this document. All header files are provided for usage in C/C++ programs. The AAC decoder library API functions are located at aacdecoder_lib.h.

In binary releases the decoder core resides in statically linkable libraries called for example libAACdec.a, (Linux) or FDK_aacDec_lib (Microsoft Visual C++).

## 2.2 Calling Sequence

For decoding of ISO/MPEG-2/4 AAC or HE-AAC v2 bitstreams the following sequence is mandatory. Input read and output write functions as well as the corresponding open and close functions are left out, since they may be implemented differently according to the user's specific requirements. The example implementation in main.cpp uses file-based input/output, and in such case call mpegFileRead_Open() to open an input file and to allocate memory for the required structures, and the corresponding mpegFileRead_Close() to close opened files and to de-allocate associated structures. mpegFileRead_Open() tries to detect the bitstream format and in case of MPEG-4 file format or Raw Packets file format (a Fraunhofer IIS proprietary format) reads the Audio Specific Config data (ASC). An unsuccessful attempt to recognize the bitstream format requires the user to provide this information manually (see CommandLineUsage). For any other bitstream formats that are usually applicable in streaming applications, the decoder itself will try to synchronize and parse the given bitstream fragment using the FDK transport library. Hence, for streaming applications (without file access) this step is not necessary.

1. Call aacDecoder_Open() to open and retrieve a handle to a new AAC decoder instance.

   ```
   aacDecoderInfo = aacDecoder_Open(mpegFileRead_GetTransportType(hDataSrc), nrOfL
       ayers);
   ```

2. If out-of-band config data (Audio Specific Config (ASC) or Stream Mux Config (SMC)) is available, call aacDecoder_ConfigRaw() to pass it to the decoder and before the decoding process starts. If this data is not available in advance, the decoder will get it from the bitstream and configure itself while decoding with aacDecoder_DecodeFrame().

3. Begin decoding loop.

```
do {
```

4. Read data from bitstream file or stream into a client-supplied input buffer ("inBuffer" in main.cpp). If it is very small like just 4, aacDecoder_DecodeFrame() will repeatedly return AAC_DEC_NOT_-ENOUGH_BITS until enough bits were fed by aacDecoder_Fill(). Only read data when this buffer has completely been processed and is then empty. For file-based input execute mpegFileRead_Read() or any other implementation with similar functionality.

5. Call aacDecoder_Fill() to fill the decoder's internal bitstream input buffer with the client-supplied external bitstream input buffer.

   ```
   aacDecoder_Fill(aacDecoderInfo, inBuffer, bytesRead, bytesValid);
   ```

6. Call aacDecoder_DecodeFrame() which writes decoded PCM audio data to a client-supplied buffer. It is the client's responsibility to allocate a buffer which is large enough to hold this output data.

   ```
   ErrorStatus = aacDecoder_DecodeFrame(aacDecoderInfo, TimeData, OUT_BUF_SIZE,
     flags);
   ```

   If the bitstream's configuration (number of channels, sample rate, frame size) is not known in advance, you may call aacDecoder_GetStreamInfo() to retrieve a structure containing this information and then initialize an audio output device. In the example main.cpp, if the number of channels or the sample rate has changed since program start or since the previously decoded frame, the audio output device will be re-initialized. If WAVE file output is chosen, a new WAVE file for each new configuration will be created.

7. Repeat steps 5 to 7 until no data to decode is available anymore, or if an error occured.

8. Call aacDecoder_Close() to de-allocate all AAC decoder and transport layer structures.

## 2.3 Buffer System

There are three main buffers in an AAC decoder application. One external input buffer to hold bitstream data from file I/O or elsewhere, one decoder-internal input buffer, and one to hold the decoded output PCM sample data, whereas this output buffer may overlap with the external input buffer.

The external input buffer is set in the example framework main.cpp and its size is defined by IN_BUF_-SIZE. You may freely choose different sizes here. To feed the data to the decoder-internal input buffer, use the function aacDecoder_Fill(). This function returns important information about how many bytes in the external input buffer have not yet been copied into the internal input buffer (variable bytesValid). Once the external buffer has been fully copied, it can be re-filled again. In case you want to re-fill it when there are still unprocessed bytes (bytesValid is unequal 0), you would have to additionally perform a memcpy(), so that just means unnecessary computational overhead and therefore we recommend to re-fill the buffer only when bytesValid is 0.

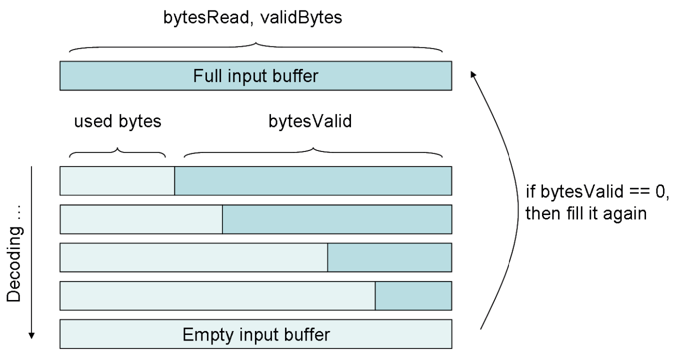

Figure 2.1: Lifecycle of the external input buffer

The size of the decoder-internal input buffer is set in tpdec_lib.h (see define TRANSPORTDEC_INBUF_-SIZE). You may choose a smaller size under the following considerations:

- each input channel requires 768 bytes
- the whole buffer must be of size 2^n

So for example a stereo decoder:

$$TRANSPORTDEC\_INBUF\_SIZE = 2 * 768 = 1536 => 2048$$

tpdec_lib.h and TRANSPORTDEC_INBUF_SIZE are not part of the decoder's library interface. Therefore only source-code clients may change this setting. If you received a library release, please ask us and we can change this in order to meet your memory requirements.

# Chapter 3

# Decoder audio output

## 3.1 Obtaining channel mapping information

The decoded audio output format is indicated by a set of variables of the CStreamInfo structure. While the members sampleRate, frameSize and numChannels might be quite self explaining, pChannelType and pChannelIndices might require some more detailed explanation.

These two arrays indicate what is each output channel supposed to be. Both array have CStream-Info::numChannels cells. Each cell of pChannelType indicates the channel type, described in the enum AUDIO_CHANNEL_TYPE defined in FDK_audio.h. The cells of pChannelIndices indicate the sub index among the channels starting with 0 among all channels of the same audio channel type.

The indexing scheme is the same as for MPEG-2/4. Thus indices are counted upwards starting from the front direction (thus a center channel if any, will always be index 0). Then the indices count up, starting always with the left side, pairwise from front toward back. For detailed explanation, please refer to ISO/IEC 13818-7:2005(E), chapter 8.5.3.2.

In case a Program Config is included in the audio configuration, the channel mapping described within it will be adopted.

In case of MPEG-D Surround the channel mapping will follow the same criteria described in ISO/IEC 13818-7:2005(E), but adding corresponding top channels to the channel types front, side and back, in order to avoid any loss of information.

## 3.2 Changing the audio output format

The channel interleaving scheme and the actual channel order can be changed at runtime through the parameters AAC_PCM_OUTPUT_INTERLEAVED and AAC_PCM_OUTPUT_CHANNEL_MAPPING. See the description of those parameters and the decoder library function aacDecoder_SetParam() for more detail.

## 3.3 Channel mapping examples

The following examples illustrate the location of individual audio samples in the audio buffer that is passed to aacDecoder_DecodeFrame() and the expected data in the CStreamInfo structure which can be obtained by calling aacDecoder_GetStreamInfo().