en

08-6329-API-ZCH66
JANUARY 10, 2008
3.4

Application Programmers Interface for aacPlus Enhanced Decoder

ABSTRACT:

Application Programmers Interface for aacPlus Enhanced Decoder

KEYWORDS:

Multimedia codecs, AAC

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	29-Mar-2005	Ashok Kumar	Initial Draft
1.0	30-Mar-2005	Ashok Kumar	Added review comments
1.1	29-Jun-2005	Webber Wang	Add SBR decoder config and table init for relocation; Change some names for aacPlus project.
1.2	07-Jul-2005	Ashok Kumar	Reviewed and updated
2.0	02-Sep-2005	Ashok Kumar	Added review comments and updated performance number
2.1	20-Jan-2006	Kusuma	Modified structure names exposed in the API
3.0	06-Feb-2006	Lauren Post	Using new format
3.1	13-Oct-2006	Shyam Krishnan M	Raw bit stream support.
3.2	19-Oct-2007	Fan Zhang	Restructured API between library and application so that the application uses one unified aacplus_dec_interface.h
3.3	12-Nov-2007	Bing Song	Support 3 channels AAC LC bitstream
3.4	10-Jan-2008	Bing Song	Support up to 5.1 channels and interleave output samples
4.0	28-Apr-09	Lyon Wang	Support invoke AAC LC lib as part of AAC plus
4.1	22-May-09	Lyon Wang	Change SBR decode frame interface

Table of Contents

- 1. Introduction 4

1.1 Purpose 4
1.2 Scope 4
1.3 Audience Description 4
1.4 References 4
1.4.1 Standards 4
1.4.2 General References 5
1.4.3 Freescale Multimedia References 5
1.5 Definitions, Acronyms, and Abbreviations 5
1.6 Document Location 6

- 2 API Description7

Step 1 (Optional): Get Version Information 7
Step 2: Allocate memory for Decoder parameter structure 8
Step 3: Get the decoder memory requirements 10
Step 4: Allocate Data Memory for the decoder 11
Step 5: Get the header information 12
Section A: Parsing ADIF header 12
Section B: Initialization for raw bit stream 15
Step 6: Memory allocation for input buffer 16
Step 7: Initialization routine 16
Step 8: Memory allocation for output buffer 17
Step 9: Call the frame decode routine 17
Step 10: Free memory 20
2.1.1 app_swap_buffers_aacp_dec function usage 20

- Appendix A Debug Logs **Error! Bookmark not defined.**

1. Introduction

1.1 Purpose

This document gives the details of the application programmer's interface of aacPlus decoder. The aacPlus decoder takes the parsed raw aacPlus bit stream as the input and generates audio PCM samples. The calling application needs to parse and provide header information required for decoding of frame.

The decoder is designed as a set of library routines that read the bit-stream from the input buffers and write the decoded output to the output buffers. The decoder implementation currently supports decoding of MPEG-4 AAC (Advanced Audio Coding) LC (low complexity) (level 1 and 2 AAC profile) and aacPlus (level 2 and 3 of High Efficiency (HE) AAC profile) bitstream. The decoder modules are OS independent and do not assume any underlying drivers.

1.2 Scope

This document describes only the functional interface of the aacPlus decoder. It does not describe the internal design of the decoder. Specifically, it describes only those functions by which a software module can use the decoder.

1.3 Audience Description

The reader is expected to have basic understanding of Audio Signal processing and aacPlus decoding. The intended audience for this document is the development community who wish to use the aacPlus decoder in their systems.

1.4 References

1.4.1 Standards

- ISO/IEC 13818-7:1997 Information technology -- Generic coding of moving pictures and associated audio information -- Part 7 (popularly known as *MPEG-2 AAC*)
- ISO/IEC 13818-4:1997 Information technology -- Generic coding of moving pictures and associated audio information -- Part 4 (compliance testing)
- ISO/IEC 14496-3:2001(E) Information technology -- Coding of audio-visual objects -- Part 3: Audio.
- ISO/IEC 14496-4:2000, Information technology -- Coding of audio-visual objects -- Part 4: Conformance testing.
- ISO/IEC 14496-3:2001 Amendment 1 -- WD Text for Backward Compatible Bandwidth Extension for General Audio Coding.
- Coding Technology aacPlus Decoder Certification Document, Version 2.3

1.4.2 General References

- Ted Painter and Andreas Spanias, “Perceptual Coding of Digital Audio”, Proc. IEEE, vol-88, no.4, April 2000
- H.S.Malvar, “Lapped transforms for efficient subband/transform coding”, IEEE trans. ASSP, June 1990.
- Seymour Shlien, “The Modulated Lapped Transform, Its Time-Varying Forms and Its Applications to Audio Coding Standards.”
- “A Tutorial on MPEG/Audio compression” by Davis Pan
- Martin Wolters, Kristofer Kjorling2, Daniel Homm and Heiko Purnhagen, “A closer look into MPEG-4 High Efficiency AAC”, AES 115th Convention, 2003.

1.4.3Freescale Multimedia References

- AAC Plus Enhanced Application Programming Interface - aacplus_dec_api.doc
- AAC Plus Enhanced Requirements Book - aacplus_dec_reqb.doc
- AAC Plus Enhanced Test Plan - aacplus_dec_test_plan.doc
- AAC Plus Enhanced Release notes - aacplus_dec_release_notes.doc
- AAC Plus Enhanced Test Results – aacplus_dec_test_results.doc
- AAC Plus Enhanced Performance Results – aacplus_dec_perf_results.doc
- AAC Decoder API doc Appendix C on Header data extraction – aac_dec_api.doc
- AAC Plus Enhanced Interface Header – aacplus_dec_interface.h
- AAC Plus Enhanced Application Code - aacplus_main.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
AAC	Advanced Audio Coding
aacPlus	AAC low complexity plus SBR decoder
aacPlus V2	AAC low complexity plus SBR plus parametric stereo decoder
ADIF	Audio_Data_Interchange_Format
ADTS	Audio_Data_Transport_Stream
API	Application Programming Interface
ARM	Advanced RISC Machine
FSL	Freescale
HE-AAC	High Efficiency AAC
IEC	International Electro-technical Commission
ISO	International Organization for Standardization
LC	Low Complexity
MDCT	Modified Discrete Cosine Transform
MPEG	Moving Pictures Expert Group

PCM	Pulse Code Modulation
PNS	Perceptual Noise Substitution
PS	Parametric Stereo
SBR	Spectral Band Replication
RVDS	ARM RealView Development Suite
UNIX	Linux PC x/86 C-reference binaries
TBD	To be decided

1.6 Document Location

docs/aacplus_dec

2 API Description

This section describes the steps followed by the application to call the aacPlus decoder. During each step the data structures and functions used will be explained. Pseudo code is given at the end of each step. The member variables inside the structures are prefixed as aacpd_ or app_ to indicate if that member variable needs to be initialized by the decoder or application.

Step 1 (Optional): Get Version Information

This function returns the codec library version information details. It can be called at any time and it provides the library's information: Component name, supported ARM family, Version Number, supported OS, build date and time and so on.

C prototype:
```
const char * AACPDCodecVersionInfo ();
```

Arguments:
- None.

Return value:
- const char * - The pointer to the constant char string of the version information string.

Description of the version information

It is defined as :

```
#define SEPARATOR " "
#define BASELINE_SHORT_NAME "AACPD_ARM_03.00.00"

#ifdef __WINCE
#define OS_NAME "_WINCE"
#else
#define OS_NAME ""
#endif

#ifdef DEMO_VERSION
#define CODEC_RELEASE_TYPE "_DEMO"
#else
#define CODEC_RELEASE_TYPE ""
#endif

/* user define suffix */
#define VERSION_STR_SUFFIX ""

#define CODEC_VERSION_STR \
    (BASELINE_SHORT_NAME OS_NAME CODEC_RELEASE_TYPE \
     SEPARATOR VERSION_STR_SUFFIX \
     SEPARATOR "build on" \
     SEPARATOR __DATE__ SEPARATOR __TIME__)
```

For instance:
"AACPD_ARM_03.00.00 build on May 14 2008 16:21:50"
is returned by calling AACPDCodecVersionInfo (), meaning AAC-Plus decoder for ARM11, version 2.01.00.

Example pseudo code for the memory information request

```
{
  // Output the AAC Decoder Version Info
  printf("%s \n", aacd_decode_versionInfo());
}
```

Step 2: Allocate memory for Decoder parameter structure

The application allocates memory for the structure mentioned below. This structure contains the decoder parameters and memory information structures.

```
/* Decoder  parameter  structure */
typedef struct AACPD_Decoder_Config_struct
{
      AACPD_Mem_Alloc_Info    aacpd_mem_info;
      AACPD_Void              *aacpd_decode_info_struct_ptr;
      AACPD_ UINT8            *app_initialized_data_start;
      AACPD_INT8
            (*app_swap_buf)(  AACPD_UINT8 **new_buf_ptr,
                              AACPD_INT32 *new_buf_len,
                        struct AACPD_Decoder_Config_struct *dec_config);
                              ACPD_UINT8  num_pcm_bits;
      AACPD_UINT8      num_pcm_bits;  /* Specifies number of Bits in the
PCM.(16 or 24)*/
      AACPD_Block_Params *params;
      SBRD_Decoder_Config sbrd_dec_config;
      void *pContext;
} AACPD_Decoder_Config;
```

Description of the decoder parameter structure *AACPD_Decoder_Config*

aacpd_mem_info

This is a memory information structure. The application needs to call the function aacpd_query_dec_mem to get the memory requirements from the decoder. The decoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

aacpd_decode_info_struct_ptr

This is a void pointer. This will be initialized by the decoder during the initialization routine. This will then be a pointer to a structure which contains the pointers to tables, buffers and symbols used by the decoder.

app_initialized_data_start

The application has to assign this pointer with the start of the array of table pointers. The array of table pointers should have been initialized with the starting addresses of the tables used for aacPlus decoder. The starting addresses of the tables used is given in the header file aacpd_tables.h

app_swap_buf

Function pointer to swap buffers. The application has to initialize this pointer.

num_pcm_bits

The application has to indicate the decoder about required output precision. The decoder can output the PCM samples either as 16 bit samples or as 24 bit samples. Currently 16bit PCM output samples are used.

params

This is a pointer to a structure containing the header information. The header is parsed by the application and this structure is filled with the appropriate header information. This structure needs to be updated whenever header is parsed. This will be discussed in **step 4**

sbrd_dec_config

This is a SBR Decoder config structure. The application will use the elements in this structure to decode the sbr data.

pContext

Used to indicate instrance.

```
/* SBR Decoder parameter structure */
typedef struct
{
  AACPD_INT32   sbrd_down_sample;
           /*flag to indicate SBR down sample mode*/
  AACPD_INT32   sbrd_stereo_downmix;
           /* flag related to stereo down mix */
}SBRD_Decoder_Config;
```

Description of SBR Decoder parameter *SBRD_Decoder_Config*

sbrd_down_sample

This is flag used to enable/disable downsampled mode of SBR. The application needs to initialize this to zero as this is not supported.

sbrd_stereo_downmix

This flag is used to enable/disable stereo to mono downmix. The application needs to initialize this to zero as this is not supported.

Example pseudo code for this step:

```
/* Allocate memory for the decoder parameter */
   dec_config = (AACPD_Decoder_Config *)
                        alloc (sizeof(AACPD_Decoder_Config);

/* Initialize aptable with the start address of all the tables used
   for aacPlus decoder. The start addresses of all the tables are
   available in the header file aacpd_tables.h  */

/* Request the output PCM samples to be in the 16bit format/24bit format
   dec_config->num_pcm_bits = AACPD_16_BIT_OUTPUT;

/* Assign swap-buffer function to the swap buffer function pointer,
   function app_swap_buffers_aacp_dec need to be implemented by
   application */
   dec_config->app_swap_buf = app_swap_buffers_aacp_dec;

/* Fill up the Block Params field in decoder config, to be used by adif
   or adts header processing */
   dec_config->params = &BlockParams;
```

Step 3: Get the decoder memory requirements

The aacPlus decoder does not perform any dynamic memory allocation. The application calls the function *aacpd_query_dec_mem* to get the decoder memory requirements. This function must be called before any other decoder functions are invoked.

C prototype

```
AACPD_RET_TYPE aacpd_query_dec_mem (AACPD_Decoder_config * dec_config);
```

Arguments

- dec_config - pointer to decoder configuration structure.

Return value

- AACPD_ERROR_NO_ERROR - Memory query successful.
- Other codes - Error (For other error codes refer to appendix).

This function populates the memory information structure, which is described below:

Memory information structure array

```
typedef struct
{
      /* Number of valid memory requests */
      AACPD_INT32                  aacpd_num_reqs;
      AACPD_Mem_Alloc_Info_Sub     mem_info_sub[AACPD_MAX_NUM_MEM_REQS];
}AACPD_Mem_Alloc_Info;
```

Description of the structure *AACPD_Mem_Alloc_Info*

aacpd_num_reqs

The number of memory chunks requested by the decoder.

mem_info_sub

This structure contains each chunk's memory configuration parameters.

Memory configuration Structure

```
typedef struct
{
      AACPD_INT32       aacpd_size;       /* Size in bytes */
      AACPD_INT32       aacpd_type;       /* Memory type Fast or Slow */
      AACPD_MEM_DESC    aacpd_mem_desc;
            /* Flag to indicate Static/Scratch
                        memory */
      AACPD_MEMALIGN    aacpd_mem_align;  /* required alignment */
      AACPD_MEM_PRIORITY   aacpd_priority;
            /* In case of fast Memory type, specify the priority */
      AACPD_Void  *app_base_ptr;    /* Pointer to the base memory, which
                                       will be allocated and filled by
                                       the application */
} AACPD_Mem_Alloc_Info_sub
```

Description of the structure *AACPD_Mem_Alloc_Info_sub*

aacpd_size
This indicates size of chunk in bytes.

aacpd_type
The type of the memory indicates if the requested chunk of memory needs to be allocated in external or internal memory. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In targets where there is no internal memory, the application can allocate memory in external memory.
(Note: If the decoder requests for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the decoder will still decode, but the performance (MHz) will suffer.)

aacpd_mem_des
The memory description field indicates whether requested chunk of memory is static or scratch.

aacpd_mem_align
This indicates requirement alignment in bytes for requested memory. It can be 1, 2 and 4 bytes.

aacpd_priority
In case, if the decoder requests for multiple memory chunks in the fast memory, the priority indicates the order in which the application has to prioritize placing the requested chunks in fast memory

app_base_ptr
This will be initialized by the application. The application will allocate the memory for each chunk depending on the requested size and the type, and then assign the base address of this chunk of memory to *app_base_ptr*. The application should allocate the memory that is aligned to a 4 byte boundary in any case.

Example pseudo code for the memory information request

```
/* Query for memory */
retval = aacpd_query_dec_mem (&dec_config);

if (retval != AACPD_ERROR_NO_ERROR)
    {
        printf("Failed to get the memory configuration for the
decoder\n");
      aacd_free(dec_config);
        return 2;
    }
```

Step 4: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by the aacPlus decoder and fills up the base memory pointer *'app_base_ptr' of 'AACPD_Mem_Alloc_Info_sub'* structure for each chunk of memory requested by the decoder.

Example pseudo code for the memory allocation and filling the base memory pointer by the application

```
AACPD_Mem_Alloc_Info_sub *mem;

/* Number of memory chunks requested by the decoder */
nr = dec_config->aacpd_mem_info.aacpd_num_reqs;
```

```
for(i = 0; i < nr; i++)
{
	mem = &(dec_config-> aacpd_mem_info.mem_info_sub[i]);
	if (mem->aacpd_type == FAST_MEMORY)
	{
		/* this function allocates memory in internal memory as per
			requested alignment (mem->aacpd_mem_align) */
		mem->app_base_ptr = alloc_fast(mem->aacpd_size);
	}
	else
	{
		/* this function allocates memory in external memory as per
			requested alignment (mem->aacpd_mem_align) */
		mem->app_base_ptr = alloc_slow(mem->aacpd_size);
	}
}
```

Step 5: Get the header information

Section A: Parsing ADIF header

The application does the parsing of the header. The header structure which is required by the decoder to process the raw data is updated by the application. Please refer for more details about the structures given below. (The header parse function will also defined by application).

Header information structure:

```
typedef struct
{
	AACPD_INT32		num_pce;
	AACPD_ProgConfig *pce;
	AACPD_INT32 ChannelConfig;
	AACPD_INT32 SamplingFreqIndex;
	AACPD_INT32 BitstreamType;
	AACPD_INT32 BitRate;
	AACPD_INT32 BufferFullness; /*No. of bits in decoder buffer after
					decoding first raw_data_block */
	AACPD_INT32 ProtectionAbsent;
	AACPD_INT32 CrcCheck;
#ifdef OLD_FORMAT_ADTS_HEADER
	AACPD_INT32 Flush_LEN_EMPHASIS_Bits;
#endif
} AACPD_Block_Params;
```

Description of the structure *AACPD_Block_Params*

num_pce

Number of program config elements in the header. If there are no program config elements in the header, num_pce should be set to zero.

pce

This is pointer to the program configuration structure. (Details of Program config structure is given below).

ChannelConfig

Channel Configuration parameter can take values in the range 0-7. Each value specified indicates preset channel-to-speaker mapping defined in the standard. If num_pce is greater than zero, then ChannelConfig will be ignored as pce will be used.

SamplingFreqIndex

This is an index into an array of valid sampling-frequency values. This will be ignored if num_pce is greater than zero.

BitstreamType

0 = Constant bit rate (CBR)
1 = Variable bit rate (VBR)

BitRate

If BitstreamType = CBR, then it indicates actual bit rate
If BitstreamType = VBR, then it indicates peak bit rate if not equal to zero
If BitstreamType = VBR and BitRate = 0, then peak bit rate is unknown.

BufferFullness

This indicates number of bits remaining in the encoder buffer after encoding the first raw_data_block in the frame. This will be ignored if num_pce is greater than zero.

ProtectionAbsent

If ProtectionAbsent =1, then CrcCheck value will not be present in the stream
If ProtectionAbsent =0, then CrcCheck value will be present in the stream and CrcCheck will be done by the decoder.

CrcCheck

32bit field used to perform CRC check by the decoder.

Flush_LEN_EMPHASIS_Bits

This is used only for the MPEG4 streams with old format ADTS headers
0: If byte alignment is already existing at the end of the header
1: If there is no byte alignment at the end of the header. If this is 1, decoder flushes 2 bits before decoding every frame.

PCE Structure

```
typedef struct
{
      AACPD_INT32       profile;
      AACPD_INT32       sampling_rate_idx;
      AACPD_EleList     front;
      AACPD_EleList     side;
      AACPD_EleList     back;
      AACPD_EleList     lfe;
      AACPD_EleList     data;
      AACPD_EleList     coupling;
      AACPD_MIXdown     mono_mix;
      AACPD_MIXdown     stereo_mix;
      AACPD_MIXdown     matrix_mix;
      AACPD_INT8        comments[1];
      AACPD_INT32       buffer_fullness;
      AACPD_INT32       tag;
}AACPD_ProgConfig;
```

Description of the structure *AACPD_ProgConfig*

profile

This should always be set to 1 (MPEG4 AAC Low Complexity profile)

sampling_rate_idx

This is an index into an array of valid sampling-frequency values

front
This is structure containing information about front channel elements.
side
This is structure containing information about side channel elements.
back
This is structure containing information about back channel elements.
lfe
This is structure containing information about low frequency enhancement channel elements.
data
This is structure containing information about the associated data elements for this program
coupling
This is structure containing information about the coupling channel elements.
mono_mix
This is structure containing information about the mono mixdown element
Stereo_mix
This is structure containing information about the stereo mixdown element
Matrix_mix
This is structure containing information about the matrix mixdown element.
Comments
General information
Buffer_fullness
This indicates number of bits remaining in the encoder buffer after encoding the first raw_data_block in the frame.
tag
The ID of the PCE

EleList structure:

```
typedef struct
{
      AACPD_INT32             num_ele;
      AACPD_INT32             ele_is_cpe[16];
      AACPD_INT32             ele_tag[16];
} AACPD_EleList;
```

Description of the structure *AACPD_EleList*
num_ele
Number of elements in the channel
ele_is_cpe
This indicates whether the element is channel_pair or not
ele_tag
ID of channel element

Mixdown structure:

```
typedef struct
{
      AACPD_INT32           present;
      AACPD_INT32           ele_tag;
      AACPD_INT32           pseudo_enab;
} AACPD_MIXdown;
```


Description of the structure *AACPD_MIXdown*

present

This indicates if the mix_down element is present or not

ele_tag

This indicates the number of the mix_down element

pseudo_enab

This is defined only for matrix mix-down and indicates if pseudo-surround is enabled or not.

Section B: Initialization for raw bit stream

Whenever the input is raw bit stream, we initialize the adts header structure to default values except for sampling frequency and channel configuration. This initialization is done only once.

ADTS structure:

```
typedef struct
{
    int             syncword;
    int             id;
    int             layer;
    int             protection_abs;
    int             profile;
    int             sampling_freq_idx;
    int             private_bit;
    int             channel_config;
    int             original_copy;
    int             home;
    int             copyright_id_bit;
    int             copyright_id_start;
    int             frame_length;
    int             adts_buffer_fullness;
    int             num_of_rdb;
    int             crc_check;
}               ADTS_Header;
```

Initialization of the structure shown above for raw bit stream decoding

```
Id=0;
Layer=0;
protection_abs=1;
profile=1;
sampling_freq_idx
        Index derived from the sampling frequency. Sampling frequency
needs to be given as an input parameter.
private_bit=0;
channel_config;
          Represents mono(1) or stereo(2) bit stream. This is also an
input parameter.
original_copy=0;
home=0;
```

```
copyright_id_bit=0;
copyright_id_start=0;
frame_length=0;
adts_buffer_fullness=0;
num_of_rdb=0;
```

Step 6: Memory allocation for input buffer

The application has to allocate the memory needed for the input buffer. It is desirable to have the input buffer allocated in FAST_MEMORY, as this may improve the performance (MHz) of the decoder. There is no restriction on the size of the input buffer to be given to the decoder. The recommended minimum size would be 2Kbytes. The decoder, whenever it needs the aacPlus bit-stream, shall call the function *app_swap_buffers_aacp_dec* internally from the function *aacp_decode_frame.*

app_swap_buffers_aacp_dec should be implemented by the application. The application might have different techniques to implement this function. Sample code is given in section 2.1.1

Example pseudo code for allocating the input buffer

```
/* Allocate memory for input buffer */
      input_buf = alloc_fast(AACPD_INPUT_BUFFER_SIZE);
```

Step 7: Initialization routine

All initializations required for the decoder are done in *aacpd_decode_init*. This function must be called before the main decoder function is called.

C prototype:

```
AACD_RET_TYPE aacd_decode_init (AACPD_Decoder_Config *);
AACD_RET_TYPE SBRD_decode_frame(
        AACD_Decoder_Config *dec_config,
        AACD_Decoder_info *dec_info,
        AACD_INT32 *out_buf);
```

Arguments:

- Pointer to decoder configuration structure

Return value:

- AACD_ERROR_NO_ERROR - Initialization successful.
- Other codes - Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the AAC decoder. */
retval = aacd_decode_init (dec_config);
if (retval != AACD_ERROR_NO_ERROR)
      return 1;
retval = SBRD_decoder_init(dec_config);
if (retval != AACD_ERROR_NO_ERROR)
      return 1;
```

Step 8: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the decoded stereo PCM samples for a maximum of one frame size. The pointer to this output buffer needs to be passed to the aacpd_decode_frame function. The application can allocate memory for output buffer in external memory using alloc_slow. Allocating memory in internal memory using alloc_fast will improve the performance (MHz) of the decoder marginally. It would be desirable to allocate the buffer in the slow memory.

Example pseudo code for allocating memory for output buffer

```
/* allocate memory for output buffer */
      outbuf = alloc_slow(num_of_channels * AACP_FRAME_SIZE);
```

In the example code, the output buffer has been declared as a two dimensional array

```
      AACPD_INT32 outbuf[num_of_channels * AACP_FRAME_SIZE];
```

Note:
The current implementation of aacPlus decoder supports up to 5.1 channels AAC LC bitstream and stereo AAC PLUS bitstream. The output samples have been interleaved.

Step 9: Call the frame decode routine

The main aacPlus decoder function is *aacpd_decode_frame*. This function decodes the aacPlus raw bit stream of input buffer to generate one frame of decoder output per channel in every call. If the input buffer does not contain enough data for one frame, then only the call back function app_swap_buffers_aacp_dec will be called to get more input.

The output buffer is first filled with left channel samples and then with the right channel, samples. For mono streams, the decoder fills only the left channel samples and leaves the right channel samples unfilled.

The decoder fills up the following AACD_Decoder_Info structure:

```
typedef struct
{
      AACPD_UINT32      aacpd_sampling_frequency;
            /* Sampling frequency of the current frame in KHz */
      AACPD_UINT32      aacpd_num_channels;
            /* Number of channels decoded in current frame */
      AACPD_UINT32      aacpd_frame_size;
            /* Number of stereo samples being output for this frame */
      AACPD_UINT32      aacpd_len;
      AACPD_UINT32      aacpd_bit_rate;         /* bit rate */
      AACPD_UINT32      BitsInBlock;
            /* Decoder outputs the number of bits in the raw data block
            decoded. This can be used by the application to keep track of
            the bitstream */
```

```
    AACPD_INT8      ch_is_present[Chans]; /* flag to indicate presence of
each channel */
    AACPD_UINT32    AACD_bno;        /* frame number, updated after each
call to aacpd_decode_frame*/

} AACD_Decoder_Info;
```

If the bit stream has errors, the decoder will return the corresponding error (mentioned in the appendix).

C prototype:

*AACD_RET_TYPE aacpd_decode_frame (AACD_Decoder_Config *dec_config,*
*AACD_Decoder_Info *dec_info,*
*AACD_INT32 *output_buffer,*
*AACD_INT8 * inbuf, AACPD_INT32 buf_len);*

Arguments:

- dec_config Decoder parameter structure pointer
- dec_info Decoder output parameter pointer
- output_buffer Pointer to the output buffer to hold the decoded samples
- inbuf Pointer to the input buffer containing the raw data block
- buf_len Length of the input buffer

Return value:

- AACD_ERROR_NO_ERROR indicates decoding was successful.
- Others indicates error

*AACD_RET_TYPE SBRD_decode_frame(AACD_Decoder_Config *dec_config,*
*AACD_Decoder_info *dec_info,*
*AACD_INT32 *out_buf*
*SBR_FRAME_TYPE * sbr_frame_type);*

Arguments:

- dec_config Decoder parameter structure pointer
- dec_info Decoder output parameter pointer
- out_buf Pointer to the output buffer to hold the decoded samples
- sbr_frame_type Indicate if current frame contains SBR information

Return value:

- AACD_ERROR_NO_ERROR indicates decoding was successful.
- Others indicates error

```
typedef enum
{
   NO_SBR_FRAME, /* indicate no sbr information */
   SBR_FRAME     /* indicate sbr information contained */
}SBR_FRAME_TYPE;
```

When the decoder encounters the end of bit-stream, the application comes out of the loop. In case of error while decoding the current frame, the application can just ignore the frame without processing the output samples by continuing the loop.
In case of mono bit-streams, as mentioned earlier, the decoder fills only the left channel. It is the responsibility of the application to use it accordingly. One example is the case in which the

application can copy the left channel samples into right channel. This is illustrated in the example code below.

Example pseudo code for calling the main decode routine of the decoder

```
AACPD_Decoder_Info      dec_info;
AACPD_INT8              inbuf;
AACPD_UINT32            len;

while (TRUE)
{
      /* Decode one frame */
      /* The decoded parameters for this frame are available in the
         structure AACPD_Decoder_info */

      len = AACPD_INPUT_BUFFER_SIZE;

      /* Get the input raw data block */
      read (inbuf,len)

      Retval = aacd_decode_frame (dec_config, &dec_info, outbuf, inbuf,
                        len);
      if (retval == AACD_ERROR_EOF)
      {
            /* Reached the end of bit-stream */
            break;
      }
      Retval = SBRD_decode_frame (dec_config, &dec_info, outbuf,
&sbr_frame_type));

      if (retval != AACD_ERROR_NO_ERROR)
      {
            /* Invalid frame encountered, do not output */
            continue;
      }

      /* If mono, copy the left channel to the right channel. */
      if (dec_info.aacpd_num_channels == 1)
      {
            int i;
            for (i = 0; i < dec_info.aacpd_frame_size; i++)
                outbuf [AACP_FRAME_SIZE+i] = outbuf [i];
      }

      /* The output frame is ready for use. Decoding of the next frame
       * should start only if the previous output frame has been fully
       * used by the application.
       */
       /* audio_output_frame () is an application function that outputs
the
          decoded samples to the output port/device. This function is not
          given in this document.
      */
       audio_output_frame(outbuf, 2*AACP_FRAME_SIZE);
```



```
}
```

Step 10: Free memory

The application releases the memory that it allocated to aacPlus decoder if it no longer needs the decoder instance.

```
free (outbuf);
free (inbuf);
for (i=0; i<nr; i++)
{
      free (dec_config-> aacpd_mem_info.mem_info_sub[i].app_base_ptr);
}
free (dec_config);
```

2.1.1 app_swap_buffers_aacp_dec function usage

app_swap_buffers_aacp_dec is called by the decoder to get a new input buffer for decoding. This function is called by the aacPlus decoder from within the *aacpd_decode_frame* function when it runs out of current bit stream input buffer. The decoder uses this function to return the used buffer and get a new bit stream buffer.

This function should be implemented by the application. The parameter new_buf_ptr is a double pointer. This will hold the recently used buffer by the decoder when this function is called. The application can decide to free this or do any sort of arithmetic to get any new address. The application needs to put the new input buffer pointer in *new_buf_ptr to be used by the decoder.

The interface for this function is described below:

C prototype:

```
AACPD_INT8 app_swap_buffers_aacp_dec (AACPD_UINT8 ** new_buf_ ptr,
                          ACPD_UINT32 * new_buf_len,
                          AACPD_Decoder_Config *aacpd_decod_config);
```

Arguments:

- *new_buf_ ptr* - Pointer to the new buffer given by the application.
- *new_buf_len* - Pointer to length of the new buffer in bytes
- *aacpd_decod_config* - Decoder configuration structure.

Return value:

- 0 - Buffer allocation successful.
- -1 - End of bit-stream

Example pseudo code

```
AACPD_INT8 app_swap_buffers_aacp_dec (AACPD_INT8 ** new_buf_ptr,
                                      AACPD_UINT32 * new_buf_len,
                                      AACPD_Decoder_Config
                    *aacpd_decod_config)
{
```

```
        Request for an input buffer from the application
        Wait for the input buffer corresponding to the decoder
        instance of instance_id.

        if the new buffer arrives
        {
              Return the used buffer to the application.

              Set the new_buf_ptr to point to the new buffer and
              set *new_buf_len to the length of the new buffer.

              Return 0 to the calling function to indicate that new buffer
              has been received.
        }
        else if the application indicates end of bit stream
        {
              Set new_buf_ptr to NULL and *new_buf_len to 0.

              Return -1 to the calling function to indicate the end of
              input bit stream.
        }
}
```