

AAC HEv2 Encoder on C64x+

User's Guide



Literature Number: SPRUET3A
December 2008

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright 2008, Texas Instruments Incorporated

Read This First

About This Manual

This document describes how to install and work with Texas Instruments' (TI) AAC Encoder implementation on the C64x+ platform. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI's codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

Intended Audience

This document is intended for system engineers who want to integrate TI's codecs with other software to build a multimedia system based on the C64x+ platform.

This document assumes that you are fluent in the C language, have a good working knowledge of Digital Signal Processing (DSP), digital signal processors, and DSP applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) and eXpressDSP Digital Media (XDM) standard will be helpful.

How to Use This Manual

This document includes the following chapters:

- ❑ **Chapter 1 - Introduction**, provides a brief introduction to the XDAIS and XDM standards. It also provides an overview of the codec and lists its supported features.
- ❑ **Chapter 2 - Installation Overview**, describes how to install, build, and run the codec.
- ❑ **Chapter 3 - Sample Usage**, describes the sample usage of the codec.
- ❑ **Chapter 4 - API Reference**, describes the data structures and interface functions used in the codec.
- ❑ **Appendix A - AAC Encoder Bit-rate and Sampling Frequency Combination**, contains the AAC LC Encoder bit-rate and sampling frequency combination.

- ❑ **Appendix B - AAC Encoder Bit-rate and Sampling Frequency Combination**, contains the AAC HC Encoder bit-rate and sampling frequency combination.
- ❑ **Revision History** - highlights the changes made to the SPRUET3 codec specific user guide to make it SPRUET3A.

Related Documentation From Texas Instruments

The following documents describe TI's DSP algorithm standards such as, XDAIS and XDM. To obtain a copy of any of these TI documents, visit the Texas Instruments website at www.ti.com.

- ❑ *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) defines a set of requirements for DSP algorithms that, if followed, allow system integrators to quickly assemble production-quality systems from one or more such algorithms.
- ❑ *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360) describes all the APIs that are defined by the TMS320 DSP Algorithm Interface Standard (also known as XDAIS) specification.
- ❑ *Technical Overview of eXpressDSP - Compliant Algorithms for DSP Software Producers* (literature number SPRA579) describes how to make algorithms compliant with the TMS320 DSP Algorithm Standard which is part of TI's eXpressDSP technology initiative.
- ❑ *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577) describes how an eXpressDSP-compliant algorithm may be used effectively in a static system with limited memory.
- ❑ *DMA Guide for eXpressDSP-Compliant Algorithm Producers and Consumers* (literature number SPRA445) describes the DMA architecture specified by the TMS320 DSP Algorithm Standard (XDAIS). It also describes two sets of APIs used for accessing DMA resources: the IDMA2 abstract interface and the ACPY2 library.
- ❑ *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8)

The following documents describe TMS320 devices and related support tools:

- ❑ *Design and Implementation of an eXpressDSP-Compliant DMA Manager for C6X1X* (literature number SPRA789) describes a C6x1x-optimized (C6211, C6711) ACPY2 library implementation and DMA Resource Manager.
- ❑ *TMS320C64x+ Megamodule* (literature number SPRAA68) describes the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.
- ❑ *TMS320C64x+ DSP Megamodule Reference Guide* (literature number SPRU871) describes the C64x+ megamodule peripherals.

-
- ❑ *TMS320C64x to TMS320C64x+ CPU Migration Guide* (literature number SPRAA84) describes migration from the Texas Instruments TMS320C64x™ digital signal processor (DSP) to the TMS320C64x+™ DSP.
 - ❑ *TMS320C6000 Optimizing Compiler v 6.0 Beta User's Guide* (literature number SPRU187N) explains how to use compiler tools such as compiler, assembly optimizer, standalone simulator, library-build utility, and C++ name demangler.
 - ❑ *TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide* (literature number SPRU732) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.
 - ❑ *TMS320DM6446 Digital Media System-on-Chip* (literature number SPRS283)
 - ❑ *TMS320DM6446 Digital Media System-on-Chip Errata (Silicon Revision 1.0)* (literature number SPRZ241) describes the known exceptions to the functional specifications for the TMS320DM6446 Digital Media System-on-Chip (DMSoC).
 - ❑ *TMS320DM6443 Digital Media System-on-Chip* (literature number SPRS282)
 - ❑ *TMS320DM6443 Digital Media System-on-Chip Errata (Silicon Revision 1.0)* (literature number SPRZ240) describes the known exceptions to the functional specifications for the TMS320DM6443 Digital Media System-on-Chip (DMSoC).
 - ❑ *TMS320DM644x DMSoC DSP Subsystem Reference Guide* (literature number SPRUE15) describes the digital signal processor (DSP) subsystem in the TMS320DM644x Digital Media System-on-Chip (DMSoC).
 - ❑ *TMS320DM644x DMSoC ARM Subsystem Reference Guide* (literature number SPRUE14) describes the ARM subsystem in the TMS320DM644x Digital Media System on a Chip (DMSoC).
 - ❑ *DaVinci Technology - Digital Video Innovation Product Bulletin (Rev. A)* (literature number SPRT378A)
 - ❑ *The DaVinci Effect: Achieving Digital Video Without Complexity White Paper* (literature number SPRY079)
 - ❑ *DaVinci Benchmarks Product Bulletin* (literature number SPRT379)
 - ❑ *DaVinci Technology for Digital Video White Paper* (literature number SPRY067)
 - ❑ *The Future of Digital Video White Paper* (literature number SPRY066)

Related Documentation

You can use the following documents to supplement this user guide:

- ❑ *ISO/IEC IS 14496-3 Information Technology -- Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbps -- Part 3: Audio*
- ❑ *ISO/IEC IS 13818-7 Information Technology -- Generic Coding of Moving Pictures and Associated Audio Information -- Part 7: Advanced Audio Coding*

Abbreviations

The following abbreviations are used in this document:

Table 1-1. List of Abbreviations

Abbreviation	Description
API	Application Programming Interface
ADIF	Audio Data Interchange Format
ADTS	Audio Data Transport Stream
CBR	Constant Bit-rate
CRC	Cyclic Redundancy Check
DMA	Direct Memory Access
DMAN3	DMA Manager
EVM	Evaluation Module
HE	High Efficiency
HEv2	High Efficiency with parametric stereo
Kbps	Kilo bits per second
LC	Low Complexity
MPEG	Moving Picture Experts Group
PCM	Pulse code modulation
PNS	Perceptual noise substitution
PS	Parametric Stereo
SBR	Spectral Band Replication

Abbreviation	Description
TNS	Tonal Noise Shaping
VBR	Variable Bit-rate
XDAIS	eXpressDSP Algorithm Interface Standard
XDM	eXpressDSP Digital Media

Text Conventions

The following conventions are used in this document:

- Text inside back-quotes (“”) represents pseudo-code.
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

Product Support

When contacting TI for support on this codec, quote the product name (AAC Encoder on C64x+) and version number. The version number of the codec is included in the Title of the Release Notes that accompanies this codec.

Trademarks

Code Composer Studio, the DAVINCI Logo, DAVINCI, DSP/BIOS, eXpressDSP, TMS320, TMS320C64x, TMS320C6000, TMS320DM644x, and TMS320C64x+ are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

This page is intentionally left blank

Contents

Read This First	iii
About This Manual	iii
Intended Audience	iii
How to Use This Manual	iii
Related Documentation From Texas Instruments.....	iv
Related Documentation.....	vi
Abbreviations	vi
Text Conventions	vii
Product Support	vii
Trademarks	vii
Contents	ix
Figures	xi
Tables	xiii
Introduction	1-1
1.1 Overview of XDAIS and XDM.....	1-2
1.1.1 XDAIS Overview	1-2
1.1.2 XDM Overview	1-2
1.2 Overview of AAC Encoder.....	1-4
1.3 Supported Services and Features.....	1-4
Installation Overview	2-1
2.1 System Requirements	2-2
2.1.1 Hardware.....	2-2
2.1.2 Software	2-2
2.2 Installing the Component.....	2-2
2.3 Before Building the Sample Test Application	2-4
2.3.1 Installing DSP/BIOS.....	2-4
2.4 Building and Running the Sample Test Application	2-4
2.5 Configuration Files	2-5
2.5.1 Generic Configuration File	2-5
2.5.2 Encoder Configuration File.....	2-6
2.6 Standards Conformance and User-Defined Inputs	2-6
2.7 Uninstalling the Component	2-7
2.8 Evaluation Version	2-7
Sample Usage	3-1
3.1 Overview of the Test Application.....	3-2
3.1.1 Parameter Setup	3-3
3.1.2 Algorithm Instance Creation and Initialization.....	3-3
3.1.3 Process Call.....	3-4
3.1.4 Algorithm Instance Deletion	3-5
API Reference	4-1
4.1 Symbolic Constants and Enumerated Data Types.....	4-2
4.2 Data Structures	4-8
4.2.1 Common XDM Data Structures.....	4-8
4.2.2 AAC Encoder Data Structures	4-17

4.3	Interface Functions.....	4-21
4.3.1	Creation APIs.....	4-21
4.3.2	Initialization API.....	4-23
4.3.3	Control API.....	4-24
4.3.4	Data Processing API.....	4-26
4.3.5	Termination API.....	4-28
	AAC LC Encoder Bit-rate and Sampling Frequency Combination.....	A-1
	AAC HE Encoder Bit-rate and Sampling Frequency Combination.....	B-1
	Revision History.....	C-1

Figures

Figure 2-1. Component Directory Structure	2-2
Figure 3-1. Test Application Sample Implementation.....	3-2

This page is intentionally left blank

Tables

Table 1-1. List of Abbreviations.....	vi
Table 2-1. Component Directories.....	2-3
Table 4-1. List of Enumerated Data Types.....	4-2
Table 4-2. AAC Encoder Error Status.....	4-5
Table C-1. Revision History of AAC High Efficiency Encoder on C64x+	C-1

This page is intentionally left blank

Introduction

This chapter provides a brief introduction to XDAIS and XDM. It also provides an overview of TI's implementation of the AAC Encoder on the C64x+ platform and its supported features.

Topic	Page
1.1 Overview of XDAIS and XDM	1-2
1.2 Overview of AAC Encoder	1-4
1.3 Supported Services and Features	1-4

1.1 Overview of XDAIS and XDM

TI's multimedia codec implementations are based on the eXpressDSP Digital Media (XDM) standard. XDM is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

1.1.1 XDAIS Overview

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. In order to facilitate these functionalities, the IALG interface defines the following APIs:

- ❑ `algAlloc()`
- ❑ `algInit()`
- ❑ `algActivate()`
- ❑ `algDeactivate()`
- ❑ `algFree()`

The `algAlloc()` API allows the algorithm to communicate its memory requirements to the client application. The `algInit()` API allows the algorithm to initialize the memory allocated by the client application. The `algFree()` API allows the algorithm to communicate the memory to be freed when an instance is no longer required.

Once an algorithm instance object is created, it can be used to process data in real-time. The `algActivate()` API provides a notification to the algorithm instance that one or more algorithm processing methods is about to be run zero or more times in succession. After the processing methods have been run, the client application calls the `algDeactivate()` API prior to reusing any of the instance's scratch memory.

The IALG interface also defines three more optional APIs `algControl()`, `algNumAlloc()`, and `algMoved()`. For more details on these APIs, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

1.1.2 XDM Overview

In the multimedia application space, you have the choice of integrating any codec into your multimedia system. For example, if you are building a video decoder system, you can use any of the available video decoders (such as MPEG4, H.263, or H.264) in your system. To enable easy integration with the client application, it is important that all codecs with similar functionality use similar APIs. XDM was primarily defined as an extension to XDAIS to ensure uniformity across different classes of codecs

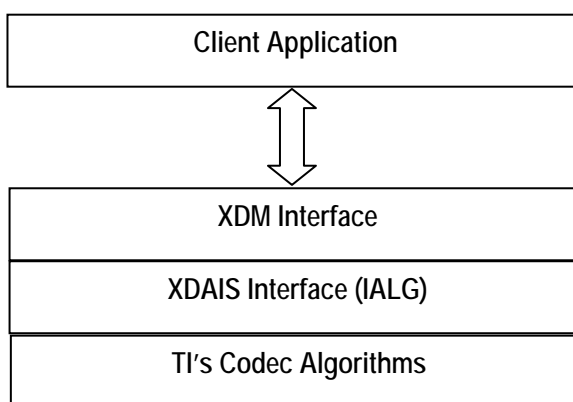
(for example audio, video, image, and speech). The XDM standard defines the following two APIs:

- ❑ `control()`
- ❑ `process()`

The `control()` API provides a standard way to control an algorithm instance and receive status information from the algorithm in real-time. The `control()` API replaces the `algControl()` API defined as part of the IALG interface. The `process()` API does the basic processing (encode/decode) of data.

Apart from defining standardized APIs for multimedia codecs, XDM also standardizes the generic parameters that the client application must pass to these APIs. The client application can define additional implementation specific parameters using extended data structures.

The following figure depicts the XDM interface to the client application.



As depicted in the figure, XDM is an extension to XDAIS and forms an interface between the client application and the codec component. XDM insulates the client application from component-level changes. Since TI's multimedia algorithms are XDM-compliant, it provides you with the flexibility to use any TI algorithm without changing the client application code. For example, if you have developed a client application using an XDM-compliant MPEG4 video decoder, then you can easily replace MPEG4 with another XDM-compliant video decoder, say H.263, with minimal changes to the client application.

For more details, see *eXpressDSP Digital Media (XDM) Standard API Reference* (literature number SPRUEC8).

1.2 Overview of AAC Encoder

AAC is one of the most popular audio compression standards across wide spectrum of application ranging from portable player, cell phones, music systems, internet, and so on.

1.3 Supported Services and Features

This user guide accompanies TI's implementation of AAC Encoder on the C64x+ platform.

This version of the codec has the following supported features:

- ❑ Supports 16-bit and 32-bit PCM samples as input. In case of 32-bit PCM it takes the most significant 16-bits as input internally
- ❑ Supports constant Bit-rate (CBR) encoding and Variable Bit-rate (VBR) encoding
- ❑ Supports input sampling frequencies from 8 kHz to 96 kHz for AAC-LC encoding
- ❑ Supports input sampling frequencies from 16 kHz to 48 kHz for AAC-HE encoding
- ❑ Supports AAC-LC, AAC-HE and AAC-HEv2 (Parametric stereo) output format
- ❑ Supports mono, stereo and dual mono input files
- ❑ Supports bit-rates based on sampling frequency and number of channels
- ❑ Supports Audio Data Interchange Format (ADIF), Audio Data Transport Stream (ADTS), and raw output format
- ❑ Compliant with the ISO/IEC 14496-3 (MPEG 4 AAC) and ISO/IEC 13818-7 (MPEG 2-AAC) standards
- ❑ eXpressDSP Digital Media (XDM 1.0 IAUDENC1) interface compliant

Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing the codec component. It also provides information on building and running the sample test application.

Topic	Page
2.1 System Requirements	2-2
2.2 Installing the Component	2-2
2.3 Before Building the Sample Test Application	2-4
2.4 Building and Running the Sample Test Application	2-4
2.5 Configuration Files	2-5
2.6 Standards Conformance and User-Defined Inputs	2-6
2.7 Uninstalling the Component	2-7
2.8 Evaluation Version	2-7

2.1 System Requirements

This section describes the hardware and software requirements for the normal functioning of the codec component.

2.1.1 Hardware

This codec has been built and tested on the DM644x EVM with XDS560 USB.

This codec can be used on any of TI's C64x+ based platforms such as DM644x, DM648, DM643x, OMAP35xx and their derivatives.

2.1.2 Software

The following are the software requirements for the normal functioning of the codec:

- ❑ **Development Environment:** This project is developed using Code Composer Studio version 3.2.37.12.
- ❑ **Code Generation Tools:** This project is compiled, assembled, archived, and linked using the code generation tools version 6.0.8.

2.2 Installing the Component

The codec component is released as a compressed archive. To install the codec, extract the contents of the zip file onto your local hard disk. The zip file extraction creates a top-level directory called 100_A_AAC_E_1_20_00, under which another directory named C64XPLUS_HE is created.

Figure 2-1 shows the sub-directories created in the C64XPLUS_HE directory.

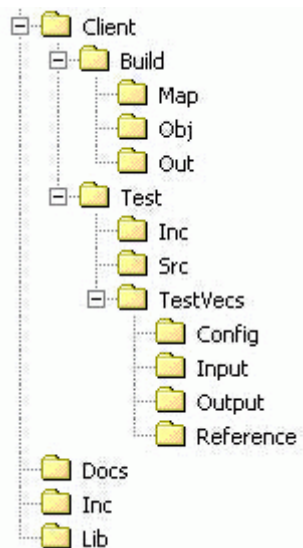


Figure 2-1. Component Directory Structure

Table 2-1 provides a description of the sub-directories created in the C64XPLUS_HE directory.

Note:

If you are installing an evaluation version of this codec, the directory name will be 100E_A_AAC_E_1_20_00.

Table 2-1. Component Directories

Sub-Directory	Description
\Inc	Contains XDM related header files which allow interface to the codec library
\Lib	Contains the codec library file
\Docs	Contains user guide and datasheet
\Client\Build	Contains the sample test application project (.pj1) file
\Client\Build\Map	Contains the memory map generated on compilation of the code
\Client\Build\Obj	Contains the intermediate .asm and/or .obj file generated on compilation of the code
\Client\Build\Out	Contains the final application executable (.out) file generated by the sample test application
\Client\Test\Src	Contains application C files
\Client\Test\Inc	Contains header files needed for the application code
\Client\Test\TestVecs\Input	Contains input test vectors
\Client\Test\TestVecs\Output	Contains output generated by the codec
\Client\Test\TestVecs\Reference	Contains read-only reference output to be used for verifying against codec output
\Client\Test\TestVecs\Config	Contains configuration parameter files

2.3 Before Building the Sample Test Application

This codec is accompanied by a sample test application. To run the sample test application, you need DSP/BIOS.

This version of the codec has been validated with DSP/BIOS version 5.31.

2.3.1 Installing DSP/BIOS

You can download DSP/BIOS from the TI external website:

https://www-a.ti.com/downloads/sds_support/targetcontent/bios/index.html

Install DSP/BIOS at the same location where you have installed Code Composer Studio. For example:

<install directory>\CCStudio_v3.2

The sample test application uses the following DSP/BIOS files:

- ❑ Header file, bcache.h available in the <install directory>\CCStudio_v3.2<bios_directory>\packages\ti\bios\include directory.
- ❑ Library file, biosDM420.a64P available in the <install directory>\CCStudio_v3.2<bios_directory>\packages\ti\bios\lib directory.

2.4 Building and Running the Sample Test Application

The sample test application that accompanies this codec component will run in TI's Code Composer Studio development environment. To build and run the sample test application in Code Composer Studio, follow these steps:

- 1) Verify that you have installed TI's Code Composer Studio version 3.2.37.12 and code generation tools version 6.0.8.
- 2) Verify that the codec object library aacenc_tii_he.l64P exists in the \Lib sub-directory.
- 3) Open the test application project file, TestAppEncoder.pjt in Code Composer Studio. This file is available in the \Client\Build sub-directory.
- 4) Select **Project > Build** to build the sample test application. This creates an executable file, TestAppEncoder.out in the \Client\Build\Out sub-directory.
- 5) Select **File > Load**, browse to the \Client\Build\Out sub-directory, select the codec executable created in step 4, and load it into Code Composer Studio in preparation for execution.
- 6) Select **Debug > Run** to execute the sample test application.

The sample test application takes the input files stored in the \Client\Test\TestVecs\Input sub-directory, runs the codec, and uses the

reference files stored in the \Client\Test\TestVecs\Reference sub-directory to verify that the codec is functioning as expected and dumps the output in the \Client\Test\TestVecs\Output directory.

- 7) On successful completion, the application displays one of the following message for each frame:
 - “Encoder compliance test passed/failed”(for compliance check mode)
 - “Encoder output dump completed”(for output dump mode)

2.5 Configuration Files

This codec is shipped along with:

- Generic configuration file (Testvecs.cfg) – specifies input and output files for the sample test application.
- Encoder configuration file (Testparams.cfg) – specifies the configuration parameters used by the test application to configure the Encoder.

2.5.1 Generic Configuration File

The sample test application shipped along with the codec uses the configuration file, Testvecs.cfg for determining the input and output files for running the codec. The Testvecs.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

The format of the Testvecs.cfg file is:

```
X
Input
Output/Reference
```

where:

- X may be set as:
 - 1 - for compliance checking, no output file is created
 - 0 - for writing the output to the output file
- Input is the input file name (use complete path).
- Output/Reference is the output file name (if X is 0) or reference file name (if X is 1).

A sample Testvecs.cfg file is as shown:

```
1
..\..\Test\TestVecs\Input\input.wav
..\..\Test\TestVecs\Reference\output_he_ref.aac
0
..\..\Test\TestVecs\Input\input.wav
..\..\Test\TestVecs\Output\output.aac
```

2.5.2 Encoder Configuration File

The encoder configuration file, Testparams.cfg contains the configuration parameters required for the encoder. The Testparams.cfg file is available in the \Client\Test\TestVecs\Config sub-directory.

A sample Testparams.cfg file is as shown:

```
# Input File Format is as follows
# <ParameterName> <ParameterValue> /* Comment */
#####
Parameters
#####
-b 128000          /* bit-rate */
```

2.6 Standards Conformance and User-Defined Inputs

To check the conformance of the codec for the default input file shipped along with the codec, follow the steps as described in Section 2.4.

To check the conformance of the codec for other input files of your choice, follow these steps:

- 1) Copy the input files to the \Client\Test\TestVecs\Inputs sub-directory.
- 2) Copy the reference files to the \Client\Test\TestVecs\Reference sub-directory.
- 3) Edit the configuration file, Testvecs.cfg available in the \Client\Test\TestVecs\Config sub-directory. For details on the format of the Testvecs.cfg file, see Section 2.5.1.
- 4) Execute the sample test application. On successful completion, the application displays one of the following message for each frame:
 - o "Encoder compliance test passed/failed" (if x is 1)
 - o "Encoder output dump completed" (if x is 0)

If you have chosen the option to write to an output file (x is 0), you can use any standard file comparison utility to compare the codec output with the reference output and check for conformance.

Note:

The comparison is valid only with a set of vectors provided as part of the release package.

2.7 Uninstalling the Component

To uninstall the component, delete the codec directory from your hard disk.

2.8 Evaluation Version

If you are using an evaluation version of this codec, an audible tone will be heard for every 50th frame.

This page is intentionally left blank

Sample Usage

This chapter provides a detailed description of the sample test application that accompanies this codec component.

3.1 Overview of the Test Application

The test application exercises the `IAUDENC1` base class of the AAC Encoder library. The main test application files are `TestAppEncoder.c` and `TestAppEncoder.h`. These files are available in the `\Client\Test\Src` and `\Client\Test\Inc` sub-directories respectively.

Figure 3-1 depicts the sequence of APIs exercised in the sample test application.

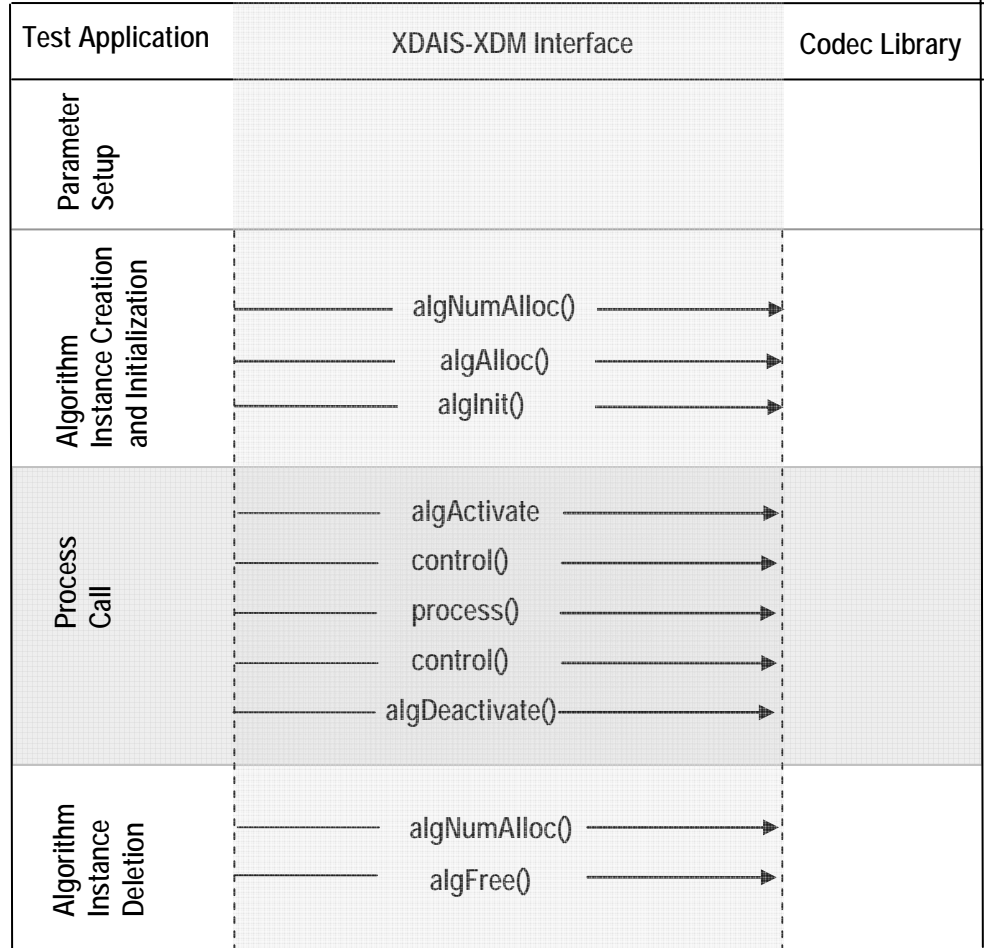


Figure 3-1. Test Application Sample Implementation

Note:

Audio codecs do not use `algActivate()` and `algDeactivate()` APIs.

The test application is divided into four logical blocks:

- ❑ Parameter setup
- ❑ Algorithm instance creation and initialization
- ❑ Process call
- ❑ Algorithm instance deletion

3.1.1 Parameter Setup

Each codec component requires various codec configuration parameters to be set at initialization. For example, a video codec requires parameters such as video height, video width, and so on. The test application obtains the required parameters from the Encoder configuration files.

In this logical block, the test application does the following:

- 1) Opens the generic configuration file, `config.txt` and reads the compliance checking parameter, input file name, and output file name.
For more details on the configuration files, see Section 2.5.
- 2) Reads the input bit-stream into the application input buffer.

After successful completion of these steps, the test application does the algorithm instance creation and initialization.

3.1.2 Algorithm Instance Creation and Initialization

In this logical block, the test application accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs are called in sequence:

- 1) `algNumAlloc()` - To query the algorithm about the number of memory records it requires.
- 2) `algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.
- 3) `algInit()` - To initialize the algorithm with the memory structures provided by the application.

A sample implementation of the create function that calls `algNumAlloc()`, `algAlloc()`, and `algInit()` in sequence is provided in the `ALG_create()` function implemented in the `alg_create.c` file.

3.1.3 Process Call

After algorithm instance creation and initialization, the test application does the following:

- 1) Sets the dynamic parameters (if they change during run-time) by calling the `control()` function with the `XDM_SETPARAMS` command.
- 2) Sets the input and output buffer descriptors required for the `process()` function call. The input and output buffer descriptors are obtained by calling the `control()` function with the `XDM_GETBUFINFO` command.
- 3) Calls the `process()` function to encode/decode a single frame of data. The behavior of the algorithm can be controlled using various dynamic parameters (see Section 4.2.1.8). The inputs to the `process()` function are input and output buffer descriptors, pointer to the `IAUDENC1_InArgs` and `IAUDENC1_OutArgs` structures.

There could be any ordering of `control()` and `process()` functions. The following APIs are called in sequence:

- 1) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.
- 2) `process()` - To call the Encoder with appropriate input/output buffer and arguments information.
- 3) `control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the six available control commands.

The do-while loop encapsulates frame level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts. It also protects the `process()` call from file operations by placing appropriate calls for cache operations. The test application does a cache invalidate for the valid input buffers before `process()` and a cache write back invalidate for output buffers after `process()`.

In the sample test application, after calling `process()`, the output data is either dumped to a file or compared with a reference file.

3.1.4 Algorithm Instance Deletion

Once encoding/decoding is complete, the test application deletes the current algorithm instance. The following APIs are called in sequence:

- 1) `algNumAlloc()` - To query the algorithm about the number of memory records it used.
- 2) `algFree()` - To query the algorithm to get the memory record information.

A sample implementation of the delete function that calls `algNumAlloc()` and `algFree()` in sequence is provided in the `ALG_delete()` function implemented in the `alg_create.c` file.

This page is intentionally left blank

API Reference

This chapter provides a detailed description of the data structures and interfaces functions used in the codec component.

Topic	Page
4.1 Symbolic Constants and Enumerated Data Types	4-2
4.2 Data Structures	4-8
4.3 Interface Functions	4-21

4.1 Symbolic Constants and Enumerated Data Types

This section summarizes all the symbolic constants specified as either #define macros and/or enumerated C data types. For each symbolic constant, the semantics or interpretation of the same is also provided.

Table 4-1. List of Enumerated Data Types

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
IAUDIO_ChannelMode	IAUDIO_1_0	Mono
	IAUDIO_2_0	Stereo
	IAUDIO_11_0	Dual Mono
	IAUDIO_3_0	Left, Right, Center. Not supported in this version of AAC Encoder.
	IAUDIO_2_1	Left, Right, Sur. Not supported in this version of AAC Encoder.
	IAUDIO_3_1	Left, Right, Center, Sur. Not supported in this version of AAC Encoder.
	IAUDIO_2_2	Left, Right, SurL, SurR. Not supported in this version of AAC Encoder.
	IAUDIO_3_2	Left, Right, Center, SurL, SurR. Not supported in this version of AAC Encoder.
	IAUDIO_2_3	Left, Right, SurL, SurR, surC. Not supported in this version of AAC Encoder.
	IAUDIO_3_3	Left, Right, Center, SurL, SurR, surC. Not supported in this version of AAC Encoder.
	IAUDIO_3_4	Left, Right, Center, SurL, SurR, sideL, sideR. Not supported in this version of AAC Encoder.
IAUDIO_PcmFormat	IAUDIO_BLOCK	Left channel data followed by right channel data. Note: For single channel (mono), right channel data will be same as left channel data.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	IAUDIO_INTERLEAVED	Left and right channel data interleaved. Note: For single channel (mono), right channel data will be same as left channel data.
IAUDIO_DualMonoMode	IAUDIO_DUALMONO_LR	Play / Encode both left and right channel
	IAUDIO_DUALMONO_LEFT	Play / Encode only left channel
	IAUDIO_DUALMONO_RIGHT	Play / Encode only right channel
	IAUDIO_DUALMONO_LR_MIX	Mix and play
IAUDIO_EncMode	IAUDIO_CBR	Constant bit-rate
	IAUDIO_VBR	Variable bit-rate
XDM_DataFormat	XDM_BYTE	Big endian stream
	XDM_LE_16	16-bit little endian stream
	XDM_LE_32	32-bit little endian stream
	XDM_LE_64	64 bit little endian stream
	XDM_BE_16	16 bit big endian stream
	XDM_BE_32	32 bit big endian stream
	XDM_BE_64	64 bit big endian stream
XDM_CmdId	XDM_GETSTATUS	Query algorithm instance to fill <i>Status</i> structure
	XDM_SETPARAMS	Set run-time dynamic parameters through the <i>DynamicParams</i> structure
	XDM_RESET	Reset the algorithm
	XDM_SETDEFAULT	Initialize all fields in <i>Params</i> structure to default values specified in the library
	XDM_FLUSH	Handle end of stream conditions. This command forces algorithm instance to output data without additional input. This command is not used for this version of AAC Encoder.
	XDM_GETBUFINFO	Query algorithm instance regarding the properties of input and output buffers
	XDM_GETVERSION	Query the algorithms version
XDM_AccessMode	XDM_ACCESSMODE_READ	The algorithm reads from the buffer using the CPU.

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_ACCESSMODE_WRITE	The algorithm writes to the buffer using the CPU.
AACENC_OBJ_TYP	AACENC_OBJ_TYP_LC	AAC low complexity <input type="checkbox"/> 2 - Low complexity
	AACENC_OBJ_TYP_HEAAC	AAC Encoder with SBR capability <input type="checkbox"/> 5 - SBR capability
	AACENC_OBJ_TYP_PS	AAC Encoder with SBR and PS <input type="checkbox"/> 29 - SBR and PS capability
AACENC_VBR_MODE	AACENC_BR_MODE_VBR_1	Variable bit-rate mode-1 <input type="checkbox"/> 1 - VBR Mode 1
	AACENC_BR_MODE_VBR_2	Variable Bit-rate Mode-2 <input type="checkbox"/> 2 - VBR Mode 2
	AACENC_BR_MODE_VBR_3	Variable Bit-rate Mode-3 <input type="checkbox"/> 3 - VBR Mode 3
	AACENC_BR_MODE_VBR_4	Variable Bit-rate Mode-4 <input type="checkbox"/> 4 - VBR Mode 4
	AACENC_BR_MODE_VBR_5	Variable Bit-rate Mode-5 <input type="checkbox"/> 5 - VBR Mode 5
AACENC_TRANSPORT_TYPE	AACENC_TT_RAW	<input type="checkbox"/> 0 - Raw output format
	AACENC_TT_ADIF	<input type="checkbox"/> 1 - ADIF file format
	AACENC_TT_ADTS	<input type="checkbox"/> 2 - ADTS file format
AACENC_BOOL_TYPE	AACENC_FALSE	<input type="checkbox"/> 0 - False
	AACENC_TRUE	<input type="checkbox"/> 1 - True
XDM_ErrorBit		The bit fields in the 32-bit error code are interpreted as shown
	XDM_PARAMSCHANGE	Bit 8 <input type="checkbox"/> 1 - Sequence parameters change <input type="checkbox"/> 0 - Ignore Not applicable for AAC Encoder.
	XDM_APPLIEDCONCEALMENT	Bit 9 <input type="checkbox"/> 1 - Applied concealment <input type="checkbox"/> 0 - Ignore Not applicable for AAC Encoder.
	XDM_INSUFFICIENTDATA	Bit 10 <input type="checkbox"/> 1 - Insufficient input data <input type="checkbox"/> 0 - Ignore

Group or Enumeration Class	Symbolic Constant Name	Description or Evaluation
	XDM_CORRUPTEDDATA	Bit 11 <input type="checkbox"/> 1 - Invalid data <input type="checkbox"/> 0 - Ignore
	XDM_CORRUPTEDHEADER	Bit 12 <input type="checkbox"/> 1 - Corrupted frame header <input type="checkbox"/> 0 - Ignore
	XDM_UNSUPPORTEDINPUT	Bit 13 <input type="checkbox"/> 1 - Unsupported feature/parameter in input <input type="checkbox"/> 0 - Ignore
	XDM_UNSUPPORTEDPARAM	Bit 14 <input type="checkbox"/> 1 - Unsupported input parameter or configuration <input type="checkbox"/> 0 - Ignore
	XDM_FATALERROR	Bit 15 <input type="checkbox"/> 1 - Fatal error (stop decoding) <input type="checkbox"/> 0 - Recoverable error

Note:

The remaining bits that are not mentioned in `XDM_ErrorBit` are interpreted as:

- Bit 16-32: Reserved
- Bit 0-7: Codec and implementation specific (see Table 4-2)

The algorithm can set multiple bits to 1 depending on the error condition.

The AAC Encoder specific error status messages are listed in Table 4-2. The value column indicates the decimal value of the last 8-bits reserved for codec specific error statuses.

Table 4-2. AAC Encoder Error Status

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
XDM_ErrorBit	AACENC_INVALID_PARAM	1	Invalid parameter for the encoder
	AACENC_INVALID_FREQ	2	Invalid input sampling frequency
	AACENC_INVALID_BITRATE	3	Invalid output bit-rate
	AACENC_INVALID_CHANNELS	4	Invalid number of channels
	AACENC_INVALID_ELTYPE	5	Invalid element type
	AACENC_INSUFFICIENT_INPUT	6	Insufficient input PCM data

Group or Enumeration Class	Symbolic Constant Name	Value	Description or Evaluation
	AACENC_WRITEBITSTREAM_ERROR	7	Error during writing bit-stream
	AACENC_ANCILLARYDATA_ERROR	8	Error in ancillary data parameters
	AACENC_INVALID_OUTFORMAT	9	Invalid output format
	AACENC_INVALID_BLOCK	10	Invalid block type
	AACENC_DIV_BY_ZERO	11	Divide by zero error
	AACENC_NULL_POINTER	12	Null pointer error
	AACENC_CRC_ERROR	13	Cyclic redundancy check error
	AACENC_TNS_ERROR	14	TNS error
	AACENC_INVALID_SFB	15	Invalid scale factor band
	AACENC_QUANTIZATION_ERROR	16	Quantization error
	AACENC_PSY_THRESHOLD_ERROR	17	Psychoacoustic threshold calculation error
	AACENC_DYNAMICBITCOUNT_ERROR	18	Dynamic bit-count calculation error
	AACENC_WRITEADTSHEADER_ERROR	19	Error while writing ADTS header
	AACENC_WRITEADIFHEADER_ERROR	20	Error while writing ADIF header
	AACENC_OUTBUF_TOO_SMALL	21	Number of output bytes is greater than output buffer size
	AACENC_ERROR	22	Unspecified error

The following errors are fatal errors and the application has to reset the encoder with correct dynamic parameter values:

- ❑ AACENC_WRITEBITSTREAM_ERROR
- ❑ AACENC_DIV_BY_ZERO
- ❑ AACENC_NULL_POINTER
- ❑ AACENC_ERROR
- ❑ AACENC_INVALID_BLOCK
- ❑ AACENC_DIV_BY_ZERO
- ❑ AACENC_TNS_ERROR
- ❑ AACENC_INVALID_SFB
- ❑ AACENC_QUANTIZATION_ERROR
- ❑ AACENC_PSY_THRESHOLD_ERROR
- ❑ AACENC_DYNAMICBITCOUNT_ERROR
- ❑ AACENC_WRITEADTSHEADER_ERROR
- ❑ AACENC_WRITEADIFHEADER_ERROR

4.2 Data Structures

This section describes the XDM defined data structures that are common across codec classes. These XDM data structures can be extended to define any implementation specific parameters for a codec component.

4.2.1 Common XDM Data Structures

This section includes the following common XDM data structures:

- ❑ XDM_BufDesc
- ❑ XDM_SingleBufDesc
- ❑ XDM1_SingleBufDesc
- ❑ XDM1_BufDesc
- ❑ XDM_AlgBufInfo
- ❑ IAUDENC1_Fxns
- ❑ IAUDENC1_Params
- ❑ IAUDENC1_DynamicParams
- ❑ IAUDENC1_InArgs
- ❑ IAUDENC1_Status
- ❑ IAUDENC1_OutArgs

4.2.1.1 XDM_BufDesc

|| Description

This structure defines the buffer descriptor for input and output buffers. These buffer descriptors are not used in XDM 1.0.

|| Fields

Field	Data Type	Input/ Output	Description
**bufs	XDAS_Int8	Input	Pointer to the vector containing buffer addresses
numBufs	XDAS_Int32	Input	Number of buffers
*bufSizes	XDAS_Int32	Input	Size of each buffer in bytes

4.2.1.2 XDM_SingleBufDesc

|| Description

This structure defines the single buffer descriptor for input and output buffers.

|| Fields

Field	Data Type	Input/ Output	Description
*buf	XDAS_Int8	Input	Pointer to a buffer address
bufSize	XDAS_Int32	Input	Size of each buffer in bytes

4.2.1.3 XDM1_BufDesc

|| Description

This structure defines the buffer descriptor for input and output buffers.

|| Fields

Field	Data Type	Input/ Output	Description
numBufs	XDAS_Int32	Input	Number of buffers
descs[XDM_MAX_IO_BUFFERS]	XDM1_SingleBufDesc	Input	Buffer descriptors

4.2.1.4 XDM1_SingleBufDesc

|| Description

This structure defines the single buffer descriptor for input and output buffers.

|| Fields

Field	Data Type	Input/ Output	Description
*buf	XDAS_Int8	Input	Pointer to a buffer address
bufSize	XDAS_Int32	Input	Size of each buffer in bytes
accessMask	XDAS_Int32	Output	Mask filled by the algorithm

4.2.1.5 XDM_AlgBufInfo

|| Description

This structure defines the buffer information descriptor for input and output buffers. This structure is filled when you invoke the `control()` function with the `XDM_GETBUFINFO` command.

|| Fields

Field	Data Type	Input/ Output	Description
minNumInBufs	XDAS_Int32	Output	Number of input buffers
minNumOutBufs	XDAS_Int32	Output	Number of output buffers
minInBufSize[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Output	Size in bytes required for each input buffer
minOutBufSize[XDM_MAX_IO_BUFFERS]	XDAS_Int32	Output	Size in bytes required for each output buffer

Note:

For AAC Encoder, the buffer details are:

- ❑ Number of input buffer required is 1.
- ❑ Number of output buffer required is 1.
- ❑ The size of the input buffer should be such that, atleast one frame of input PCM samples is present in the input buffer. The input buffer size (in words) is 2048 samples per channel.
- ❑ The output buffer size (in bytes) for worst case is 1536 bytes.

These are the maximum buffer sizes but you can re-configure depending on the format of the bit-stream.

4.2.1.6 IAUDENC1_Fxns**|| Description**

This structure contains pointers to all the XDAIS and XDM interface functions.

|| Fields

Field	Data Type	Input/ Output	Description
ialg	IALG_Fxns	Input	Structure containing pointers to all the XDAIS interface functions. For more details, see <i>TMS320 DSP Algorithm Standard API Reference</i> (literature number SPRU360).
*process	XDAS_Int32	Input	Pointer to the <code>process()</code> function
*control	XDAS_Int32	Input	Pointer to the <code>control()</code> function

4.2.1.7 IAUDENC1_Params

|| Description

This structure defines the creation parameters for an algorithm instance object. Set this data structure to `NULL`, if you are not sure of the values to specify for these parameters.

|| Fields

Field	Data Type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes. Default Value is size of (IAACENC_Params).
sampleRate	XDAS_Int32	Input	Sampling Frequency in Hz. Default value is 44100.
bitRate	XDAS_Int32	Input	Average bit-rate, in bits per second. Default value is 128000.
channelMode	XDAS_Int32	Input	Input Channel Mode. See <code>IAUDIO_ChannelMode</code> for details. Default value is <code>IAUDIO_2_0</code> .
dataEndianness	XDAS_Int32	Input	Endianness of input data. See <code>XDM_DataFormat</code> enumeration for details. Default value is <code>XDM_LE_16</code> .
encMode	XDAS_Int32	Input	Encoding mode. See <code>IAUDIO_EncMode</code> for details. Default value is <code>IAUDIO_CBR</code> .
inputFormat	XDAS_Int32	Input	Input PCM format. See <code>IAUDIO_PcmFormat</code> enumeration for details. Default value is <code>IAUDIO_INTERLEAVED</code> .
inputBitsPerSample	XDAS_Int32	Input	Number of bits per input PCM sample. Default value is 16.
maxBitRate	XDAS_Int32	Input	Maximum bit-rate in case of VBR. Default value is 192000.
dualMonoMode	XDAS_Int32	Input	Mode to indicate type of dual mono. Applicable only if <code>channelMode</code> is <code>dualMono</code> . See <code>IAUDIO_DualMonoMode</code> for details. Default value is <code>IAUDIO_DUALMONO_LR</code> .
crcFlag	XDAS_Int32	Input	Flag indicating whether the encoder should insert CRC bits into the bit-stream or not. Default value is <code>XDAS_FALSE</code> .
ancFlag	XDAS_Int32	Input	Ancillary Data Flag. Default value is <code>XDAS_FALSE</code> .

Field	Data Type	Input/ Output	Description
lfeFlag	XDAS_Int32	Input	Flag indicating whether LFE channel data is present or not in the input. Default value is XDAS_FALSE.

Note:

- ❑ AAC-LC Encoder supports `sampleRate` from 8kHz to 96kHz.
- ❑ The supported `bitRate` range is from 8kbps to 576kbps for LC and 8kbps to 128kbps for HE. See Appendix A for details.
- ❑ Currently, the AAC Encoder implementation supports `XDM_LE_16` and `XDM_LE_32` format.
- ❑ The supported `maxBitrate` in VBR case is 192 kbps
- ❑ Supports a maximum of two input channels, mono, stereo and dual mono.
- ❑ AAC Encoder does not support LFE.
- ❑ Supports both `IAUDIO_BLOCK` and `IAUDIO_INTERLEAVED` `pcmFormats`.

4.2.1.8 IAUDENC1_DynamicParams

|| Description

This structure defines the run-time parameters for an algorithm instance object. Set this data structure to `NULL`, if you are not sure of the values to be specified for these parameters.

|| Fields

Field	Data Type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
bitRate	XDAS_Int32	Input	Average bit-rate in bits per second. Default value is 128000.
sampleRate	XDAS_Int32	Input	Sampling frequency in Hertz. Default value is 44100.
channelMode	XDAS_Int32	Input	Number of channels. See <code>IAUDIO_ChannelMode</code> enumeration for details. Default value is <code>IAUDIO_2_0</code> .
lfeFlag	XDAS_Int32	Input	Flag indicating whether LFE channel data is present or not in the input. Default value is <code>XDAS_FALSE</code> .
dualMonoMode	XDAS_Int32	Input	Mode to indicate type of Dual Mono. Applicable only if <code>channelMode</code> is <code>dualMono</code> . Default value is <code>IAUDIO_DUALMONO_LR</code> .
inputBitsPerSample	XDAS_Int32	Input	Number of bits per input PCM Sample. Default value is 16.

Note:

- ❑ This encoder supports 16-bit per input sample and 32-bit per input sample. In case of 32-bits per input sample only the MSB 16-bits are considered.
- ❑ Encoder does not support LFE channel.
- ❑ Supports a maximum of two input channels, mono, stereo and dual mono.
- ❑ Supports both 16 and 32-bits per sample.

4.2.1.9 IAUDENC1_InArgs

|| Description

This structure defines the run-time input arguments for an algorithm instance object.

|| Fields

Field	Data Type	Input/Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
numInSamples	XDAS_Int32	Input	Number of input samples per channel.
ancData	XDM1_SingleBufDesc	Input	Ancillary data

Note:

ancData will be interpreted only if ancFlag in IAUDENC1_Params is set.

4.2.1.10 IAUDENC1_Status

|| Description

This structure defines parameters that describe the status of the algorithm instance object.

|| Fields

Field	Data Type	Input/Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error enumeration for XDM compliant encoders and decoders. See XDM_ErrorBit enumeration for details.
data	XDM1_SingleBufDesc	Output	Buffer descriptor for data passing
bufInfo	XDM_AlgBufInfo	Output	Input and output buffer information. See XDM_AlgBufInfo data structure for details.
validFlag	XDAS_Int32	Output	Flag indicating the validity of the status structure.

Field	Data Type	Input/ Output	Description
lfeFlag	XDAS_Int32	Output	Flag indicating whether LFE channel data is present or not in the input.
bitRate	XDAS_Int32	Output	Average bit-rate, in bits per second
sampleRate	XDAS_Int32	Output	Sampling frequency in Hz
channelMode	XDAS_Int32	Output	Input channel mode. See <code>IAUDIO_ChannelMode</code> for details.
encMode	XDAS_Int32	Output	Encoding mode. See <code>IAUDIO_EncMode</code> for details.

Note:

Current implementation of AAC Encoder does not update data field.

4.2.1.11 IAUDENC1_OutArgs**|| Description**

This structure defines the run-time output arguments for the algorithm instance object.

|| Fields

Field	Data Type	Input/ Output	Description
size	XDAS_Int32	Input	Size of the basic or extended (if being used) data structure in bytes.
extendedError	XDAS_Int32	Output	Extended error enumeration for XDM compliant encoders and decoders. See <code>XDM_ErrorBit</code> enumeration for details.
bytesGenerated	XDAS_Int32	Output	Bytes generated during the process call.
numZeroesPadded	XDAS_Int32	Output	Number of zeroes padded to input.
numInSamples	XDAS_Int32	Output	Number of input samples per channel consumed by the encoder.

Note:

Encoder returns `numZeroesPadded` in bytes.

4.2.2 AAC Encoder Data Structures

This section includes the following AAC Encoder specific extended data structures:

- ❑ IAACENC_Params
- ❑ IAACENC_DynamicParams
- ❑ IAACENC_InArgs
- ❑ IAACENC_Status
- ❑ IAACENC_OutArgs

4.2.2.1 IAACENC_Params

|| Description

This structure defines the creation parameters and any other implementation specific parameters for the AAC Encoder instance object. The creation parameters are defined in the table below.

|| Fields

Field	Data Type	Input/Output	Description
audenc_params	IAUDENC1_Params	Input	See IAUDENC1_Params data structure for details.
outObjectType	AACENC_OBJ_TYP	Input	Output object type LC, HE or HEV2. See AACENC_OBJ_TYP enumeration for details. Default value is AACENC_OBJ_TYP_PS.
outFileFormat	AACENC_TRANSPORT_TYPE	Input	Output file format. See AACENC_TRANSPORT_TYPE enumeration for details. Default value is AACENC_TT_ADTS.
useTns	AACENC_BOOL_TYPE	Input	Flag for activating TNS feature. See AACENC_BOOL_TYPE enumeration for details. Default value is AACENC_TRUE.
usePns	AACENC_BOOL_TYPE	Input	Flag for activating PNS feature. See AACENC_BOOL_TYPE enumeration for details. Default value is AACENC_TRUE.
downMixFlag	AACENC_BOOL_TYPE	Input	Flag for enabling down mixing of channels. See AACENC_BOOL_TYPE enumeration for details. Default value is AACENC_FALSE.
bitRateMode	AACENC_VBR_MODE	Input	Type of VBR mode. See AACENC_VBR_MODE for details. Default value is AACENC_BR_MODE_VBR_1.
ancRate	XDAS_Int32	Input	Ancillary data rate. Default value is -1.

Note:

- ❑ `outFileFormat` supports Audio Data Interchange Format (ADIF), Audio Data Transport Stream (ADTS), and raw output format.
- ❑ `DownMixFlag` is supported only for LC.
- ❑ Encoder will switch of PNS in case of HEAAC and PS. So `usePns` will not have any impact.
- ❑ `bitRateMode` will be interpreted only if `encMode` from `IAUDENC1_Params` is `IAUDIO_VBR`.
- ❑ In case of VBR mode,
 - VBR Mode 1 => Low quality
 - VBR Mode 5 => Very high quality

The quality and the bit-rate increases from VBR Mode 1 to VBR Mode 5.
- ❑ `ancRate` should be less than or equal to 15% of `bitRate`, subject to an absolute maximum value of 19199.

4.2.2.2 IAACENC_DynamicParams**|| Description**

This structure defines the run-time parameters and any other implementation specific parameters for the AAC Encoder instance object. The run-time parameters are defined in the following table.

|| Fields

Field	Data Type	Input/ Output	Description
<code>audenc_dynamicparams</code>	<code>IAUDENC1_DynamicParams</code>	Input	See <code>IAUDENC1_DynamicParams</code> data structure for details.
<code>useTns</code>	<code>AACENC_BOOL_TYPE</code>	Input	Flag for activating TNS feature. See <code>AACENC_BOOL_TYPE</code> enumeration for details. Default value is <code>AACENC_TRUE</code> .
<code>usePns</code>	<code>AACENC_BOOL_TYPE</code>	Input	Flag for activating PNS feature. See <code>AACENC_BOOL_TYPE</code> enumeration for details. Default value is <code>AACENC_TRUE</code> .
<code>downMixFlag</code>	<code>AACENC_BOOL_TYPE</code>	Input	Flag for enabling down sampling. See <code>AACENC_BOOL_TYPE</code> enumeration for details. Default value is <code>AACENC_FALSE</code> .

Field	Data Type	Input/ Output	Description
ancFlag	AACENC_BOOL_TYPE	Input	Ancillary data flag. See AACENC_BOOL_TYPE enumeration for details. Default value is AACENC_FALSE.
ancRate	XDAS_Int32	Input	Ancillary data rate. Default value is -1.

Note:

ancRate should be less than or equal to 15% of bitRate, subject to an absolute maximum value of 19199.

4.2.2.3 IAACENC_InArgs**|| Description**

This structure defines the run-time input arguments for the AAC Encoder instance object.

|| Fields

Field	Data Type	Input/ Output	Description
audenc_inArgs	IAUDENC1_InArgs	Input	See IAUDENC1_InArgs data structure for details.

4.2.2.4 IAACENC_Status**|| Description**

This structure defines parameters that describe the status of the AAC Encoder and any other implementation specific parameters. The status parameters are defined in the table below.

|| Fields

Field	Data Type	Input/ Output	Description
audenc_status	IAUDENC1_Status	Output	See IAUDENC1_Status data structure for details.
outputObjectType	AACENC_OBJ_TYPE	Output	Output object type LC, HE or HEv2. See AACENC_OBJ_TYPE enumeration for details.
outFileFormat	AACENC_TRANSPORT_TYPE	Output	Output file format. See AACENC_TRANSPORT_TYPE enumeration for details.

4.2.2.5 IAACENC_OutArgs

|| Description

This structure defines the run-time output arguments for the AAC Encoder instance object.

|| Fields

Field	Data Type	Input/ Output	Description
audenc_outArgs	IAUDENC1_OutArgs	Output	See IAUDENC1_OutArgs data structure for details.

4.3 Interface Functions

This section describes the Application Programming Interfaces (APIs) used in the AAC Encoder. The APIs are logically grouped into the following categories:

- ❑ **Creation** – `algNumAlloc()`, `algAlloc()`
- ❑ **Initialization** – `algInit()`
- ❑ **Control** – `control()`
- ❑ **Data processing** – `algActivate()`, `process()`, `algDeactivate()`
- ❑ **Termination** – `algFree()`

You must call these APIs in the following sequence:

- 1) `algNumAlloc()`
- 2) `algAlloc()`
- 3) `algInit()`
- 4) `algActivate()`
- 5) `process()`
- 6) `algDeactivate()`
- 7) `algFree()`

`control()` can be called any time after calling the `algInit()` API.

`algNumAlloc()`, `algAlloc()`, `algInit()`, `algActivate()`, `algDeactivate()`, and `algFree()` are standard XDAIS APIs. This document includes only a brief description for the standard XDAIS APIs. For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

Note:

Audio codecs do not use `algActivate()` and `algDeactivate()` APIs.

4.3.1 Creation APIs

Creation APIs are used to create an instance of the component. The term creation could mean allocating system resources, typically memory.

|| Name

`algNumAlloc()` – determine the number of buffers that an algorithm requires

|| Synopsis

```
XDAS_Int32 algNumAlloc(Void);
```

|| Arguments

Void

|| Return Value

```
XDAS_Int32; /* number of buffers required */
```

|| Description

`algNumAlloc()` returns the number of buffers that the `algAlloc()` method requires. This operation allows you to allocate sufficient space to call the `algAlloc()` method.

`algNumAlloc()` may be called at any time and can be called repeatedly without any side effects. It always returns the same result. The `algNumAlloc()` API is optional.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

|| See Also

`algAlloc()`

|| Name

`algAlloc()` – determine the attributes of all buffers that an algorithm requires

|| Synopsis

```
XDAS_Int32 algAlloc(const IALG_Params *params, IALG_Fxns
**parentFxns, IALG_MemRec memTab[]);
```

|| Arguments

```
IALG_Params *params; /* algorithm specific attributes */
```

```
IALG_Fxns **parentFxns; /* output parent algorithm
functions */
```

```
IALG_MemRec memTab[]; /* output array of memory records */
```

|| Return Value

```
XDAS_Int32 /* number of buffers required */
```

|| Description

`algAlloc()` returns a table of memory records that describe the size, alignment, type, and memory space of all buffers required by an algorithm. If successful, this function returns a positive non-zero value indicating the number of records initialized.

The first argument to `algAlloc()` is a pointer to a structure that defines the creation parameters. This pointer may be `NULL`; however, in this case, `algAlloc()` must assume default creation parameters and must not fail.

The second argument to `algAlloc()` is an output parameter. `algAlloc()` may return a pointer to its parent's IALG functions. If an algorithm does not require a parent object to be created, this pointer must be set to `NULL`.

The third argument is a pointer to a memory space of size `nbufs * sizeof(IALG_MemRec)` where, `nbufs` is the number of buffers returned by `algNumAlloc()` and `IALG_MemRec` is the buffer-descriptor structure defined in `ialg.h`.

After calling this function, `memTab[]` is filled up with the memory requirements of an algorithm.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

|| See Also

```
algNumAlloc(), algFree()
```

4.3.2 Initialization API

Initialization API is used to initialize an instance of the algorithm. The initialization parameters are defined in the `Params` structure (see *Data Structures* section for details).

|| Name

algInit() – initialize an algorithm instance

|| Synopsis

```
XDAS_Int32 algInit(IALG_Handle handle, IALG_MemRec  
memTab[], IALG_Handle parent, IALG_Params *params);
```

|| Arguments

```
IALG_Handle handle; /* algorithm instance handle*/  
IALG_memRec memTab[]; /* array of allocated buffers */  
IALG_Handle parent; /* handle to the parent instance */  
IALG_Params *params; /* algorithm initialization  
parameters */
```

|| Return Value

```
IALG_EOK; /* status indicating success */  
IALG_EFAIL; /* status indicating failure */
```

|| Description

algInit() performs all initialization necessary to complete the run-time creation of an algorithm instance object. After a successful return from algInit(), the instance object is ready to be used to process data.

The first argument to algInit() is a handle to an algorithm instance. This value is initialized to the base field of memTab[0].

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers allocated for an algorithm instance. The number of initialized records is identical to the number returned by a prior call to algAlloc().

The third argument is a handle to the parent instance object. If there is no parent object, this parameter must be set to NULL.

The last argument is a pointer to a structure that defines the algorithm initialization parameters.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

|| See Also

algAlloc(), algMoved()

4.3.3 Control API

Control API is used for controlling the functioning of the algorithm instance during run-time. This is done by changing the status of the controllable parameters of the algorithm during run-time. These controllable parameters are defined in the Status data structure (see Data Structures section for details).

|| Name

`control()` – change run-time parameters and query the status

|| Synopsis

```
XDAS_Int32 (*control) (IAUDENC1_Handle handle,
IAUDENC1_Cmd id, IAUDENC1_DynamicParams *params,
IAUDENC1_Status *status);
```

|| Arguments

```
IAUDENC1_Handle handle; /* algorithm instance handle */
IAUDENC1_Cmd id; /* algorithm specific control commands*/
IAUDENC1_DynamicParams *params /* algorithm run-time
parameters */
IAUDENC1_Status *status /* algorithm instance status
parameters */
```

|| Return Value

```
IALG_EOK; /* status indicating success */
IALG_EFAIL; /* status indicating failure */
```

|| Description

This function changes the run-time parameters of an algorithm instance and queries the algorithm's status. `control()` must only be called after a successful call to `algInit()` and must never be called after a call to `algFree()`.

The first argument to `control()` is a handle to an algorithm instance.

The second argument is an algorithm specific control command. See `XDM_CmdId` enumeration for details.

The third and fourth arguments are pointers to the `IAUDENC1_DynamicParams` and `IAUDENC1_Status` data structures respectively.

Note:

If you are using extended data structures, the third and fourth arguments must be pointers to the extended `DynamicParams` and `Status` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters.

|| Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- ❑ `control()` can only be called after a successful return from `algInit()` and `algActivate()`.
- ❑ If algorithm uses DMA resources, `control()` can only be called after a successful return from `DMAN3_init()`.
- ❑ `handle` must be a valid handle for the algorithm's instance object.

|| Postconditions

The following conditions are true immediately after returning from this function.

- ❑ If the control operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- ❑ If the control command is not recognized, the return value from this operation is not equal to `IALG_EOK`.

|| Example

See test application file, `TestAppEncoder.c` available in the `\Client\Test\Src` sub-directory.

|| See Also

`algInit()`, `algActivate()`, `process()`

Note:

Audio codecs do not use `algActivate()`, `algDeactivate()`, and `DMAN3_init()` APIs.

4.3.4 Data Processing API

Data processing API is used for processing the input data.

|| Name

`process()` – basic encoding/decoding call

|| Synopsis

```
XDAS_Int32 (*process)(IAUDENC1_Handle handle, XDM1_BufDesc
*inBufs, XDM1_BufDesc *outBufs, IAUDENC1_InArgs *inargs,
IAUDENC1_OutArgs *outargs);
```

|| Arguments

```
IAUDENC1_Handle handle; /* algorithm instance handle */
XDM1_BufDesc *inBufs; /* algorithm input buffer descriptor
*/
XDM1_BufDesc *outBufs; /* algorithm output buffer
descriptor */
IAUDENC1_InArgs *inargs /* algorithm run-time input
arguments */
IAUDENC1_OutArgs *outargs /* algorithm run-time output
arguments */
```

|| Return Value

```
IALG_EOK; /* status indicating success */
IALG_EFAIL; /* status indicating failure */
```

|| Description

This function does the basic encoding/decoding. The first argument to `process()` is a handle to an algorithm instance.

The second and third arguments are pointers to the input and output buffer descriptor data structures respectively (see `XDM1_BufDesc` data structure for details).

The fourth argument is a pointer to the `IAUDENC1_InArgs` data structure that defines the run-time input arguments for an algorithm instance object.

The last argument is a pointer to the `IAUDENC1_OutArgs` data structure that defines the run-time output arguments for an algorithm instance object.

Note:

If you are using extended data structures, the fourth and fifth arguments must be pointers to the extended `InArgs` and `OutArgs` data structures respectively. Also, ensure that the `size` field is set to the size of the extended data structure. Depending on the value set for the `size` field, the algorithm uses either basic or extended parameters.

|| Preconditions

The following conditions must be true prior to calling this function; otherwise, its operation is undefined.

- ❑ `process()` can only be called after a successful return from `algInit()` and `algActivate()`.
- ❑ If algorithm uses DMA resources, `process()` can only be called after a successful return from `DMAN3_init()`.
- ❑ `handle` must be a valid handle for the algorithm's instance object.
- ❑ Buffer descriptor for input and output buffers must be valid.
- ❑ Input buffers must have valid input data.

|| Postconditions

The following conditions are true immediately after returning from this function.

- ❑ If the process operation is successful, the return value from this operation is equal to `IALG_EOK`; otherwise it is equal to either `IALG_EFAIL` or an algorithm specific return value.
- ❑ After successful return from `process()` function, `algDeactivate()` can be called.

|| Example

See test application file, `TestAppEncoder.c` available in the `\Client\Test\Src` sub-directory.

|| See Also

`algInit()`, `algDeactivate()`, `control()`

Note:

- ❑ Audio codecs do not use `algActivate()`, `algDeactivate()`, and `DMAN3_init()` APIs.
- ❑ AAC Encoder supports 16-bit and 32-bit PCM samples in little endian format as input.

4.3.5 Termination API

Termination API is used to terminate the algorithm instance and free up the memory space that it uses.

|| Name

`algFree()` – determine the addresses of all memory buffers used by the algorithm

|| Synopsis

```
XDAS_Int32 algFree(IALG_Handle handle, IALG_MemRec  
memTab[]);
```

|| Arguments

```
IALG_Handle handle; /* handle to the algorithm instance */  
IALG_MemRec memTab[]; /* output array of memory records */
```

|| Return Value

```
XDAS_Int32; /* Number of buffers used by the algorithm */
```

|| Description

`algFree()` determines the addresses of all memory buffers used by the algorithm. The primary aim of doing so is to free up these memory regions after closing an instance of the algorithm.

The first argument to `algFree()` is a handle to the algorithm instance.

The second argument is a table of memory records that describe the base address, size, alignment, type, and memory space of all buffers previously allocated for the algorithm instance.

For more details, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

|| See Also

`algAlloc()`

This page is intentionally left blank

AAC LC Encoder Bit-rate and Sampling Frequency Combination

The following table contains information on AAC LC Encoder bit-rate and sampling frequency combination.

Sampling frequency	Mono/Stereo	Min Bit-Rate	Max Bit-Rate
8000	Mono	8000	48000
8000	Stereo	16000	96000
16000	Mono	8000	96000
16000	Stereo	16000	192000
22050	Mono	8000	132300
22050	Stereo	16000	264600
32000	Mono	8000	192000
32000	Stereo	16000	384000
44100	Mono	8000	264600
44100	Stereo	16000	529200
48000	Mono	8000	288000
48000	Stereo	16000	576000
96000	Mono	16000	288000
96000	Stereo	20000	576000

This page is intentionally left blank

AAC HE Encoder Bit-rate and Sampling Frequency Combination

The following table contains information on AAC HE Encoder bit-rate and sampling frequency combination.

Sampling Frequency	Mono/Stereo	Min Bit-Rate	Max Bit-Rate
16 kHz	Mono	8000	48000
22.05 kHz	Mono	8000	64000
24 kHz	Mono	8000	64000
32 kHz	Mono	8000	64000
44.1 kHz	Mono	12000	64000
48 kHz	Mono	12000	64000
16 kHz	Stereo	16000	128000
22.05 kHz	Stereo	16000	128000
24 kHz	Stereo	16000	128000
32 kHz	Stereo	16000	128000
44.1 kHz	Stereo	16000	128000
48 kHz	Stereo	16000	128000

Note:
 For AAC-HEv2, encoding, only stereo input files are valid. However, the bit-rates applicable to AAC-HEv2 are the ones specified for "Mono" in the table above.

This page is intentionally left blank

Revision History

This user guide revision history highlights the changes made to the SPRUET3 codec specific user guide to make it SPRUET3A.

Table C-1. Revision History of AAC High Efficiency Encoder on C64x+

Section	Additions/Modifications/Deletions
Global Changes	<ul style="list-style-type: none"> <input type="checkbox"/> Added new appendix - AAC LC Encoder Bit-rate and Sampling Frequency Combination <input type="checkbox"/> Added new appendix - AAC HE Encoder Bit-rate and Sampling Frequency Combination <input type="checkbox"/> Added Revision History <input type="checkbox"/> Modified XDM version to 1.0 <input type="checkbox"/> Modified DSP BIOS version to 5.31 <input type="checkbox"/> Changed all instances of IAUDENC to IAUDENC1
Related Documentation From Texas	Added Davinci technology related information
Section 2.1.1	Hardware: <ul style="list-style-type: none"> <input type="checkbox"/> Added list of supported platforms that this codec supports
Section 2.2	Installing the Component: <ul style="list-style-type: none"> <input type="checkbox"/> Modified Top-level directory name <input type="checkbox"/> Modified sub-directory name <input type="checkbox"/> Added evaluation version note
Table 2-1	Component Directories: <ul style="list-style-type: none"> <input type="checkbox"/> Modified description of \Docs
Section 2.4	Building and Running the Sample Test Application: <ul style="list-style-type: none"> <input type="checkbox"/> Added messages that the codec displays after successful encoding
Section 2.5.1	Generic Configuration File: <ul style="list-style-type: none"> <input type="checkbox"/> Modified format of the Testvecs.cfg file <input type="checkbox"/> Modified sample Testvecs.cfg file
Section 2.5.2	Encoder Configuration File: <ul style="list-style-type: none"> <input type="checkbox"/> Modified sample Testparams.cfg file
Section 2.6	Standards Conformance and User-Defined Inputs: <ul style="list-style-type: none"> <input type="checkbox"/> Added messages that the codec displays after successful encoding
Section 2.8	Evaluation Version: <ul style="list-style-type: none"> <input type="checkbox"/> Added evaluation version

Section	Additions/Modifications/Deletions
Section 4.1	<p data-bbox="542 268 1073 289">Symbolic Constants and Enumerated Data Types:</p> <p data-bbox="542 323 1390 375">Added Group or Enumeration Class <code>IAUDIO_ChannelMode</code> and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul data-bbox="542 380 756 653" style="list-style-type: none">❑ <code>IAUDIO_1_0</code>❑ <code>IAUDIO_2_0</code>❑ <code>IAUDIO_11_0</code>❑ <code>IAUDIO_3_0</code>❑ <code>IAUDIO_2_1</code>❑ <code>IAUDIO_3_1</code>❑ <code>IAUDIO_2_2</code>❑ <code>IAUDIO_3_2</code>❑ <code>IAUDIO_2_3</code>❑ <code>IAUDIO_3_3</code>❑ <code>IAUDIO_3_4</code> <p data-bbox="542 684 1060 705">Removed following Group or Enumeration Class:</p> <ul data-bbox="542 709 911 835" style="list-style-type: none">❑ <code>IAUDIO_ChannelId</code>❑ <code>AACENC_TRANSPORT_TYPE</code>❑ <code>AACENC_BITRATE_MODE</code>❑ <code>AACENC_BOOL_TYPE</code>❑ <code>AACENC_BOOL_TYPE</code> <p data-bbox="542 867 1403 919">Added Group or Enumeration Class <code>IAUDIO_DualMonoMode</code> and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul data-bbox="542 924 924 1020" style="list-style-type: none">❑ <code>IAUDIO_DUALMONO_LR</code>❑ <code>IAUDIO_DUALMONO_LEFT</code>❑ <code>IAUDIO_DUALMONO_RIGHT</code>❑ <code>IAUDIO_DUALMONO_LR_MIX</code> <p data-bbox="542 1052 1369 1104">Added Group or Enumeration Class <code>IAUDIO_EncMode</code> and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul data-bbox="542 1108 743 1150" style="list-style-type: none">❑ <code>IAUDIO_CBR</code>❑ <code>IAUDIO_VBR</code> <p data-bbox="542 1182 1369 1234">Added Group or Enumeration Class <code>XDM_DataFormat</code> and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul data-bbox="542 1239 727 1356" style="list-style-type: none">❑ <code>XDM_LE_32</code>❑ <code>XDM_LE_64</code>❑ <code>XDM_BE_16</code>❑ <code>XDM_BE_32</code>❑ <code>XDM_BE_64</code> <p data-bbox="542 1388 1390 1440">Added new symbolic constant <code>XDM_GETVERSION</code> under <code>XDM_CmdId</code> Group or Enumeration Class</p>

Section	Additions/Modifications/Deletions
Section 4.1	<p>Added Group or Enumeration Class XDM_AccessMode and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul style="list-style-type: none"> <input type="checkbox"/> XDM_ACCESSMODE_READ <input type="checkbox"/> XDM_ACCESSMODE_WRITE <p>Added Group or Enumeration Class AACENC_VBR_MODE and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul style="list-style-type: none"> <input type="checkbox"/> AACENC_BR_MODE_VBR_1 <input type="checkbox"/> AACENC_BR_MODE_VBR_2 <input type="checkbox"/> AACENC_BR_MODE_VBR_3 <input type="checkbox"/> AACENC_BR_MODE_VBR_4 <input type="checkbox"/> AACENC_BR_MODE_VBR_5 <p>Added Group or Enumeration Class AACENC_TRANSPORT_TYPE and the following are the Symbolic Constants added under this Enumeration Class:</p> <ul style="list-style-type: none"> <input type="checkbox"/> AACENC_TT_RAW <input type="checkbox"/> AACENC_TT_ADIF <input type="checkbox"/> AACENC_TT_ADTS <p>Added new symbolic constant XDM_PARAMSCHANGE under XDM_ErrorBit Group Enumeration Class</p>
Section 4.2.1	<p>Common XDM Data Structures:</p> <p>Added following new XDM data structures:</p> <ul style="list-style-type: none"> <input type="checkbox"/> XDM_SingleBufDesc <input type="checkbox"/> XDM1_SingleBufDesc <input type="checkbox"/> XDM1_BufDesc <input type="checkbox"/> IAUDENC1_Fxns <input type="checkbox"/> IAUDENC1_Params <input type="checkbox"/> IAUDENC1_DynamicParams <input type="checkbox"/> IAUDENC1_InArgs <input type="checkbox"/> IAUDENC1_Status <input type="checkbox"/> IAUDENC1_OutArgs <p>Removed following new XDM data structures:</p> <ul style="list-style-type: none"> <input type="checkbox"/> XDM_BufDesc <input type="checkbox"/> IAUDENC_Fxns <input type="checkbox"/> IAUDENC_Params <input type="checkbox"/> IAUDENC_DynamicParams <input type="checkbox"/> IAUDENC_InArgs <input type="checkbox"/> IAUDENC_Status <input type="checkbox"/> IAUDENC_OutArgs
Section 4.2.2.1	<p>Removed the following fields from IAACENC_Params Data Structure:</p> <ul style="list-style-type: none"> <input type="checkbox"/> useCRC <input type="checkbox"/> DownMixFlag <input type="checkbox"/> bitRate <input type="checkbox"/> sampleRate <input type="checkbox"/> nbInChannels <input type="checkbox"/> dualMono <input type="checkbox"/> ancFlag
Section 4.2.2.2	<p>Removed the following fields from IAACENC_DynamicParams Data Structure:</p> <ul style="list-style-type: none"> <input type="checkbox"/> dualMono <input type="checkbox"/> useCRC <input type="checkbox"/> DownMixFlag <input type="checkbox"/> outObjectType <input type="checkbox"/> outFileFormat

Section	Additions/Modifications/Deletions
Section 4.2.2.3	Removed the following fields from IAACENC_InArgs Data Structure: <ul style="list-style-type: none"><li data-bbox="542 296 773 317">❑ numInSamples<li data-bbox="542 321 711 342">❑ *ancData<li data-bbox="542 346 756 367">❑ numAncBytes
Section 4.2.2.4	Removed the following fields from IAACENC_Status AAC Encoder Data Structure: <ul style="list-style-type: none"><li data-bbox="542 457 834 478">❑ outputObjectType<li data-bbox="542 483 695 504">❑ bitRate<li data-bbox="542 508 695 529">❑ isValid
