DESIGN AND IMPLEMENTATION OF AAC DECODERS

Matthew A. Watson and Peter Buettner
Dolby Laboratories, Inc.
San Francisco, CA 94103

ABSTRACT

MPEG-2 AAC is a high-quality perceptual audio coder with flexible configurations enabling its use in many applications where high audio quality and limited transmission channel bandwidth are required. This paper provides a technology overview and describes the design and implementation of a reference AAC decoder.

1. INTRODUCTION

With the increase in demand for high quality audio over narrow-bandwidth channels, new technologies have been created to distribute audio to consumers. MPEG-2 AAC (Advanced Audio Coding) incorporates several innovative technologies in order to achieve high fidelity at low bitrates [1,2].

The AAC coder has received widespread acceptance internationally and is used in many aspects of consumer electronics such as electronic music delivery, HDTV systems, and digital audio broadcasting. The AAC coder has achieved ITU-R broadcast quality at a bit-rate of 128 kb/s for stereo in an independent test [3].

The coder is comprised of three different profiles, allowing tradeoffs in audio quality and encoding/decoding complexity for different applications. AAC also supports a wide range of sample rates and data rates. An overview of the AAC coder, as well as design and implementation details of a reference AAC decoder will be discussed in this paper. An overview of the reference software architecture and its main components as well as the correspondence between the architecture and the decoding process will be given.

2. THE MPEG-2 AAC CODING SYSTEM

AAC supports up to forty-eight audio channels. Sample rates supported range from 8 kHz to 96 kHz. The coder also supports three independent modes of operation: Main, Low Complexity (LC), and Scalable Sample Rate (SSR) profiles. The Main profile delivers the highest audio quality. The LC profile achieves nearly the same audio quality as the Main profile, but with significant savings in memory and processing requirements. The SSR mode provides flexibility for scalable and low-complexity applications. With this mode, it is possible to decode the bitstream into a PCM signal having one of a variety of different sample rates. This paper will focus on the two-channel, Low Complexity profile implementation of the coder, which represents the configuration that is best suited for consumer electronics applications.

The coding system supports a variety of data rates, up to a maximum that is dependent on sample rate. The maximum data rate per channel for some popular sample rates are shown in Table 1.

Sample rate	Max bit rate/ch (kb/s)
96000	576
48000	288
44100	264.6
32000	192

Table 1: Maximum data rates

AAC encoders are able to operate in either a fixed or variable bit-rate mode. In the fixed bit-rate mode, a bit reservoir technique can optionally be employed to improve audio quality. With this technique, the AAC encoder produces a constant average bitrate, while allowing short-term variations in the size of packed audio frames. The size of the packed frames depends on signal characteristics, increasing when the signal is challenging to encode without audible distortion and decreasing otherwise. Decoder input buffer requirements limit the size of the bit reservoir.

AAC uses a combination of multiple coding tools to achieve bit rate reduction. All of the coding tools described below (except for prediction) are used in both the Main and LC profiles.

The filter bank tool for the Main and Low Complexity profiles uses a Modified Discrete Cosine Transform (MDCT) with transform length block switched between 1024 and 128 frequency points. The SSR profile uses a hybrid filter bank consisting of a Polyphase

Original manuscript received June 19, 2000

Quadrature Filter (PQF) connected in cascade with an MDCT block switched between 256 and 32 frequency points. All three profiles employ TDAC (time-domain alias cancellation) techniques. At the 48 kHz sample rate, AAC has a 23 Hz frequency resolution and a 2.5 millisecond time resolution.

The Temporal Noise Shaping (TNS) tool uses frequency-domain prediction to improve temporal resolution [5]. TNS uses prediction in the frequency domain to shape noise in the time domain. This tool reduces the need to use short blocks to achieve greater temporal resolution. While all profiles use the TNS tool, the TNS order and bandwidth is limited in the LC and SSR profiles.

The joint stereo tools exploit redundancies across channels to reduce the audio bit-rate. Three types of tools are offered: M/S (mid-side), intensity stereo, and coupling. When a coding benefit is realized, M/S transforms the left/right signal into sum and difference prior to quantization. This transformation is inverted in the decoder. With intensity stereo coding and coupling, the transform coefficients for one composite channel are transmitted in place of multiple sets of transform coefficients representing all encoder input channels. The individual spectral envelopes for all input channels are also transmitted, allowing the composite signal to be scaled appropriately to reconstruct each output channel in the decoder.

The prediction tool uses backward-adaptive prediction to remove signal redundancies contained in successive audio blocks. Since this tool has high MIPS and memory requirements it is only defined in the Main profile.

The quantizer tool quantizes the spectrum coefficients using a non-uniform step-size of 1.5 dB. This step-size is fixed within a scale factor band, but can vary from band to band. The common scale factor and the scale factor for the band determine the quantization level for a particular scale factor band.

Huffman coding is utilized for noiseless coding of the quantized spectral values. The Huffman strategy uses multiple switched codebooks with multiple dimensions to code the spectral values. One or more scale factor bands are grouped into a section and coded using the same Huffman codebook. The noiseless coding tool determines the optimum section lengths and codebooks for use in the bitstream packing module and calculates the number of bits used for spectrum encoding.

The bitstream formatter assembles the quantized and coded coefficients and the control parameters into a stream for transmission. The AAC bitstream is composed of frames with varying size, depending on the variation caused by bit-reservoir control.

The SSR profile requires a preprocessing tool, which splits the frequency domain signal into four regions and controls the gain on each region independently.

2.1 MPEG-2 AAC Bitstream Features

The length of the AAC frames varies from frame to frame because of the bit reservoir technique. Each frame represents 1024 PCM samples per channel. An illustration of the AAC bitstream frame is shown in Figure 1.

The AAC bitstream starts with a header. Two different headers are specified in the AAC standard [1], Audio Data Interchange Format (ADIF) and Audio Data Transport Stream (ADTS). Depending on the application either of these headers may be used. It is not possible to use both headers at the same time. The ADIF header is geared towards file-based applications, while the ADTS header is suited more for serial transmission protocols. The ADIF format allows for only one header per file, while the ADTS format consists of one header per AAC frame.

The ADIF header contains the following information:

- ADIF ID
- Copyright information

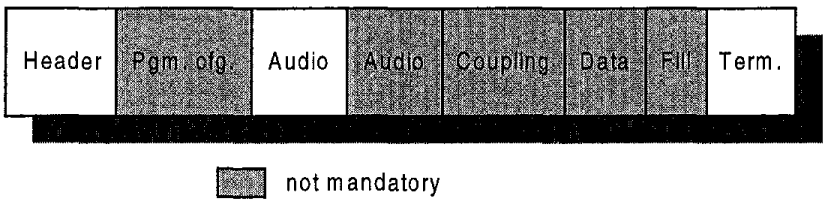

Figure 1: MPEG-2 AAC block structure

- Bitrate information
- Number of program configuration elements

The ADTS header contains the following information:

- ADTS sync word
- Audio information (sample rate, channel configuration, etc.)
- Copyright information
- CRC word

The Program Configuration Element (PCE) specifies several parameters, including the output channel assignment, sample rate, and multi-lingual configuration.

The audio element can be a Single Channel Element (SCE), Channel Pair Element (CPE), or LFE Channel Element (LFE). At least one audio element must be present in an AAC frame.

The Coupling Channel Element (CCE) is used for intensity coding of more than two input audio channels, or alternately for dialog in multilingual programs.

The Data Stream Element (DSE) can contain auxiliary non-audio information.

Fill Elements (FIL) are added to the frame if the frame size is not big enough to reach the target bit rate.

Finally, a terminator Element allows for parsing and synchronization of the bitstream.

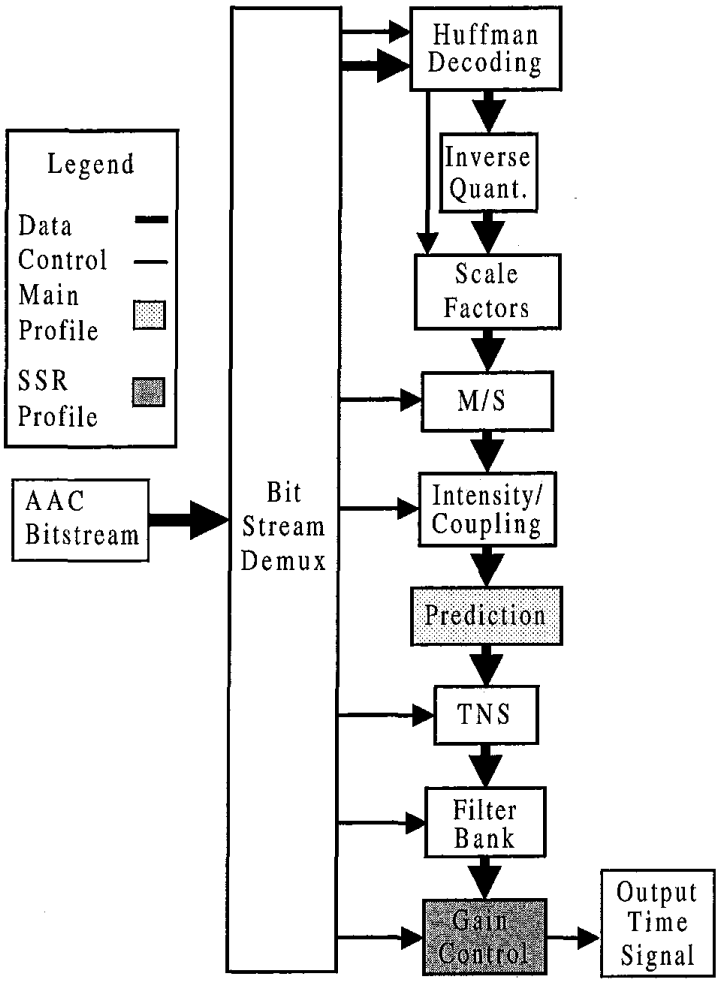

Figure 2: MPEG-2 AAC decoder diagram

3. DECODER IMPLEMENTATION STRATEGY

A basic overview of the MPEG-2 AAC decoding process is illustrated in Figure 2. The diagram and this section outline the Low Complexity profile except where noted.

The first step in decoding (bit stream de-multiplexing) is to establish frame alignment. This involves finding the AAC sync word and confirming that the AAC frame does not contain any errors.

If the AAC stream is decoded as a real-time event, an active synchronization search will be required. If the AAC stream is decoded from a file and the ADIF header is present, since there is no error detection, it is assumed that the termination element will aid with frame recovery.

Once the frame sync is found, the bitstream is de-multiplexed or unpacked. This includes unpacking of the Huffman encoded and quantized scale factors, the M/S synthesis side information, the intensity stereo side information, the TNS coefficients, the filter bank side information and the gain control words.

The coding tools are then applied to the bitstream and will be described in detail in the next section.

After the samples have been decoded, they are copied to the output buffers in a block or sample fashion.

3.1 Decoder processing

The input buffer for the decoder has a minimum size of 6144 bits per channel (384 16-bit words). For two-channel implementations, the input buffer size is 768 16-bit words. If used, the coupling channel also requires storage for 384 16-bit words per channel.

After the bitstream has been de-multiplexed within the input buffer, the quantized spectral coefficients are Huffman decoded. The decoder must traverse a Huffman code tree from the root node to the terminal node. The Huffman codes represent 2- or 4-tuple signed or unsigned spectral coefficients or escaped sequences if the coefficient is out of range for vector coding. Appropriate tables exist in ROM which match the particular Huffman scheme.

Upon Huffman decoding, each coefficient must be inverse quantized by a 4/3 power nonlinearity and then scaled by the quantizer step size. The inverse quantization may be performed by using a table lookup. At a sampling rate of 48 kHz, only 854 spectral coefficients need to be inverse quantized and scaled in order to achieve a 20 kHz bandwidth.

M/S synthesis conditionally dematrixes two channels into a stereo pair. The samples may already represent the left and right signals, in which case no computation is necessary. Otherwise, the pair must be de-matrixed via one add and one subtract per sample pair in order to retrieve the proper channel coefficients. This computation can be done in place and doesn't require any extra dynamic storage.

Intensity stereo identifies regions in a channel pair that are similar, except for their position. Left-channel intensity regions must have inverse quantization and scaling applied. Right-channel intensity stereo regions use the left-channel inverse quantized and scaled coefficients, which must be re-scaled by the intensity position factors. Hence, the net complexity of intensity stereo is a savings of one inverse quantization per intensity stereo coded coefficient (see negative MIPS value under 'Intensity Stereo' in Table 2).

Channel coupling is another tool used to reduce data. Frequency spectra at higher frequencies are combined across channel pairs, which tend to share time-energy envelopes. This information is carried in the coupling element and is scaled properly before being combined with the target channel in the frequency or time domains. In addition to providing intensity stereo coding, channel coupling also serves as a means for dynamically mixing a separate signal with a stereo signal. This application is sometimes referred to as a "voice-over".

TNS has a variable load, depending on the number of spectral coefficients that are filtered. Table 2 shows the worst-case MIPS for the TNS tool. In the LC profile, the maximum TNS filter order is 12 operating on a maximum of 683 spectral coefficients (16 kHz bandwidth) for long blocks. For short blocks, the filter order is 7. TNS decoding requires all-pole filtering selected groups of coefficients in the spectral domain. The number of filters and their ranges are defined in the bitstream.

Finally, the Inverse Modified Discrete Cosine Transform (IMDCT) transforms the spectral coefficients into time-domain samples. After the IMDCT transform, the time-domain samples are windowed, overlapped, and added for proper reconstruction. For fixed-point implementations it is generally required that any round-off noise generated by the IMDCT operation is less than ½ LSB after the transform result is rounded to linear 16-bit values. Higher width buffers throughout the time-domain sample reconstruction process will result in PCM output with greater accuracy. Fixed-point realizations using 24-bit words should be sufficient for professional applications.

Once the processing of the AAC encoded bitstream is complete, the decoder can distribute the output PCM samples from its buffers to the digital-to-analog converters.

3.2 Decoder memory and MIPS summary

The theoretical estimated memory and MIPS requirements of a floating-point implementation for a two-channel AAC decoder are displayed in Table 2 [6]. The estimates assume a 48 kHz sample rate with a bit rate of 64 kb/s for each channel. In contrast, an example of memory usage and MIPS for an actual fixed-point implementation, using a Motorola DSP, are shown in Table 3.

Tool	MIPS	RAM	ROM
Input buffers		768	
Framing, sync/error detect	2.5		
Huffman (worst case)	4.48		995
Inv. Quant. and Scaling	0.16		256
M/S Synthesis	0.04		
Intensity Stereo	-0.04		
TNS (worst case)	0.76		24
IMDCT	1.87	3072	2270
Output Buffer		2048	
Total	9.77	5888	3545

Table 2: Decoder floating point requirements by tool, 2 channels (LC profile)

MIPS	D-RAM	D-ROM	P-ROM
24	6500	5500	5000

Table 3: Decoder fixed-point implementation example, 2 channels (LC profile)

4. Reference Decoder Design

4.1 Overview

Figure 3 shows the class hierarchy of a reference C++ software for the AAC decoder. The following types of classes exist in the decoder software: interface, input, syntactic element, main data, and tools classes. The different class types are described in detail in this section.

4.2 Application Level Code

The application level code consists of the user interface and data I/O. Typically the buffers for the AAC input and the PCM output are located here. The application code passes pointers to the beginning of an AAC frame to the interface module. It also specifies the location where the interface module should place the decoded PCM data. The application code calls the decoder core once for each AAC frame.

4.3 Interface Classes

The interface classes consist of the top-level AAC decoder core object (CAacDecoder), the stream configuration and information object (CStreamInfo), and the exception handler (CAacException). These classes form the top-layer of the core decoder code and include the interface to the application-level code. At the interface the decoder core expects one AAC frame as input. It returns the decoded PCM samples in an interleaved buffer, as well as side information (sample rate, number of channels, etc.) as output. The interface classes are responsible for parsing the bitstream and calling the underlying functional blocks.

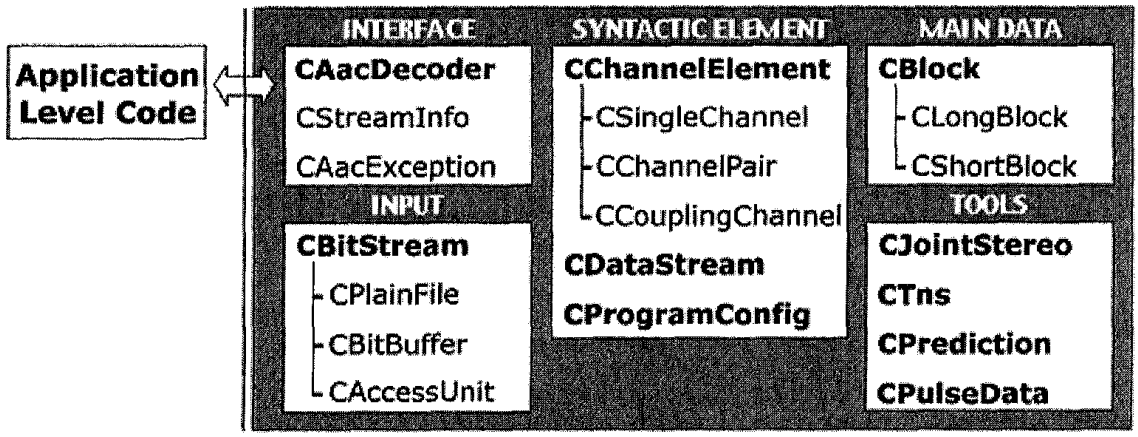

Figure 3: AAC Decoder Class Hierarchy

4.4 Input Classes

The input classes are responsible for parsing the AAC bit stream. CBitstream forms the abstract class for the bitstream parsing, CPlainFile does the ADIF format parsing and CBitBuffer does the ADTS format parsing. The input classes work together with the interface classes to parse the bitstream. The resulting parsed data is stored within the interface class.

4.5 Syntactic Element Classes

The syntactic element classes provide specific classes for the different bitstream elements. CChannelElement is an abstract class that defines the interface for the syntactic element classes. CSingleChannel processes the SCE's (single channel element), CChannelPair processes the CPE's (channel pair element), CDataStream processes the optional DSE's (data stream element), and CProgramConfig processes the PCE's (program configuration element).

Each syntactic element class contains storage space for the decoded data of the respective element as well as objects to perform the decoding functions.

4.6 Main Data Classes

The main data classes hold the spectral coefficients of the current frame. CBlock forms the interface, CLongBlock contains the spectral coefficients for long blocks and CShortBlock holds the spectral coefficients for short blocks.

4.7 Tools Classes

The tool classes contain the arithmetic elements of the decoder. CJointStereo represents the joint stereo coding tools, CTns contains the TNS tool, CPrediction contains the prediction tool, as well as the static state information for each spectral component, and CPulseData performs noiseless coding.

5. CONCLUSION

The MPEG-2 AAC decoder design was described and an overview of the coding tools was presented. The decoding process for the Low Complexity profile was discussed with an emphasis on complexity and memory usage for each decoding tool.

An overview of the software architecture for an MPEG-2 AAC reference C++ decoder was also provided.

7. REFERENCES

[1] ISO/IEC 13818-7:1997(E), "Information technology – Generic coding of moving pictures and associated audio information, Part 7: Advanced Audio Coding."

[2] M. Bosi, K. Brandenburg, S. Quackenbush, L. Fielder, K. Akagiri, H. Fuchs, M. Dietz, J. Herre, G. Davidson, Y. Oikawa, "ISO/IEC MPEG-2 Advanced Audio Coding", Journal of the Audio Engineering Society, Vol. 45, no. 10, pp. 789-814, October 1997.

[3] G. Soulodre, T. Grusec, M. Lavoie, and L. Thibault "Subjective Evaluation of State-of-the-Art Two-Channel Audio Codecs", Journal of the Audio Engineering Society, Vol. 46, no. 3, pp.164-177, March 1998.

[4] J. Herre, J. D. Johnston, "Enhancing the Performance of Perceptual Audio Coders by Using Temporal Noise Shaping (TNS)", presented at the 101st Convention of the Audio Engineering Society, preprint 4384.

[5] S. Quackenbush, Y. Toguri, "Revised Report on Complexity of MPEG-2 AAC Tools", ISO/IEC JTC1/SC29/WG11 N2005, February 1998.