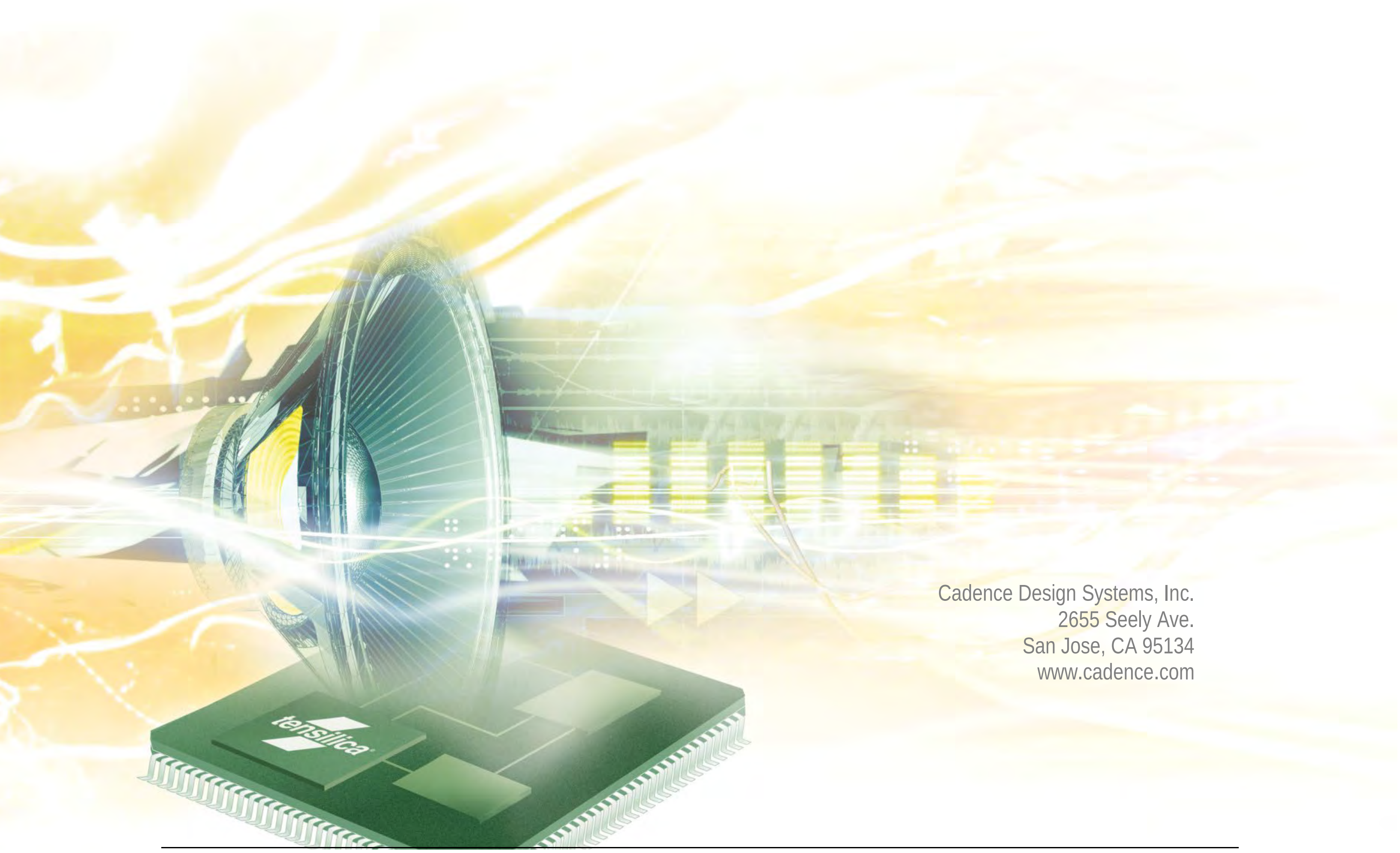
cādence®

Opus Codec

Programmer's Guide

For HiFi DSPs



Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

Version: 1.11
Last Updated: June 2023

Cadence Design Systems, Inc.
2655 Seely Ave.
San Jose, CA 95134
www.cadence.com

Contents

Figures

Tables

Document Change History

Version	Changes
1.0	**Initial revision**
1.1	**Updated the performance data in Section 1.4**
1.2	**Updated performance data in Section 1.4.** **Added two new error codes in Section 3.4.3:** ▪ XA_OPUS_CONFIG_FATAL_ENC_INVALID_PARAM_COMBINATION ▪ XA_OPUS_EXECUTE_NONFATAL_INVALID_FORCED_MODE_ SETTINGS
1.2.1	**Reference source code version updated to v1.1.3.** **Updated performance data in Section 1.4.** **Added control parameter 'variable_duration' for encoder in Section 3.3.**
1.3	**Added Ogg container support for Opus Decoder in Section 5.** **Added performance data for Ogg container support in Section 1.4.** **Added six new control parameters for decoder in Section 3.3.**
1.4	**Reference source code version updated to v1.2.1.** **Control paramter 'variable_duration' usage is deprecated. Updated Section 1.3, 3.3, 3.4.4, and 0 accordingly.** **Updated performance data in Section 1.4.** **Control parameter 'framesize' supports additional values. Updated Section 0 accordingly.**
1.5	**Added performance data for Fusion F1 DSP in Section 1.41.**

1.6	**Updated performance data, added new multistream decoder case.** **Added three new control parameters in decoder control structure to support multistream decoding in Section 3.3.** **Error code XA_OPUS_CONFIG_FATAL_ENC_INVALID_PARAM_COMBINATION is replaced by XA_OPUS_CONFIG_FATAL_INVALID_PARAM_COMBINATION for xa_opus_enc_init in Section 3.4.3.** **Error code XA_OPUS_CONFIG_FATAL_DEC_MULTISTREAM_DECODE_NOT_SUPPORTED is removed and XA_OPUS_CONFIG_FATAL_INVALID_PARAM_COMBINATION, XA_OPUS_CONFIG_FATAL_DEC_NUM_STREAMS_NOT_SUPPORTED, XA_OPUS_CONFIG_FATAL_DEC_UNKNOWN_CHANNEL_MAPPING, XA_OPUS_CONFIG_FATAL_DEC_BAD_STREAM_MAP, XA_OPUS_CONFIG_NONFATAL_DEC_STREAM_MAP_IGNORED added for xa_opus_dec_init in Section 3.4.3.** **Added new macro XA_OPUS_MAX_NUM_STREAMS and updated value of XA_OPUS_MAX_NUM_CHANNELS from 2 to 8 in Section 3.4.4.** **Updated decoder testbench help in Section 4.2, -numch range is now 0-8 and new option -strmap is added.**
1.7	**Added performance data for HiFi 5 in Section 1.4.**
1.8	**Reference source code version updated to v1.3.1.** **Updated performance data in Section 1.4.** **Now released in three packages: codec, decoder, and encoder.**
1.9	**Now released in single package that contains three libraries (codec, decoder, and encoder). Added instructions for switching library in Section 4.1.**
1.10	**Updated performance data for all cores and added new column for HiFi1 core in Section 1.4.** **Updated Section 4 to list testbench error handling source files .** **Section 4.2 updated to add details about constraint on sweep.**
1.11	**Updated build instructions in Section 4.1. Added support for standalone OPUS-SILK and OPUS-CELT libraries for encoder as well as decoder.**

		Added information about usage of OPUS-SILK and OPUS-CELT libraries in Sections 4.2.2 and 4.2.3. **Updated Memory requirements in Section 1.4.1. Added support for config based dynamic memory allocation.** **Added the following error codes in Section 3.4.4:** • XA_OPUS_EXECUTE_NONFATAL_UNSUPPORTED_MODE • XA_OPUS_EXECUTE_NONFATAL_HYBRID_MODE_DECODED

1. Introduction to the HiFi Opus Codec

The HiFi DSP Opus Codec implements the speech codec standard specified in the definition of the Opus audio codec (RFC 6716) [1]. The HiFi DSP Opus decoder also supports decoding of Opus streams in Ogg container format.

The vendor source version is libopus 1.3.1 [2].

1.1 Opus Codec Description

The Opus codec is designed to handle a wide range of interactive audio applications, including Voice over IP, video conferencing, in-game chat, and even live distributed music performances. It scales from low bitrate narrowband speech at 6 kbps to very high-quality stereo music at 510 kbps. Opus uses both Linear Prediction (LP) and the Modified Discrete Cosine Transform (MDCT) to achieve good compression of both speech and music.

1.2 Document Overview

This document covers all the information required to integrate the HiFi speech codecs into an application. The HiFi codec libraries implement a simple API to encapsulate the complexities of the coding operations and simplify the application and system implementation. Parts of the API are common to all the HiFi speech codecs and are described in Section 2. Section 3 covers all the features and information particular to the HiFi Opus Codec. Section 4 describes the example test bench. Section 5 describes the Ogg container format support interface of the HiFi Opus Codec. Section 6 provides references.

1.3 HiFi Opus Codec Specifications

The HiFi DSP Opus Codec implements the following features:

Cadence Speech Codec API is used

Encoder input/decoder output PCM: 16 bits per sample

Supported input sampling frequencies: 8 kHz, 12 kHz, 16 kHz, 24 kHz, 48 kHz

Support for following complexity modes: voip, audio, and restricted-low-delay

Supports five audio bandwidths:

- Narrow band (NB): 8 kHz effective sampling rate (4 kHz audio bandwidth)
- Medium band (MB): 12 kHz effective sampling rate (6 kHz audio bandwidth)
- Wideband (WB): 16 kHz effective sampling rate (8 kHz audio bandwidth)
- Super wideband (SWB): 24 kHz effective sampling rate (12 kHz audio bandwidth)
- Full-band (FB): 48 kHz effective sampling rate (20 kHz audio bandwidth)

Constant and variable bitrate encoding from 6 kbps to 510 kbps[1]

Discontinuous Transmission (DTX)

Voice Activity Detection (VAD)

Three operating modes of OPUS libraries:

- A SILK-only mode for use in low bitrate connections with an audio bandwidth of WB or less
- A Hybrid (SILK+CELT) mode for SWB or FB speech at medium bitrates
- A CELT-only mode for very low delay speech transmission as well as music transmission (NB to FB)

Supports Forward Error Correction (FEC)

Packet Loss Resilience

Frame duration: 2.5, 5, 10, 20, 40, 60, 80, 100 and 120 milliseconds(ms)

Supports the Ogg container format in decoder

Supports the standalone OPUS-SILK and OPUS-CELT encoder and decoder libraries

Stream/Config parameter based memory allocation

[1] RFC6716 Section 2.1.1

Table 1-1 shows the possible configurations of operating mode, audio bandwidth, and frame size.

Table 1-1 Opus Codec Configurations

Mode	Bandwidth	Frame sizes in ms
SILK-only	NB	10, 20, 40, 60, 80, 100, 120
SILK-only	MB	10, 20, 40, 60, 80, 100, 120
SILK-only	WB	10, 20, 40, 60, 80, 100, 120
Hybrid	SWB	10, 20
Hybrid	FB	10, 20
CELT-only	NB	2.5, 5, 10, 20, 40, 60, 80, 100, 120
CELT-only	WB	2.5, 5, 10, 20, 40, 60, 80, 100, 120
CELT-only	SWB	2.5, 5, 10, 20, 40, 60, 80, 100, 120
CELT-only	FB	2.5, 5, 10, 20, 40, 60, 80, 100, 120

The decoder implementation is verified to be bit-exact for all the test vectors available with the libopus 1.3.1 package. The encoder implementation is verified for internally generated test vectors and matches bit-exact with the reference implementation.

As some specific applications require only one of the SILK-only or CELT-only operating mode, standalone libraries for OPUS-SILK and OPUS-CELT encoder/decoder are provided. This reduces static and dynamic memory requirement by removing overheads of unused operating mode. For more information on the memory requirements of the standalone libraries, see Section 1.4.1.

Note: The OPUS-SILK and OPUS-CELT libraries follow Generic HiFi Speech Codec API and HiFi DSP Opus Codec API as described in Section 2 and Section 3. The encoder/decoder control parameters that are not supported in standalone libraries are mentioned in the description (see section 3.3).

For OPUS SILK/CELT standalone decoder binary, handling the Hybrid mode frames is challenging. The OPUS-SILK decoder is equipped with a feature to decode hybrid-mode frames with limited quality. The CELT mode is considered as invalid for OPUS-SILK standalone decoder. If the CELT mode is detected, such a frame is skipped, and decoding is resumed for next valid frame. Similarly, SILK mode and Hybrid mode frames are considered invalid for OPUS-CELT decoder and are skipped for decoding and a Non-Fatal error is returned. Warnings/Non-Fatal errors used for this feature are listed in Section 3.4.4 (Execution stage of decoder). This feature ensures that OPUS-SILK/OPUS-CELT decoder is compatible with any opus encoder and can handle opus-encoded frames using different types of modes.

Opus standard reference C code allocates memories assuming the worst-case scenario. This results in higher memory requirements for some scenarios. The Opus codec is enhanced with a feature to decide the dynamic memory requirement based on the stream and config parameters provided by the application. This feature reduces persistent/scratch/input/output memory footprint. It is achieved by calculating memory requirements based on config parameters like sample rate, number of channels, etc., instead of allocating memory based on the maximum possible value for the given config parameter.

1.4 HiFi Codec Performance

The HiFi DSP Opus codec is characterized on the 5-stage HiFi DSP. The memory usage and performance figures are provided for design reference.

Opus is more efficient when operating with variable bitrate (VBR), that is the default mode. Compared to the VBR mode, the constant bitrate (CBR) mode has lower audio quality at a given average bitrate and has higher computational complexity. For very low bitrates, the average CPU load (MHz) for the CBR mode is approximately 2.5 to 3 times that of the VBR mode.

1.4.1Memory

Library	Code (Kbytes)					Data (Kbytes)
	HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
xa_opus_codec.a	187.6	191.7	201.5	211.8	224.0	24.8
xa_opus_dec.a	81.6	82.2	88.0	92.1	100.9	22.4
xa_opus_enc.a	145	148.9	156.7	165.5	175.8	23.6
xa_opus_silk_codec.a	118.6	118.2	126.0	131.7	139.4	8.9
xa_opus_silk_enc.a	92.4	92.5	98.7	103.2	110.3	7.9
xa_opus_silk_dec.a	45.2	45.0	48.8	50.7	54.9	7.4
xa_opus_celt_codec.a	102.6	106.7	112.5	118.7	125.9	17.2
xa_opus_celt_enc.a	76.1	80.0	84.2	89.8	95.3	16.0
xa_opus_celt_dec.a	56.7	56.4	61.3	64.1	70.7	16.2

Encoder and Decoder

	Run Time Memory (Kbytes)				
	Persistent	Scratch	Stack	Input	Output
Opus Encoder (2 channels, 48KHz)	28.7	43.0	2.6	3.7	1.5
Opus Celt Encoder (2 channels, 48KHz)	11.7	39.0	2.6	3.7	1.5
Opus Encoder (1 Channel, 16KHz)	15.6	33.0	2.0	0.6	1.5
Opus Silk Encoder (1 Channel, 16KHz)	10.7	33.0	2.0	0.6	1.5
Opus Decoder Stereo (48KHz)	26.4	12.0	2.7	3.0	22.5
Opus Celt Decoder -Stereo (48KHz)	18	12.0	2.6	3.0	22.5
Opus Decoder (6 channels, 4 streams - 2 coupled, 2 uncoupled)	87.3	34.5	2.7	11.8	67.5
Opus Celt Decoder (6 channels, 4 streams - 2 coupled, 2 uncoupled)	53.9	34.5	2.6	11.8	67.5
Ogg parser (1 coupled streams)	140.6	0	0.6	4.0	3.0
Ogg parser (4 streams - 2 coupled, 2 uncoupled)	140.6	0	0.6	4.0	11.8

Note The Ogg parser runtime memory is only required for the Ogg Opus streams.

Note Ogg parser persistent memory requirement is dependent on maximum page size allowed and is configurable. The size reported above is for a default page size of 64 KB.

Note There is no restriction on input buffer size for Ogg Parser. If the input buffer is too small, it may require multiple calls to Ogg Parser to get an Ogg page. There is no restriction on output buffer size for Ogg Parser, but it should be large enough to hold at least one Opus packet. If the comment packet is bigger than the Opus packets, then the output buffer should be large enough to hold the comment packet.

1.4.2Timings

Encoder

Rat e kHz	Nch	Bitrate (kbps)	Average CPU Load (MHz)					
			HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
16	1	48	47.7	48.9	43.3	41.2	37.3	SILK VBR
16	1	48	65.9	68.0	60.0	57.3	52.0	SILK VBR FEC 10% loss
24	1	40	56.1	57.8	51.1	48.4	43.4	Hybrid VBR
24	1	40	72.9	75.3	66.4	63.2	57.1	Hybrid VBR FEC 10% loss
48	2	128	13.2	14.9	12.7	12.1	10.3	CELT VBR Complexity 0
48	2	128	25.1	27.9	24.1	23.3	19.4	CELT VBR Complexity 7
48	2	128	44.9	51.1	43.5	42.3	36.1	CELT VBR Complexity 10

Decoder

Rate kHz	Nch	Bitrate (kbps)	Average CPU Load (MHz)					
			HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
16	1	48	5.4	6.4	5.4	5.2	4.3	SILK (20 ms)
24	1	40	6.1	6.4	5.7	5.3	4.7	Hybrid (20 ms)
48	2	128	13.2	15.6	13.3	13.0	10.7	CELT (20 ms)
48	2	183.6	26.5	31.0	25.5	25.6	21.9	CELT (variable 2.5-20ms)
48	6	392	43.3	51.1	43.3	42.5	35.6	Multistream (20 ms, 4 streams - 2 coupled, 2 uncoupled))

Ogg Parser

Rate kHz	Nch	Bitrate (kbps)	Average CPU Load (MHz)					
			HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
48	2	183.6	0.4	0.6	0.4	0.4	0.4	CELT (variable 2.5-20ms)

Note Performance specification measurements are carried out on a cycle-accurate simulator assuming an ideal memory system, i.e., one with zero memory wait states. This is equivalent to running with all code and data in local memories or using an infinite-size, pre-warmed cache model.

Note Performance numbers of Opus encoder and decoder libraries are similar to the performance numbers of Opus-SILK and Opus-CELT encoder and decoder libraries for the corresponding test streams.

Note The MCPS numbers for HiFi 3z/HiFi 4 are obtained by running the test that is recompiled from the HiFi 3 optimized source code in the HiFi 3z/HiFi 4 configuration. No specific optimization is done for HiFi 3z/HiFi 4. Additional optimization is done for HiFi 1, and HiFi5.

Note All the memory and MCPS numbers are captured with XT-CLANG compiler and RI-2022.10 tools.

Note Encoder performance was measured with corresponding encoding mode set using command line option –force_mode.

Note Encoder performance for SILK and Hybrid vectors was measured for the highest complexity configuration parameter (complexity = 10). Encoder performance for all vectors was measured for frame size of 20 ms.

2. Generic HiFi Speech Codec API

This section describes the API, which is common to all the HiFi speech codec libraries. The API facilitates any codec that works in the overall method shown in Figure 1.

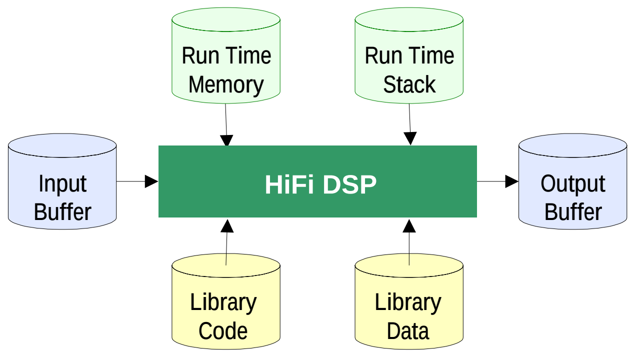


Figure 1 HiFi Speech Codec Interfaces

The following section discusses all the types of run time memory required by the codecs. There is no state information held in static memory, therefore a single thread can perform time division processing of multiple codecs. Additionally, multiple threads can perform concurrent codec processing.

2.1 Memory Management

The HiFi speech codec API supports a flexible memory scheme and a simple interface that eases the integration into the final application. The API allows the codecs to request the required memory for their operations during run time.

The run time memory requirement consists primarily of the scratch and persistent memory. The codecs also require an input buffer and output buffer for the passing of data into and out of the codec.

2.1.1 API Handle / Persistent Memory

The codec API stores its data in a structure that is passed via a handle that is a pointer to an opaque object from the application for each API call. All state information and the memory tables that the codec requires are referenced from this structure. This object also forms the static or context memory of the speech codecs. This is the state or history information that is maintained from one codec invocation to the next within the same thread or instance. The codecs expect that the contents of the persistent memory be unchanged by the system apart from the codec library itself for the complete lifetime of the codec operation.

2.1.2 Scratch Memory

This is the temporary buffer used by the codec for processing. The contents of this memory region should not be changed if the actual codec execution process is active, that is, if the thread running the codec is inside any API call. This region can be freely used by the system between successive calls to the codec.

2.1.3 Input Buffer

This is the buffer used by the algorithm for accepting input data. Before the call to the codec, the input buffer must be completely filled with input data.

2.1.4 Output Buffer

This is the buffer in which the algorithm writes the output. This buffer must be made available for the codec before its execution call. The output buffer pointer can be changed by the application between calls to the codec. This allows the codec to write directly to the required output area.

2.2 C Language API

Figure 2 shows an overview of the codec flow. The speech codec API consists of query, initialization, and execution functions. In the naming scheme below, `<codec>` is either the codec name (for example, `opus`), or the codec name with an `_enc` or `_dec` suffix for encoder- and decoder-specific functions, respectively.

2.2.1 Query Functions: xa_<codec>_get_<data>

The query functions are used in the startup and the memory allocation codec stages to obtain information about the version and the memory requirements of the codec library.

2.2.2 Initialization Functions: xa_<codec>_init

The initialization functions are used to reset the codec to its initial state. Because the codec library is fully reentrant, a process can initialize the codec library multiple times and multiple processes can initialize the same codec library as appropriate.

2.2.3Execution Functions: xa_<codec>

The execution functions are used to encode and decode speech frames.

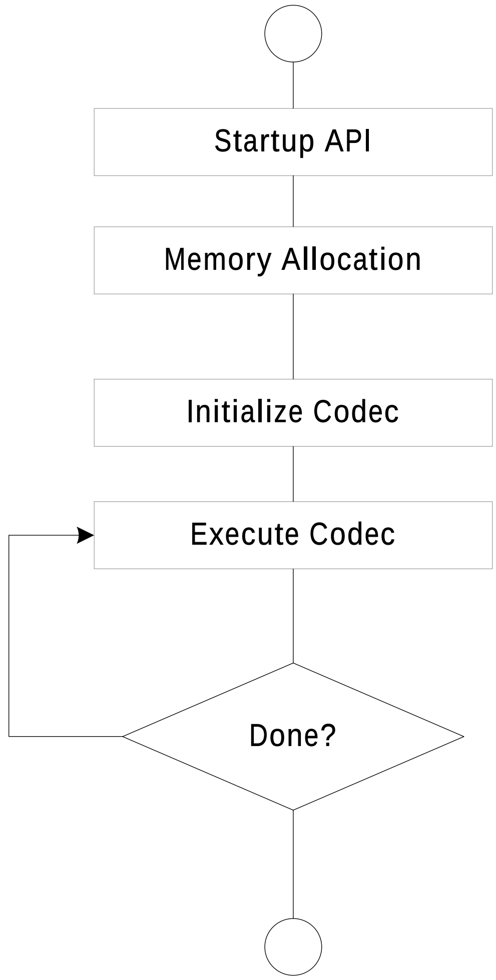


Figure 2 Speech Codec Flow Overview

2.3 Generic API Errors

Speech codec API functions return error code of type `XA_ERRORCODE`, that is of type `signed int`. The format of the error codes is defined in the following table.

31	30-15	14 – 11	10 - 6	5 - 0
Fatal	Reserved	Class	Codec	Sub code

The errors that can be returned from the API are subdivided into those that are fatal, which require the restarting of the entire codec, and those that are nonfatal and are provided for information to the application. The class of an error can be either API, Config, or Execution. The API category errors are concerned with the incorrect use of the API. The Config errors are produced when the codec parameters are incorrect or outside the supported usage. The Execution errors are returned after a call to the main encoding or decoding process and indicate situations that have arisen due to the input data.

2.4 Common API Errors

These errors are fatal and should not be encountered during normal application operation. They signal that a serious error has occurred in the application that is calling the codec.

XA_API_FATAL_MEM_ALLOC

At least one of the pointers passed into the API function is NULL

XA_API_FATAL_MEM_ALIGN

At least one of the pointers passed into the API function is not properly aligned

2.5 Files Describing the API

Following are the common include files (`include`):

`xa_error_standards.h`

The macros and definitions for all the generic errors

`xa_error_handler.h`

Definitions used by error handling functions

`xa_type_def.h`

All the types required for the API calls

3. HiFi DSP Opus Codec API

The HiFi DSP Opus Codec conforms to the generic speech codec API and the codec flow shown in Figure 2. The supported I/O formats, DTX modes, bit rates as well as the API functions, and the files specific to the Opus Codec are described in the following sections.

3.1 Files Specific to the Opus Codec

The Opus codec parameter header file (`include/opus_codec`):

`xa_opus_codec_api.h`

The Opus codec library (`lib`):

`xa_opus_codec.a`

`xa_opus_dec.a`

`xa_opus_enc.a`

3.2 I/O Formats

The input for the encoder is a block of 16-bit speech samples representing 2.5 ... 120 ms of speech frame data. The Opus encoder can have a maximum of six frames per packet. The encoder can accept input signals of sampling frequency from 8 kHz, 12 kHz, 16 kHz, 24 kHz, and 48 kHz.

The decoder processes the encoded bitstream frame-by-frame. Under normal conditions, one complete packet with additional 4 bytes at the beginning is required in the input buffer. These 4 bytes should contain 'range decoder state' variable available in 4 bytes before each packet in the bitstream encoded from conformant Opus encoder. In cases where these additional 4 bytes are not available or if the application wants to skip this check, it can be done through decoder control structure element 'no_range_dec_state' described in Table 3-2. In case of a lost packet, the decoder extracts the FEC data from the next packet and processes it. Thus, in case of packet loss, the application puts two packets in the input buffer for FEC to operate correctly. The decoder outputs 2.5 ... 120 ms frames composed of 16-bit PCM samples. The decoder can generate output signals of sampling frequency from 8 kHz, 12 kHz, 16 kHz, 24 kHz, and 48 kHz.

3.3 Control Structure for Opus Codec

The encoder control structure *xa_opus_enc_control_t* is used to communicate various parameters listed in Table 3-1 between codec library and the application.

Note All the parameters are of data type **WORD32**.

Table 3-1 Encoder Control Structure xa_opus_enc_control_t Parameters

Parameter	Input/ Output	Default value	Description
`application`	input		Application coding mode. Supported configurations: - XA_OPUS_APPLICATION_VOIP - XA_OPUS_APPLICATION_AUDIO - XA_OPUS_APPLICATION_RESTRICTED_LOWDELAY **Note:** This parameter is not configurable in OPUS-SILK library.
`API_sampleRate`	input		Input signal sampling rate in Hz. As described in Section 1.3.
`API_numChannels`	input		Number of channels in input signal; 1/2.
`bitRate`	input		Bitrate during active speech in bps. Recommended range is 6000 to 510000 as described in Section 1.3. All positive values and -1 (maximum possible bitrate based on other parameters) are accepted.
`cbr`	input	0	Flag to enable constant bitrate, 0/1.
`cvbr`	input	0	Flag to enable constrained variable bitrate, 0/1.
`bandwidth`	input	0	Audio bandwidth (from narrowband to fullband). Supported configurations: - XA_OPUS_AUTO (0) - XA_OPUS_BANDWIDTH_NARROWBAND - XA_OPUS_BANDWIDTH_MEDIUMBAND - XA_OPUS_BANDWIDTH_WIDEBAND - XA_OPUS_BANDWIDTH_SUPERWIDEBAND - XA_OPUS_BANDWIDTH_FULLBAND
`max_bandwidth`	input		Audio bandwidth (from narrowband to fullband). Supported configurations: - XA_OPUS_AUTO (0) - XA_OPUS_BANDWIDTH_NARROWBAND - XA_OPUS_BANDWIDTH_MEDIUMBAND - XA_OPUS_BANDWIDTH_WIDEBAND - XA_OPUS_BANDWIDTH_SUPERWIDEBAND - XA_OPUS_BANDWIDTH_FULLBAND

Parameter	Input/ Output	Default value	Description
`max_payload`	input	1500	Max payload size in bytes; [1, XA_OPUS_MAX_BYTES_PER_PACKET-4] range (values outside range ignored).
`complexity`	input	10	Complexity mode (0-10): 0 is lowest, 5 is medium, and 10 is highest complexity.
`SILK_InBandFEC_enabled`	input	0	Flag to enable SILK in-band Forward Error Correction (0/1). By default, it is disabled. **Note:** This parameter is not configurable in OPUS-CELT library.
`force_numChannels`	input	0	Force number of output channels (0/1/2). Default is forcing disabled (0).
`SILK_DTX_enabled`	input	0	Flag to enable discontinuous transmission (DTX); 0/1 Default DTX is disabled. 0–disable, 1–enable. **Note:** This parameter is not configurable in OPUS-CELT library.
`packetLossPercentage`	input	0	Uplink packet loss in percent (0-100).
`force_mode`	input	0	Force encode mode. Supported configurations: - XA_OPUS_AUTO (0) - XA_OPUS_MODE_SILK_ONLY - XA_OPUS_MODE_HYBRID - XA_OPUS_MODE_CELT_ONLY **Note:** This parameter is not configurable in OPUS-SILK and OPUS-CELT libraries.
`signal_type`	input	0	Configures the type of signal being encoded. Supported configurations: - XA_OPUS_AUTO (0) - XA_OPUS_SIGNAL_VOICE - XA_OPUS_SIGNAL_MUSIC **Note:** This parameter is not configurable in OPUS-SILK and OPUS-CELT libraries.
`reset_state`	input/ output	0	Resets the encoder state to be equivalent to a freshly initialized state, 0/1. 0 – no reset 1 – reset the encoder **Note:** Non-zero values are treated as 1.
`variable_duration`	Input	0	Deprecated. Not used by the library.

The decoder control structure *xa_opus_dec_control_t* communicates various parameters listed in Table 3-2 between the codec library and the application.

Note All the parameters are of data type **WORD32** except for stream_map which is an array of UWORD8.

Table 3-2 Decoder Control Structure xa_opus_dec_control_t Parameters

Parameter	Input/ Output	Default value	Description
`API_sampleRate`	input		Output signal sampling rate in Hz. As described in Section 1.3.
`API_numChannels`	input		Number of channels of output signal; 1/2 if channel_mapping is 0, 1-XA_OPUS_MAX_NUM_CHANNELS if channel_mapping is 1.
`SILK_InBandFEC_enabled`	input	0	Flag to enable SILK in-band Forward Error Correction (0/1). 0–disable, 1–enable. **Note:** This parameter is not configurable in OPUS-CELT library.
`gain`	input	0	Decoder gain adjustment, [-32768, 32767] range. gain = pow(10, x/(20.0*256))
`lostFlag`	input	0	Packet loss indicator, 0 - no loss, 1 – loss
`framesPerPacket`	output		Frames per packet 1, 2, 3, 4, 5, 6.
`moreInternalDecoderFrames`	output		Flag to indicate that the decoder has remaining payloads internally. This flag will be set to one if there are still some bytes to be decoded in the packet. It will be set to zero after decoding all the frames in the packet. Application shall load next packet only when this flag becomes zero.
`reset_state`	input/ output	0	This parameter is currently not used by decoder and will not affect decoding.
`no_range_dec_state`	input	0	'Range decoder state' conformance check variable control (Ref: RFC6716 - Section 6) 1: Excluded 'Range decoder state' variable in input payload 0: Included 'Range decoder state' variable in input payload
The following elements are required for Ogg container/multistream decode support and can be parsed from valid Opus header.			
`version`	input	0	Bitstream version from Opus Header. Only major version 0 streams are supported.
`nb_streams`	Input		Total number of streams, 1 – XA_OPUS_MAX_NUM_STREAMS, should be set to 1 for raw opus streams.

Parameter	Input/ Output	Default value	Description
nb_coupled	Input		Total number of coupled streams, 0 – nb_streams-1, For raw opus streams: should be set to 0 for mono decoding, 1 for stereo decoding
channel_mapping	Input	0	Channel mapping family; 0: RTP mapping[2], allowed number of channels 1 or 2. 1: Vorbis channel mapping[3], allowed number of channels 1 to 8 (Ref: RFC7845 - Section 5.1.1) Should be set to 0 for raw Opus streams.
stream_map[XA_OPUS_MAX_NUM_CHANNELS]	Input		Output channel map (ignored if channel_mapping is 0); Array of XA_OPUS_MAX_NUM_CHANNELS unsigned chars with range - 0 to (nb_streams+nb_coupled-1): route corresponding stream (for interpretation of channels refer to RFC7845- Section 5.1.1.2), 255: output silence (Ref: RFC7845 - Section 5.1.1)

[2] RFC7845 Section 5.1.1.1

[3] RFC7845 Section 5.1.1.2

3.4 API Functions

The following sections specify the Opus codec API functions relevant to each stage in the codec flow.

3.4.1Startup Stage

The API startup functions described in Table 3-3 get the various identifications strings from the codec library. They are for information only and their usage is optional. These functions do not take any input arguments and return `const char *`.

Table 3-3 Library Identification Functions

Function	Description
`xa_opus_get_lib_name_string`	Get the name of the library.
`xa_opus_get_lib_version_string`	Get the version of the library.
`xa_opus_get_lib_api_version_string`	Get the version of the API.

Example

```
const char *name = xa_opus_get_lib_name_string();
const char *ver = xa_opus_get_lib_version_string();
const char *apiver = xa_opus_get_lib_api_version_string();
```

Errors

None

3.4.2Memory Allocation Stage

During the memory allocation stage, the application must reserve the necessary memory for the Opus Encoder and Decoder API handles (persistent state) and scratch buffers. The required alignment of the handles and the scratch buffers is 8 bytes. The application can use the functions listed in Table 3-4 to query the codec library for the required size of each buffer.

While input and output frame buffers are required for the operation of the codec, they do not need to be reserved at this stage. Pointers to the frame buffers are passed in each invocation of the main codec execution function. The size and alignment requirements of the I/O buffers are specified in Section 3.4.4.

Table 3-4 Memory Management Functions

Function	Description
`xa_opus_enc_get_handle_byte_size`	Returns the size of the Opus Encoder API handle (persistent state) in bytes.
`xa_opus_dec_get_handle_byte_size`	Returns the size of the Opus Decoder API handle (persistent state) in bytes.
`xa_opus_enc_get_scratch_byte_size`	Returns the size of the Opus Encoder scratch buffer in bytes.
`xa_opus_dec_get_scratch_byte_size`	Returns the size of the Opus Decoder scratch buffer in bytes.

Example

```
WORD32 enc_handle_size, dec_handle_size;
WORD32 enc_scratch_size, dec_scratch_size;
enc_handle_size = xa_opus_enc_get_handle_byte_size(
                        WORD32 numChannels);
dec_handle_size = xa_opus_dec_get_handle_byte_size(
                        WORD32 nb_streams, WORD32 nb_coupled);
```

For Opus Celt library,

```
enc_scratch_size = xa_opus_enc_get_scratch_byte_size (
                       WORD32 numChannels,
                       WORD32 sampleRate,
                       WORD32 application);
```

For Opus Silk library and Opus library,

```
enc_scratch_size = xa_opus_enc_get_scratch_byte_size (
                       WORD32 numChannels,
                       WORD32 sampleRate);

dec_scratch_size = xa_opus_dec_get_scratch_byte_size(
                        WORD32 channel_mapping);
```

Errors

None

3.4.3Initialization Stage

In the initialization stage, the application points the Opus codec to its API handle. The application also specifies various other parameters related to the operation of the codec and places the codec in its initial state. The API functions for Opus encoder and decoder initialization are specified in Table 3-5 and Table 3-6, respectively.

Opus Encoder Initialization Function

Table 3-5 Opus Encoder Initialization Function

Function	`xa_opus_enc_init`

Syntax	`XA_ERRORCODE`
	`xa_opus_enc_init (xa_codec_handle_t handle,`
	`    xa_opus_enc_control_t *enc_control);`

Description	Resets the encoder API handle into its initial state. Sets up the encoder using supplied configuration. It also returns the current configuration of encoder in `enc_control` structure. You can modify this structure later before the execution call to control the behavior of encoder.
Parameters	Input: `handle` Pointer to the encoder handle (persistent state) Required size: see `xa_opus_enc_get_handle_byte_size` Required alignment: 8 bytes Input: `enc_control` Pointer to the encoder control structure. Refer to *xa_opus_enc_control_t* described in Section 3.3.

Example

```
handle =(pVOID) malloc(handle_size);
xa_opus_enc_control_t enc_control;

error_code = xa_opus_enc_init (handle,
                                &enc_control);
```


Errors

XA_API_FATAL_MEM_ALLOC

XA_API_FATAL_MEM_ALIGN

XA_OPUS_CONFIG_FATAL_SAMP_FREQ_NOT_SUPPORTED
Sampling frequency not supported.

XA_OPUS_CONFIG_FATAL_NUM_CHANNEL_NOT_SUPPORTED
Number of channels not supported.

XA_OPUS_CONFIG_FATAL_ENC_INVALID_APP_SETTING
Application mode not supported.

XA_OPUS_CONFIG_FATAL_ENC_INVALID_BITRATE_SETTING
Invalid bit rate.

XA_OPUS_CONFIG_FATAL_INVALID_PARAM_COMBINATION
Invalid configuration parameter combination. (Not used)

XA_OPUS_EXECUTE_FATAL_INIT_FAILURE
Error occurred in initializing the encoder.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_FORCE_CH_SETTING
Invalid encoder channel force setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_MAX_BANDWIDTH_SETTING
Invalid encoder maximum bandwidth setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_BANDWIDTH_SETTING
Invalid encoder bandwidth setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_COMPLEXITY_SETTING
Invalid encoder complexity setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_PACKET_LOSS_PERC_SETTING
Invalid encoder expected packet loss percentage setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_FORCE_MODE_SETTING
Invalid encoder force mode setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_SIGNAL_TYPE_SETTING
Invalid encoder signal type setting.

XA_OPUS_EXECUTE_NONFATAL_BAD_ARG
Invalid argument setting.

Opus Decoder Initialization Function

Table 3-6 Opus Decoder Initialization Function

Function	`xa_opus_dec_init`

Syntax	`XA_ERRORCODE`
	`xa_opus_dec_init (`
	`    xa_codec_handle_t handle,`
	`    xa_opus_dec_control_t *dec_control)`

Description	Resets the decoder API handle into its initial state. Sets up the decoder to run using the supplied configuration in the specified encoded speech format.
Parameters	Input: handle Pointer to the decoder handle (persistent state) Required size: see `xa_opus_dec_get_handle_byte_size` Required alignment: 8 bytes Input: `dec_control` Pointer to the decoder control structure. Refer to *xa_opus_dec_control_t* described in Section 3.3.

Example

```
dec_handle  = (pVOID)malloc(handle_size);
xa_opus_dec_control_t dec_control;
error_code = xa_opus_dec_init(dec_handle,
                              &dec_control);
```

Errors

XA_API_FATAL_MEM_ALLOC

XA_API_FATAL_MEM_ALIGN

XA_OPUS_EXECUTE_FATAL_INIT_FAILURE
Opus decoder init failure.

XA_OPUS_CONFIG_FATAL_SAMP_FREQ_NOT_SUPPORTED
Not valid setting of sampling frequency.

XA_OPUS_CONFIG_FATAL_NUM_CHANNEL_NOT_SUPPORTED
Number of channels not supported.

XA_OPUS_CONFIG_FATAL_INVALID_PARAM_COMBINATION

Invalid configuration parameter combination. Indicates that either channel_mapping and API_numChannels values are incompatible or nb_coupled is more than nb_streams.

XA_OPUS_CONFIG_FATAL_DEC_NUM_STREAMS_NOT_SUPPORTED

Number of streams is not supported by decoder.

XA_OPUS_CONFIG_FATAL_DEC_OPUS_BITSTREAM_VERSION_NOT_SUPPORTED

Bitstream version not supported (Only major version 0 is supported).

XA_OPUS_CONFIG_FATAL_DEC_UNKNOWN_CHANNEL_MAPPING

Unknown channel mapping family.

XA_OPUS_CONFIG_FATAL_DEC_BAD_STREAM_MAP

Stream map is not valid for given input stream.

XA_OPUS_CONFIG_NONFATAL_DEC_INVALID_GAIN_SETTING

Invalid decoder gain adjustment setting.

XA_OPUS_CONFIG_NONFATAL_DEC_STREAM_MAP_IGNORED

Output stream map ignored.

3.4.4 Execution Stage

The Opus codec processes the input stream and generates the output stream frame-by-frame. Each call to a codec execution function requires one complete frame as input and produces one complete packet as output.

If there is insufficient data in the input buffer, the Opus codec does not consume any bytes and returns a non-fatal error that allows the application to fill in additional data before calling the decoder again.

The Opus encoder takes as input one (2.5 ... 120) ms speech frame containing 16-bit samples. The Opus decoder also process data of a (2.5 ... 120) ms frame. It has a buffer mechanism for extracting FEC data from the next packet. It holds data for the next two packets in the input buffer.

The following macros are used to calculate I/O buffer sizes and configuration of Opus.

Defines with Values	Description
`XA_OPUS_MAX_NUM_STREAMS 8`	Maximum number of streams supported by decoder.
`XA_OPUS_MAX_NUM_CHANNELS 8`	Maximum number of channels supported by decoder, encoder support only up to 2 channels.
`XA_OPUS_MAX_FRAMES_PER_PACKET 6`	Maximum 6 20 ms frames (total 120 ms) per packet.
`XA_OPUS_MAX_SAMPLES_PER_FRAME_CELT 960`	Maximum number of samples in a frame in CELT mode.

`XA_OPUS_MAX_BYTES_CHANNEL_PACKET`
 	
`XA_OPUS_MAX_SAMPLES_PER_FRAME_CELT * sizeof(WORD16) *`
 	
`XA_OPUS_MAX_FRAMES_PER_PACKET`
	Maximum number of bytes in one channel of a packet.

`XA_OPUS_MAX_FEC_DELAY 1`	Latency in packets in case of data lost.

`XA_OPUS_MAX_PACKET_IN_INP_BUF`
 	
`XA_OPUS_MAX_FEC_DELAY + 1`
	Number of packets in input buffer in case of packets lost.

`XA_OPUS_MAX_BYTES_ENCODED_PACKET 1504`	Maximum number of bytes in one encoded packet / maximum number of bytes in input buffer if no packets lost.

`XA_OPUS_MAX_DEC_INP_BYTES`
 	
`XA_OPUS_MAX_BYTES_ENCODED_PACKET *`
 	
`XA_OPUS_MAX_PACKET_IN_INP_BUF`
	Max. number of data bytes in input buffer in case of packet loss.

Defines with Values	Description
`XA_OPUS_APPLICATION_VOIP 2048`	Setting for most VoIP/videoconference applications, where the listening quality and intelligibility matter the most.
`XA_OPUS_APPLICATION_AUDIO 2049`	Setting for broadcast/high-fidelity application where the decoded audio should be as close as possible to the input.
`XA_OPUS_APPLICATION_RESTRICTED_ LOWDELAY 2051`	Setting only used when lowest-achievable latency is what matters most. Voice-optimized modes cannot be used.
`XA_OPUS_AUTO 0`	Value of auto/default setting
`XA_OPUS_BANDWIDTH_NARROWBAND 1101`	4 kHz bandwidth
`XA_OPUS_BANDWIDTH_MEDIUMBAND 1102`	6 kHz bandwidth
`XA_OPUS_BANDWIDTH_WIDEBAND 1103`	8 kHz bandwidth
`XA_OPUS_BANDWIDTH_SUPERWIDEBAND 1104`	12 kHz bandwidth
`XA_OPUS_BANDWIDTH_FULLBAND 1105`	20 kHz bandwidth
`XA_OPUS_MODE_SILK_ONLY 1000`	SILK mode
`XA_OPUS_MODE_HYBRID 1001`	Hybrid mode
`XA_OPUS_MODE_CELT_ONLY 1002`	CELT mode
`XA_OPUS_SIGNAL_VOICE 3001`	Signal being encoded is voice.
`XA_OPUS_SIGNAL_MUSIC 3002`	Signal being encoded is music.

The syntax of the Opus encoder and decoder execution functions is specified in Table 3-7 and Table 3-8, respectively.

Opus Encoder Execution Function

Table 3-7 Opus Encoder Execution Function

Function	`xa_opus_enc`

Syntax	`XA_ERRORCODE`
	`xa_opus_enc (`
	`xa_codec_handle_t handle,`
	`pWORD16 inp_speech,`
	`pUWORD8 enc_speech,`
	`WORD16 inp_samples,`
	`xa_opus_enc_control_t *enc_control,`
	`WORD16 *out_bytes,`
	`pVOID scratch);`

Description	Encodes one frame of data.
Parameters	Input: `handle` Pointer to the encoder handle (persistent state) 8 bytes alignment is required.

	Input: `inp_speech` Pointer to the input speech pcm samples. There must be at least one frame of data in the input. 8 bytes alignment is required. Output: `enc_speech` Pointer to the encoded speech parameters (Opus core frame). 1-byte alignment is required. Input: `inp_samples` Number of samples in the input speech buffer. Input/Output: `enc_control` Pointer to the encoder control structure. Refer to *xa_opus_enc_control_t* described in section 3.3. Output: `out_bytes` Number of output bytes generated by the encoder. Input/Output: `scratch` Pointer to the scratch memory. 8 bytes alignment is required.
Note	None

Example

```
WORD16 inp_samples, out_bytes;
pWORD16 inp_speech;
pUWORD8 enc_speech;
xa_opus_enc_control_t enc_control;
pVOID scratch;

error_code = xa_opus_enc(
                xa_codec_handle_t handle,
                inp_speech,
                enc_speech,
                inp_samples,
                &enc_control,
                &out_bytes,
                scratch);
```

Errors

XA_API_FATAL_MEM_ALLOC

XA_API_FATAL_MEM_ALIGN

XA_OPUS_EXECUTE_FATAL_NOT_INITIALIZED

Opus encoder is not yet initialized.

XA_OPUS_EXECUTE_FATAL_PARAMS_CHANGED
Opus parameters changed. Opus must be reinitialized.

XA_OPUS_EXECUTE_FATAL_ENC_INVALID_NUM_INP_SAMPLES
Invalid (less than or equal to zero) number of input samples.

XA_OPUS_CONFIG_FATAL_ENC_INVALID_APP_SETTING
Application mode not supported.

XA_OPUS_CONFIG_FATAL_ENC_INVALID_BITRATE_SETTING
Invalid bit rate.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_FORCE_CH_SETTING
Invalid encoder channel force setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_MAX_BANDWIDTH_SETTING
Invalid encoder max. bandwidth setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_BANDWIDTH_SETTING
Invalid encoder bandwidth setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_COMPLEXITY_SETTING
Invalid encoder complexity setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_PACKET_LOSS_PERC_SETTING
Invalid encoder expected packet loss percentage setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_FORCE_MODE_SETTING
Invalid encoder force mode setting.

XA_OPUS_CONFIG_NONFATAL_ENC_INVALID_SIGNAL_TYPE_SETTING
Invalid encoder signal type setting.

XA_OPUS_EXECUTE_NONFATAL_INVALID_FORCED_MODE_SETTINGS
Invalid force mode setting used.

XA_OPUS_EXECUTE_NONFATAL_BAD_ARG
Invalid argument setting.

XA_OPUS_EXECUTE_FATAL_ENC_OUT_BUF_TOO_SHORT
Encoder output buffer too short.

XA_OPUS_EXECUTE_NONFATAL_INTERNAL_ERROR
Internal error.

Opus Decoder Execution Function

Table 3-8 Opus Decoder Execution Function

Function	`xa_opus_dec`

Syntax	`XA_ERRORCODE`
	`xa_opus_dec (`
	`    xa_codec_handle_t handle,`
	`    pUWORD8 enc_speech,`
	`    pWORD16 synth_speech,`
	`    const WORD32  num_bytes_in,`
	`    xa_opus_dec_control_t *dec_control,`
	`    WORD16  *num_samples_out,`
	`    pVOID scratch);`

Description	Decodes one frame of data
Parameters	Input: `handle` Pointer to the decoder handle (persistent state) 8 bytes alignment is required. Input: `enc_speech` Pointer to the encoded speech parameters (Opus core frame). 1 byte alignment is required. Output: `synth_speech` Pointer to the synthesized speech containing 16-bit pcm samples. 4 bytes alignment is required. Input: `num_bytes_in` Number of bytes in the input buffer. Input: `dec_control` Pointer to the decoder control structure. Refer to *xa_opus_dec_control_t* described in Section 3.3. Output: `num_samples_out` Number of output samples generated by the decoder. Input/Output: `scratch` Pointer to the scratch memory. 8 bytes alignment is required.
Note	None

Example

```
WORD16 num_samples_out;
```

```
WORD32 num_bytes_in;
pUWORD8 enc_speech;
pWORD16 synth_speech;
xa_opus_dec_control_t dec_control;
pVOID scratch;

error_code = xa_opus_dec (
                xa_codec_handle_t handle,
                enc_speech,
                synth_speech,
                num_bytes_in,
                &dec_control,
                &num_samples_out,
                scratch);
```

Errors

XA_API_FATAL_MEM_ALLOC

XA_API_FATAL_MEM_ALIGN

XA_OPUS_EXECUTE_FATAL_NOT_INITIALIZED
Opus decoder is not yet initialized.

XA_OPUS_EXECUTE_FATAL_PARAMS_CHANGED
Opus parameters changed. Opus must be reinitialized.

XA_OPUS_CONFIG_NONFATAL_DEC_INVALID_GAIN_SETTING
Invalid decoder gain adjustment setting.

XA_OPUS_EXECUTE_FATAL_DEC_PAYLOAD_ERROR
The compressed data passed is corrupted.

XA_OPUS_EXECUTE_NONFATAL_BAD_ARG
Invalid arguments setting.

XA_OPUS_EXECUTE_NONFATAL_INSUFFICIENT_DATA
Insufficient data.

XA_OPUS_EXECUTE_NONFATAL_INTERNAL_ERROR
Internal error.

XA_OPUS_EXECUTE_NONFATAL_INVALID_PACKET
Invalid packet.

XA_OPUS_EXECUTE_NONFATAL_UNSUPPORTED_MODE
Invalid Mode detected by standalone SILK/CELT mode library.

XA_OPUS_EXECUTE_NONFATAL_HYBRID_MODE_DECODED

Hybrid Mode frame detected for standalone SILK mode library, Decoded frame with limited quality.

4. Introduction to the Example Test Bench

The Opus Codec library is provided with two sample test bench applications: one for the encoder and one for the decoder. The supplied test benches consist of the following files:

Test bench source files (test/src)

- `xa_opus_decoder_sample_testbench.c`
- `xa_opus_encoder_sample_testbench.c`
- `xa_opus_codec_error_handler.c`
- `opus_header.c`
- `xa_common_error_handler.c`
- `xa_ogg_lib_error_handler.c`

Makefile to build the executables (`test/build`)

- `makefile_testbench_sample`

4.1 Making Executable

This section details how to build the testbench and the required libraries. You can build both the encoder and decoder testbench or a specific encoder or decoder testbench. Similarly, you can either build both libraries or a particular library.

Building OPUS Applications

Building Testbench Applications

- **Build Encoder and Decoder Testbench for Opus, Opus-SILK, and Opus-CELT**
  1. In the command prompt, navigate to the `test/build` directory.
  2. Enter the following command:
     `xt-make –f makefile_testbench_sample clean all`

The Opus encoder (`xa_opus_enc_test`) and the Opus decoder (`xa_opus_dec_test`) example test bench is built using the `xa_opus_codec.a` library.

The Opus-SILK encoder (`xa_opus_silk_enc_test`) and the Opus-SILK decoder (`xa_opus_silk_dec_test`) example test bench is built using the `xa_opus_silk_codec.a` library.

The Opus-CELT encoder (`xa_opus_celt_enc_test`) and the Opus-CELT decoder (`xa_opus_celt_dec_test`) example test bench is built using the `xa_opus_celt_codec.a` library.

- **Build Either an Encoder or a Decoder Testbench for Opus, Opus-SILK, and Opus-CELT**

  1. In the command prompt, navigate to the `test/build` directory.
  2. Enter the following command:
     ```
     xt-make -f makefile_testbench_sample clean [opus_dec|opus_silk_dec|opus_celt_dec|opus_enc|opus_silk_enc|opus_celt_enc]
     ```

The encoder example test bench `xa_opus_enc_test/xa_opus_silk_enc_test/xa_opus_celt_enc_test` or the decoder example test bench `xa_opus_dec_test/xa_opus_silk_dec_test/xa_opus_celt_dec_test` is built.

Building Libraries

- **Build Libraries for Opus, Opus-SILK, and Opus-CELT**

If you have source code distribution for the codec package, you must build the Opus codec libraries before you build the test bench.

  1. In the command prompt, navigate to the `build` directory.
  2. Enter the following command:
     ```
     xt-make clean all install
     ```

The Opus codec libraries, namely xa_opus_codec.a, xa_opus_enc.a, xa_opus_dec.a, xa_opus_silk_codec.a, xa_opus_silk_enc.a, xa_opus_silk_dec.a, xa_opus_celt_codec.a, xa_opus_celt_enc.a, and xa_opus_celt_dec.a are built and copied to the `lib` directory..

- **Build Either an Encoder or a Decoder Library for Opus, Opus-SILK, Opus-CELT**

  1. In the command prompt, navigate to the `build` directory.
  2. Enter the following command:
     ```
     xt-make clean [opus_dec install_dec|opus_silk_dec install_silk_dec|opus_celt_dec install_celt_dec|opus_enc install_enc|opus_silk_enc install_silk_enc|opus_celt_enc install_celt_enc]
     ```

The encoder `xa_opus_enc.a`, `xa_opus_silk_enc.a`, and `xa_opus_celt_enc.a` or the decoder `xa_opus_dec.a`, `xa_opus_silk_dec.a`, and `xa_opus_celt_dec.a` libraries are built and copied to the `lib` directory.

XWS Package

Note	For information on building and executing the application from a Windows-based (.xws) based release package, refer to the readme.html file available in the imported application project.

In the XWS package, by default testxa_opus_dec decoder testbench project depends on the following:

- ulibxa_opus_codec library project in object .xws package.
- libxa_opus_codec library project in source .xws package.

Switch Between Projects

To switch to the decoder ulibxa_opus_dec, perform the following steps in Xtensa Xplorer:

1. In the **Project Explorer** area:
    a. Click the triangle next to ulibxa_opus_dec to expand the folder.
    b. Right-click ulibxa_opus_dec/lib/xa_opus_dec.a and select **Unmanaged Binary Info**.
    c. Select the appropriate config (for example, AE_HiFi3_LE5) and click **OK**.

2. Right-click testxa_opus_dec and select **Library Dependencies…**.
   ulibxa_opus_dec/lib/xa_opus_dec.a (AE_HiFi3_LE5) is shown in the Available libraries area. Select an item and click **Add**. The item is shown in the Selected libraries area.

3. Select ulibxa_opus_codec/lib/xa_opus_codec.a (<none>) and click **Remove**, then click **Apply** and **OK**.

To switch to the decoder libxa_opus_dec, perform the following steps in Xtensa Xplorer:

1. In the **Project Explorer** area, right-click testxa_opus_dec and select **Library Dependencies…**.
2. In the **Available libraries** area, select libxa_opus_dec and click **Add**.
   The item is shown in the Selected libraries area.
3. In the **Selected libraries** area, select libxa_opus_codec and click **Remove**, then click **Apply** and **OK.**

Similarly, the above steps can be used to switch to the encoder library projects ulibxa_opus_enc or libxa_opus_enc by replacing “dec” with “enc”.

The above steps are for Opus, similarly, they can be used for Opus-SILK/Opus-CELT by replacing opus with opus_silk/opus_celt.

4.2 Usage

4.2.1 OPUS Usage

The sample application executable can be run with direct command-line options or with a parameter file.

Encoder

The sample application encoder executable can be run from the command line as follows:

```
xt-run xa_opus_enc_test –app:<application> -fs:<sampling rate> -
numch:<number of channels> -bps:<bit rate> [options] –ifile:<infile> -
ofile:<outfile>
```

Where:

`<application>:`	voip | audio | restricted-lowdelay
`<sampling rate>:`	8000 | 12000 | 16000 | 24000 | 48000;
`<number of channels>:`	1 | 2;
`<bit rate>:`	Recommended target bit rate<6000..510000> as described in Section 1.3; positive values outside the recommended range and -1 (maximum possible bitrate based on other parameters) are also accepted (this behavior is consistent with reference code **[2]**).
`<infile>:`	input speech file;
`<outfile>:`	output encoded file;
`[options]:`	
`-force_mode:`	<hybrid|silk|celt> forces the encoding mode; default: auto
`-cbr:`	<0/1> enable constant bitrate; default: variable bitrate;
`-cvbr:`	<0/1> enable constrained variable bitrate; default: unconstrained;
`-bandwidth:`	<NB|MB|WB|SWB|FB> audio bandwidth (from narrowband to fullband); default: depends on sampling rate;
`-framesize:`	<2.5|5|10|20|40|60|80|100|120> frame size in ms; default: 20;
`-max_payload:`	<bytes> maximum payload size in bytes, (1-XA_OPUS_MAX_BYTES_ENCODED_PACKET-4); default: 1500;(Ignored if out of range);
`-complexity:`	<comp> complexity, 0 (lowest) ... 10 (highest); default: 10;
`-inbandfec:`	<0/1> enable SILK inband FEC, default: disable;

`-forcemono:`	<0/1> force mono encoding, even for stereo input, default: disable;
`-dtx:`	<0/1> enable SILK DTX, default:disable;
`-loss:`	<perc> uplink loss estimate, in percent (0-100); default: 0;
`-sweep:`	<bps> sweep bitrate mode, bps – bitrate step, default: 0, if non-zero, used as a step-size to change the bitrate, recommended bitrate range<6000..510000>;
`-sweep_max:`	<bps> maximum bitrate in sweep bitrate mode, default: 0;
`-random_framesize:`	<0/1> enable random frame size, default: disable;
`-random_fec:`	<0/1> enable random SILK inband FEC, default: disable;
`-silk8k_test:`	<0/1> enable dynamic changing of frame size and number of channels in SILK mode (8 kHz) according to predefined list, default: disable;
`-silk12k_test:`	<0/1> enable dynamic changing of frame size and number of channels in SILK mode (12 kHz) according to predefined list, default: disable
`-silk16k_test:`	<0/1> enable dynamic changing of frame size and number of channels in SILK mode (16 kHz) according to predefined list, default: disable;
`-hybrid24k_test:`	<0/1> enable dynamic changing of frame size and number of channels in Hybrid mode (24 kHz) according to predefined list, default: disable;
`-hybrid48k_test:`	<0/1> enable dynamic changing of frame size and number of channels in Hybrid mode (48 kHz) according to predefined list, default: disable;
`-celt_test:`	<0/1> enable dynamic changing of frame size, number of channels and bandwidth in CELT mode according to predefined list, default: disable;
`-celt_hq_test:`	<0/1> enable dynamic changing of frame size and number of channels in CELT mode (FB only) according to predefined list, default: disable.

Decoder

The sample application decoder executable can be run from the command line as follows:

```
xt-run xa_opus_dec_test [options] –ifile:<infile>   -ofile:<outfile>
```

Where:

`<infile>:`	input encoded speech file;
`<outfile>:`	output file with decoded samples;
`[options]:`	
`-fs:`	8000 | 12000 | 16000 | 24000 | 48000; default: 48000 (for raw opus streams)
`-numch:`	1-8; default: 2 (for raw opus streams)
`-inbandfec:`	<0/1 enable SILK inband FEC (non-zero value is treated as 1), default: disable;
`-loss:`	<perc> simulate packet loss, in percent (0-100); default: 0;
`-btype:`	<ogg/raw> bitstream type, ogg or raw; default: raw;
`-maxpage:n`	Max Ogg page size: 'n' in kBytes, range: [1, 1024], default: 64;
`-strmap:<map>`	Output channel map, string of max. 8 characters, range for each character 0-8. 0-7 to route the corresponding channel from decoder output, 8 for silence, for streams with less than 8 channels (n) range becomes 0-(n-1), default: taken from opus header. For interpretation of channels please refer to RFC7845 Section 5.1.1.2.

Note Default values for options `-fs` and `-numch` are applicable only for raw opus streams. For Ogg opus streams, `-fs` and `-numch` are taken from ogg stream header without any check if not passed on command line.

Note If value for `-strmap` is specified partially, remaining channels are silenced, if it is longer than number of channels then extra values are ignored. This option is ignored for raw opus streams and channel mapping family 0 ogg opus streams.

If no command line arguments are given, the encoder or decoder application reads the commands from the parameter file `paramfilesimple_encode.txt` or `paramfilesimple_decode.txt`, respectively.

Following is the syntax for writing the `paramfilesimple` file:

```
@Start

@Input_path <path to be appended to all input files>

@Output_path <path to be appended to all output files>
<command line 1>
<command line 2>
....
@Stop
```

The Opus encoder and decoder can be run for multiple test files using the different command lines. The syntax for command lines in the parameter file is the same as the syntax for specifying options on the command line to the test bench program.

Note All the `@<command>`s should be at the first column of a line except the `@Newline` command.

Note All the `@<command>`s are case sensitive. If the command line in the parameter file has to be separated to two parts on two different lines, use the @Newline command.

For example,

```
<command line part 1> @ Newline
<command line part 2>
```

Note Blank lines will be ignored.

Note Individual lines can be commented out using "//" at the beginning of the line.

4.2.2 OPUS-SILK Usage

The sample application executable can be run with direct command-line options or with a parameter file.

Encoder

The sample application encoder executable can be run from the command line as follows:

```
xt-run  xa_opus_silk_enc_test  -fs:<sampling  rate>  -numch:<channels>  -
bps:<bit rate>  [options] -ifile:<input file> -ofile:<output file>
```

Where:

`<sampling rate>:`	8000 | 12000 | 16000 | 24000 | 48000;
`<number of channels>:`	1 | 2;
`<bit rate>:`	Recommended target bit rate<6000..510000> as described in Section 1.3; positive values outside the recommended range and -1 (maximum possible bitrate based on other parameters) are also accepted (this behavior is consistent with reference code **[2]**).
`<infile>:`	input speech file;
`<outfile>:`	output encoded file;
`[options]:`	
`-cbr:`	<0/1> enable constant bitrate; default: variable bitrate;
`-cvbr:`	<0/1> enable constrained variable bitrate; default: unconstrained;
`-bandwidth:`	<NB|MB|WB>audio bandwidth (from narrowband to wideband); default: sampling rate;
`-framesize:`	<10|20|40|60|80|100|120> frame size in ms; default: 20;
`-max_payload:`	<bytes> maximum payload size in bytes, (1-XA_OPUS_MAX_BYTES_ENCODED_PACKET-4); default: 1500;(Ignored if out of range);
`-complexity:`	<comp> complexity, 0 (lowest) ... 10 (highest); default: 10;
`-inbandfec:`	<0/1> enable SILK inband FEC, default: disable;
`-forcemono:`	<0/1> force mono encoding, even for stereo input, default: disable;

`-dtx:`	<0/1> enable SILK DTX, default:disable
`-loss:`	<perc> uplink loss estimate, in percent (0-100); default: 0;
`-sweep:`	<bps> sweep bitrate mode, bps – bitrate step, default: 0, if non-zero, used as a step-size to change the bitrate, recommended bitrate range<6000..510000>;
`-sweep_max:`	<bps> maximum bitrate in sweep bitrate mode, default: 0;
`-random_framesize:`	<0/1> enable random frame size, default: disable;
`-random_fec:`	<0/1> enable random SILK inband FEC, default: disable;
`-silk8k_test:`	<0/1> enable dynamic changing of frame size and number of channels in SILK mode (8 kHz) according to predefined list, default: disable;
`-silk12k_test:`	<0/1> enable dynamic changing of frame size and number of channels in SILK mode (12 kHz) according to predefined list, default: disable
`-silk16k_test:`	<0/1> enable dynamic changing of frame size and number of channels in SILK mode (16 kHz) according to predefined list, default: disable;

Decoder

The sample application decoder executable can be run from the command line as follows:

```
xt-run   xa_opus_silk_dec_test   -fs:<sampling  rate>   -numch:<channels>
[options] -ifile:<input_file> -ofile:<output_file>
```

Where:

`<input_file>:`	input encoded speech file;
`<output_file>:`	output file with decoded samples;
`[options]:`	
`-fs:`	8000 | 12000 | 16000 | 24000 | 48000; default: 48000 (for raw opus streams)
`-numch:`	1-8; default: 2 (for raw opus streams)
`-inbandfec:`	<0/1 enable SILK inband FEC (non-zero value is treated as 1), default: disable;
`-loss:`	<perc> simulate packet loss, in percent (0-100); default: 0;
`-btype:`	<ogg/raw> bitstream type, ogg or raw; default: raw;
`-maxpage:n`	Max Ogg page size: 'n' in kBytes, range: [1, 1024], default: 64;
`-strmap:<map>`	Output channel map, string of max. 8 characters, range for each character 0-8. 0-7 to route the corresponding channel from decoder output, 8 for silence, for streams with less than 8 channels (n) range becomes 0-(n-1), default: taken from opus header. For interpretation of channels please refer to RFC7845 Section 5.1.1.2.

Note Default values for options `-fs` and `-numch` are applicable only for raw opus streams. For Ogg opus streams, `-fs` and `-numch` are taken from ogg stream header without any check if not passed on command line.

Note If value for `-strmap` is specified partially, remaining channels are silenced, if it is longer than number of channels then extra values are ignored. This option is ignored for raw opus streams and channel mapping family 0 ogg opus streams.

If no command line arguments are given, the encoder or decoder application reads the commands from the parameter file `paramfilesimple_encode_silk.txt` or `paramfilesimple_decode_silk.txt`, respectively.

4.2.3 OPUS-CELT Usage

The sample application executable can be run with direct command-line options or with a parameter file.

Encoder

The sample application encoder executable can be run from the command line as follows:

```
xt-run xa_opus_celt_enc_test -app:<application> -fs:<sampling rate> -numch:<channels> -bps:<bit rate> [options] -ifile:<input file> -ofile:<output file>
```

Where:

`<application>:`	`audio | restricted-lowdelay`
`<sampling rate>:`	8000 | 12000 | 16000 | 24000 | 48000;
`<number of channels>:`	1 | 2;
`<bit rate>:`	Recommended target bit rate<6000..510000> as described in Section 1.3; positive values outside the recommended range and -1 (maximum possible bitrate based on other parameters) are also accepted (this behavior is consistent with reference code **[2]**).
`<input file>:`	input speech file;
`<output file>:`	output encoded file;
`[options]:`	
`-cbr:`	<0/1> enable constant bitrate; default: variable bitrate;
`-cvbr:`	<0/1> enable constrained variable bitrate; default: unconstrained;
`-bandwidth:`	< NB|MB|WB|SWB|FB > audio bandwidth (from narrowband to fullband); default: sampling rate;
`-framesize:`	< 2.5|5|10|20|40|60|80|100|120> frame size in ms; default: 20;
`-max_payload:`	<bytes> maximum payload size in bytes, (1-XA_OPUS_MAX_BYTES_ENCODED_PACKET-4); default: 1500;(Ignored if out of range);
`-complexity:`	<comp> complexity, 0 (lowest) ... 10 (highest); default: 10;

`-forcemono:`	<0/1> force mono encoding, even for stereo input, default: disable;
`-loss:`	<perc> uplink loss estimate, in percent (0-100); default: 0;
`-sweep:`	<bps> sweep bitrate mode, bps – bitrate step, default: 0, if non-zero, used as a step-size to change the bitrate, recommended bitrate range<6000..510000>;
`-sweep_max:`	<bps> maximum bitrate in sweep bitrate mode, default: 0;
`-random_framesize:`	<0/1> enable random frame size, default: disable;
`-celt_test:`	<0/1> enable dynamic changing of frame size, number of channels and bandwidth in CELT mode according to predefined list, default: disable;
`-celt_hq_test:`	<0/1> enable dynamic changing of frame size and number of channels in CELT mode (FB only) according to predefined list, default: disable.

Decoder

The sample application decoder executable can be run from the command line as follows:

```
xt-run  xa_opus_celt_dec_test  -fs:<sampling  rate>  -numch:<channels>
[options] -ifile:<input_file> -ofile:<output_file>
```

Where:

`<input_file>:`	input encoded speech file;
`<output_file>:`	output file with decoded samples;
`[options]:`	
`-fs:`	8000 | 12000 | 16000 | 24000 | 48000; default: 48000 (for raw opus streams)
`-numch:`	1-8; default: 2 (for raw opus streams)
`-loss:`	<perc> simulate packet loss, in percent (0-100); default: 0;
`-btype:`	<ogg/raw> bitstream type, ogg or raw; default: raw;
`-maxpage:n`	Max Ogg page size: ‘n' in kBytes, range: [1, 1024], default: 64;
`-strmap:<map>`	Output channel map, string of max. 8 characters, range for each character 0-8. 0-7 to route the corresponding channel from decoder output, 8 for silence, for streams with less than 8 channels (n) range becomes 0-(n-1), default: taken from opus header. For interpretation of channels please refer to RFC7845 Section 5.1.1.2.

Note Default values for options `-fs` and `-numch` are applicable only for raw opus streams. For Ogg opus streams, `-fs` and `-numch` are taken from ogg stream header without any check if not passed on command line.

Note If value for `-strmap` is specified partially, remaining channels are silenced, if it is longer than number of channels then extra values are ignored. This option is ignored for raw opus streams and channel mapping family 0 ogg opus streams.

If no command line arguments are given, the encoder or decoder application reads the commands from the parameter file `paramfilesimple_encode_celt.txt` or `paramfilesimple_decode_celt.txt`, respectively.

5. Appendix – Ogg Container Format Support in Decoder

The HiFi Opus decoder supports decoding of Opus streams in Ogg container format. Ogg container format support is provided through a separate HiFi Ogg Parser interface. This HiFi Ogg Parser implements Ogg encapsulation standard defined in RFC 3533 [3] and RFC 7845 [4] and is based on libogg 1.3.2 [5].

Note	The HiFi Ogg Parser only supports parsing of Ogg streams (decode) and not packing of Ogg streams (encode) in this version.

The HiFi Ogg Parser performance is mentioned in Section 1.4. The following sections briefly describe the HiFi Ogg Parser details.

5.1 Files Specific to the Ogg Parser

The Ogg Parser parameter header file (`include/ogg_lib`):

`xa_ogg_lib_api.h`

5.2 I/O Formats

The HiFi Ogg Parser does not put any restriction on alignment and size of input buffer. Smaller input buffer size may require multiple calls to the Ogg Parser process function to parse one Ogg page. The test bench application allocates 4 kBytes input buffer.

The HiFi Ogg Parser does not put any restriction on alignment and size of output buffer.

Note	If the output buffer is not sufficient to hold the produced Opus packet, the HiFi Ogg Parser will return a fatal error indicating required size of output buffer.

For more information, see Section 5.4.3. The test bench application uses 3 kBytes output buffer to hold two encoded Opus packets.

5.3 Configuration Structure for Ogg Parser

Table 5-1 xa_ogg_parse_init_cfg_t Structure Parameters

Parameter Name	Value Type	Range / Supported Values	Default	Description
max_page_size	WORD32	1 to 1024	64	Maximum page size to be allocated in Kbytes. Values outside the range are ignored and default is used in that case.

5.4 API Functions

The Ogg Parser API functions relevant to each stage in the codec flow are specified in the following sections.

5.4.1Memory Allocation Stage

During the memory allocation stage, the application needs to reserve the necessary memory for the HiFi Ogg Parser handles (persistent state) and scratch buffers. The required alignment of the handles and the scratch buffers is eight bytes. The application can use the functions listed in Table 5-2 to query the Ogg Parser for the required size of each buffer. The functions take a pointer of type `xa_ogg_parse_init_cfg_t` and return `WORD32`. `xa_ogg_parse_init_cfg_t` is described in Table 5-3.

`xa_ogg_parse_init_cfg_t` is a structure that contains the initialization parameters for this instance of the Ogg Parser. The default initial configuration parameters are used if `NULL` is passed, instead of a valid pointer.

While input and output frame buffers are required for the operation of the component, they do not need to be reserved at this stage. Pointers to the frame buffers are passed in each invocation of the main component execution function.

The size and alignment requirements of the I/O buffers are specified in Section 5.2.

Table 5-2 Memory Management Functions

Function	Description
`Xa_ogg_parse_get_handle_byte_size`	Returns the size of the HiFi Codec API handle (persistent state) in bytes.
`Xa_ogg_parse_get_scratch_byte_size`	Returns the size of the HiFi Codec scratch memory.

Example

```
WORD32 handle_size;
WORD32 scratch_size;
handle_size = xa_ogg_parse_get_handle_byte_size(NULL);
scratch_size = xa_ogg_parse_get_scratch_byte_size(NULL);
```

Errors

None

5.4.2Initialization Stage

In the initialization stage, the application points the HiFi Ogg Parser to its API handle and scratch buffer. The application also specifies various other parameters related to the operation of the Ogg Parser and places the Ogg Parser in its initial state. The API functions for the HiFi Ogg Parser initialization are specified in Table 5-3.

Table 5-3 HiFi Ogg Parser Initialization Function Details

Function	xa_ogg_parse_init

Syntax	`XA_ERRORCODE`
	`xa_ogg_parse_init (`
	`xa_codec_handle_t handle,`
	`pVOID scratch,`
	`xa_ogg_parse_init_cfg_t *cfg)`

Description	Resets the HiFi Ogg Parser API handle into its initial state. Sets up the component to run using the supplied scratch buffer and the specified initial configuration parameters.
Parameters	**Input:** handle Pointer to the component persistent memory. This is an opaque handle. Required size: See xa_ogg_parse_get_handle_byte_size Required alignment: 8 bytes **Input:** scratch Pointer to the component scratch buffer Required size: See xa_ogg_parse_get_scratch_byte_size Required alignment: 8 bytes **Input:** p_cfg Initial configuration parameters (see Section 5.3). **Note:** The initial configuration parameters MUST be identical to those passed during the memory allocation stage.

Example

```
xa_codec_handle_t handle =
      (xa_codec_handle_t)malloc(handle_size);
pVOID scratch = malloc(scratch_size);
res = xa_ogg_parse_init(handle, scratch, NULL);
```

Errors

XA_API_FATAL_MEM_ALLOC

`handle` or `scratch` is `NULL`

XA_API_FATAL_MEM_ALIGN

`handle` or `scratch` is not aligned correctly

5.4.3Execution Stage

The HiFi Ogg Parser processes the input stream and generates the output stream frame-by-frame. Each call to the component execution function produces one complete Opus packet as output, if sufficient input data is provided. If partial input data is presented, it is buffered internally, and need-more-data is signaled to the application through proper error code. The application fills the input buffer with next input data and call the execution function again.

The codec uses default parameters to perform the processing. These parameters can be changed or queried at any time after the initialization stage.

Table 5-4 Execution Stage Functions

Function	Description
`xa_ogg_parse_process`	Processes one frame of audio data
`xa_ogg_parse_set_param`	Sets the value of a particular parameter
`xa_ogg_parse_get_param`	Returns the value of a particular parameter

The syntax of the execution stage functions is specified in the following tables.

HiFi Ogg Parser Process Function

Table 5-5 HiFi Ogg Parser Process Function Details

Function	xa_ogg_parse_process
Syntax	XA_ERRORCODE xa_ogg_parse_process (xa_codec_handle_t handle, pUWORD8 in_data, pUWORD8 out_data, pWORD32 num_bytes_in, pWORD32 num_bytes_out)
Description	Processes one frame of audio using the current component state.
Parameters	**Input**: handle Opaque component handle. **Input**: in_data A pointer to the input audio stream from which the input data will be read. This is the input buffer. Required alignment: 1 byte **Output**: out_data A pointer to the output audio stream into which the output data will be written. This is the output buffer. Required alignment: 1 byte

	Input/Output: num_bytes_in Pointer to the number of input bytes. This contains the amount of data to be processed. On return, *num_bytes_in is set to the actual amount of data processed. **Input/Output**: num_bytes_out Pointer to the number of output bytes. This contains the space available in the output buffer, in terms of bytes. On return, *num_bytes_out is set to the actual number of output bytes generated by the codec. In case, if the output space is smaller than output bytes generated, *num_bytes_out would indicate new output space required.

Example

```
UWORD8 p_in_data[512];
UWORD8 p_out_data[512];
WORD32 in_samples = 512;
WORD32 out_samples = 512;
res = xa_ogg_parse_process(handle, p_in_data, p_out_data,
                    &in_samples, &out_samples);
```

Errors

XA_API_FATAL_MEM_ALLOC

One of the pointers (`handle`, `p_in_data`, `p_out_data`, `p_in_samples`, or `p_out_samples`) is `NULL`.

XA_API_FATAL_MEM_ALIGN

Codec handle is not aligned correctly.

XA_OGG_EXECUTE_FATAL_NOT_INITIALIZED

Ogg Parser is not initialized.

XA_OGG_EXECUTE_FATAL_PAGE_SIZE_TOO_SMALL

Maximum page size allocated is too small to fit one page for current input stream.

Application should re-initialize Ogg Parser with higher page size.

XA_OGG_EXECUTE_FATAL_CORRUPTED_STREAM

Corrupted input stream, two streams with same Opus serial number detected in input stream.

XA_OGG_EXECUTE_FATAL_EXTRA_PKTS_ON_HEADER_PAGE

Corrupted input stream, extra packets detected on header page.

XA_OGG_EXECUTE_FATAL_EXTRA_PKTS_ON_TAGS_PAGE

Corrupted input stream, extra packets detected on tags page.

XA_OGG_EXECUTE_FATAL_OUT_BUF_TOO_SMALL

Output buffer space is too small for output generated for current input stream.

Application should reinitialize Ogg Parser with higher output buffer size, *num_bytes_out should indicate new required size in this case.

XA_OGG_EXECUTE_NONFATAL_INSUFFICIENT_DATA

Insufficient input data.

Application should call the process function again with new input data.

XA_OGG_EXECUTE_NONFATAL_STREAM_CHANGE

A new Opus stream (Opus header) is found before end-of-stream for the current stream.

Application should treat the current output packet as Opus header and next output packet as tags packet and process them accordingly. The Opus decoder should be reinitialized with new stream parameters parsed above.

XA_OGG_EXECUTE_NONFATAL_PAGE_IGNORED

Spurious page ignored (e.g. pages before Opus header page, pages with different serial number than the one currently being decoded).

HiFi Ogg Parser Set Parameter Function

Table 5-6 HiFi Ogg Parser Set Parameter Function Details

Function	xa_ogg_parse_set_param
Syntax	XA_ERRORCODE xa_ogg_parse_set_param (xa_codec_handle_t handle, xa_ogg_parse_param_id_t param_id, pVOID p_param_value)
Description	Sets the parameter specified by param_id to the value passed in the buffer pointed to by p_param_value. **Note:** *This function is provided only for API completeness. There is no config parameter to be set.*
Parameters	**Input:** handle Opaque component handle. **Input:** param_id Identifies the parameter to be written. No parameters are supported for this function in this version. **Input:** p_param_value A pointer to a buffer that contains the parameter value. Required alignment: 4 bytes

Errors

XA_API_FATAL_MEM_ALLOC

One of the pointers (`handle` or `p_param_value`) is `NULL`

XA_API_FATAL_MEM_ALIGN

One of the pointers (`handle` or `p_param_value`) is not aligned correctly.

XA_OGG_EXECUTE_FATAL_NOT_INITIALIZED

Function is called before HiFi Ogg Parser is initialized.

XA_OGG_CONFIG_NONFATAL_INVALID_PARAM

`param_id` is not valid.

HiFi Ogg Parser Get Parameter Function

Table 5-7 HiFi Ogg Parser Get Parameter Function Details

Function	xa_ogg_parse_get_param

Syntax	XA_ERRORCODE
	xa_ogg_parse_get_param (
	    xa_codec_handle_t handle,
	    xa_ogg_parse_param_id_t param_id,
	    pVOID p_param_value)

Description	Gets the value of the parameter specified by param_id in the buffer pointed to by p_param_value
Parameters	**Input:** handle Opaque component handle. **Input:** param_id Identifies the parameter to be read. Refer to section 5.4.4 for the list of parameters supported. **Output:** p_param_value A pointer to a buffer that is filled with the parameter value when the function returns. Required alignment: 4 bytes

Example

```
WORD32 param_id = XA_OGG_PARSE_PARAM_PAGE_GRANULE;
WORD64 granule_pos;
res = xa_ogg_parse_get_param(handle, param_id, &granule_pos);
```

Errors

XA_API_FATAL_MEM_ALLOC

One of the pointers (`handle` or `p_param_value`) is `NULL`

XA_API_FATAL_MEM_ALIGN

One of the pointers (`handle` or `p_param_value`) is not aligned correctly

XA_OGG_EXECUTE_FATAL_NOT_INITIALIZED

Function is called before HiFi Ogg Parser is initialized

5.4.4HiFi Ogg Parser Parameters

This section describes the parameters that are supported by the `xa_ogg_parse_set_param` and `xa_ogg_parse_get_param` functions described in the previous section. Table 5-8 contains the following columns:

Parameter name: Parameter identifier (param_id).

Value type: A pointer (p_param_value) to a variable of this type is to be passed.

RW: Indicates whether the parameter can be read (get) and/or written (set).

Range: The range or supported values of the parameter value.

Default: Default value of the parameter.

Description: Brief description of the parameter.

Table 5-8 HiFi Ogg Parser Parameters

Parameter Name	Value Type	RW	Range / Supported Values	Default	Description
XA_OGG_PARSE_PARAM _PAGE_GRANULE	WORD 64	R	Any positive value	NA	Gets the granule position for current Ogg page. The pointer to this variable must be 8 bytes aligned.
XA_OGG_PARSE_PARAM _END_OF_STREAM	WORD 32	R	0, 1	NA	Gets value 1 if end of stream is detected in the stream, else 0.

6. Reference

[1] Definition of the Opus Audio Codec @ http://tools.ietf.org/html/rfc6716. Updates to the Opus Audio Codec draft-valin-codec-opus-update-00 @ http://tools.ietf.org/html/draft-valin-codec-opus-update-00

[2] Reference Source Code (libopus 1.3.1) @ http://opus-codec.org/downloads/

[3] The Ogg Encapsulation Format @ https://www.xiph.org/ogg/doc/rfc3533.txt

[4] Ogg Encapsulation for Opus Audio Codec @ https://tools.ietf.org/html/rfc7845

[5] downloads.xiph.org/releases/ogg/libogg-1.3.2.tar.gz 5

