cadence®

LC3 Codec

Programmer's Guide

For HiFi DSPs

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

Version: 1.8
Last Updated: Sept 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 customer release.
1.1	Added performance numbers for HiFi 3/3z/4/5 and Fusion F1.
1.2	Support added for LC3 specification version 0.9 that enables the use of 7.5 ms frame size.
1.3	Now supports three separate packages: lc3_codec, lc3_dec, and lc3_enc.
1.4	Codebase upgraded to support LC3 specification version 1.0.
1.5	Codebase updated with HiFi 1 specific optimizations.
1.6	Updated performance data for all HiFi cores.
1.7	■ Advanced Packet Loss Concealment support added to Decoder. ■ Stream based scratch optimization implemented.
1.8	Updated Performance numbers in section 1.4.

1. Introduction to HiFi LC3 Codec

The HiFi LC3 codec is a library supplied by Cadence that implements the Low Complexity Communication Codec (LC3) specifications version 1.0 on HiFi processors.

1.1 LC3 Description

LC3 Low Complexity Communication Codec is the standard for all-band audio communication. LC3 codec obtains very high audio quality at medium bitrates utilizing a low computational complexity. LC3 incorporates flexible coding modes and includes audio bandwidth from narrow-band (NB) to full-band (FB) at bitrates from 16 kbps to 426.667 kbps. The codec operates with associated channel coding in a rate-adaptive scheme, improving degraded channels' quality and resilience.

1.2 Document Overview

This guide covers all the information required to integrate the HiFi LC3 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 integration.

- Section 2 describes the APIs common to all HiFi codecs.
- Section 3 describes all the features and information particular to the HiFi LC3.
- Section 4 describes an example test bench.
- Section 5 lists the reference materials.

1.3 HiFi LC3 Codec Specifications

The HiFi DSP LC3 codec from Cadence implements the following features:

- Uses Cadence Speech Codec API.
- Encoder and Decoder based on LC3 Codec specification 1.0
- Encoder input/decoder output PCM: 16, 24 bits per sample.
- In the case of 24-bit PCM, aligns the input MSB in a 32-bit register.

- Supports sampling frequencies: 8 kHz, 16 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz.
- Supports five audio bandwidths:
    - Narrowband(NB): speech or audio sampled at 8 kHz.
    - Wideband(WB): speech or audio sampled at 16 kHz.
    - Semi-Super Wideband (SSWB): speech or audio sampled at 24 kHz.
    - Super Wideband (SWB): speech or audio sampled at 32 kHz.
    - Full-band (FB): speech or audio sampled at 44.1 kHz or 48 kHz.
- Supports bitrates: The relation between bitrate, sampling frequency, and frame size is shown in the following equation: $bitrate = \frac{8000 \times nbytes \times N}{F_{scale} \times Frame_{ms}}$

  Here,
    - nbytes is an integer number and ranges over [20, 400].
    - $Frame_{ms}$ is frame size in milliseconds.
    - The factor $8000/F_{scale}$ is used for frequency scaling for 44.1kHz case and comes out 7350. For other sampling frequencies $F_{scale} = 1$

Table 1-1 Relation between Sampling Frequency and Supported Bitrate

Framesize (ms)	Sampling Frequency (kHz)	Supported bitrate range (kbps)
10	8	16 – 114.4
	16	16 – 221.6
	24	16 – 314.4
	32/48	16 – 320
	44.1	14.7 – 294
7.5	8	21.334 – 152.534
	16	21.334 – 295.467
	24	21.334 – 419.2
	32/48	21.334 - 426.667
	44.1	19.6 – 392

- The decoder comes with the following two packages:
    - o Advanced packet loss concealment
    - o Standard packet loss concealment (PLC).
- The encoder supports one or two audio channels.
- For stereo channel or two-channel cases, the encoder expects deinterleaved input, and the decoder gives output in the same format.
- Frame duration:10 ms, 7.5ms.

1.4 HiFi LC3 Performance

The HiFi LC3 codec from Cadence is based on the HiFi 5-stage DSP. You can use the memory usage and performance figures for design reference.

The HiFi LC3 consists of two separate release packages: lc3_codec and lc3_codec_aplc. You can pick one of these two packages at each communication terminal to suit your needs.

1.4.1 Memory

Table 1-2 Read Only (RO) Memory

Library	Text (Kbytes)					Data Kbytes
	HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
lc3_codec	87.6	94.8	95.0	101.6	110.6	42.4
lc3_dec	41.4	45.7	46.1	49.2	53.0	35.0
lc3_enc	64.7	71.0	70.0	75.7	82.5	41.5
lc3_codec_aplc	105.7	114.0	115.2	122.6	133.4	49.9
lc3_aplc_dec	60.7	66.2	67.4	71.6	77.2	42.5

Table 1-3 Runtime Memory

Library	Run Time Memory (Kbytes)				
	Persistent	Scratch	Stack	Input	Output
Encoder	6.8	6.8	2.1	3.8	0.9
Decoder	10.1	7.6	2.1	0.9	3.8
Decoder with APLC	19.1	14.5	2.2	0.9	3.8

Note Memory size is the worst case for all test scenarios.

1.4.2 Timings

Encoder

Table 1-4 Encoder Performance

Rate kHz	nch	Bit Rate kbps	Average CPU Load (MHz)					
			HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
48	1	80	9.7	11.5	9.7	9.5	9.1	Frame=10ms
48	1	320	12.9	15.1	12.6	12.4	11.9	Frame=10ms
48	1	80	10.7	12.5	10.6	10.4	10.0	Frame=7.5ms
48	1	320	13.7	16.0	13.3	13.2	12.7	Frame=7.5ms

Decoder

Table 1-5 Decoder Performance

Rate kHz	nch	Bit Rate kbps	Average CPU Load (MHz)					
			HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
48	1	80	4.1	4.9	4.2	4.1	3.8	Frame=10ms
48	1	320	7.2	9.1	8.1	7.9	7.6	Frame=10ms
48	1	80	4.6	5.4	4.6	4.5	4.3	Frame=7.5ms
48	1	320	7.7	9.6	8.5	8.3	8.1	Frame=7.5ms

Decoder with Advanced PLC

Table 1-6 Decoder with Advanced PLC Performance

Corruption (Rate: 48KHz)	nch	Bit Rate kbps	Average CPU Load (MHz)					
			HiFi 1	HiFi 3	HiFi 3z	HiFi 4	HiFi 5	
9.3% corruption	1	80	4.5	5.3	4.6	4.5	4.2	Frame=10ms
9.3% corruption	1	320	7.2	9.1	8.1	7.9	7.6	Frame=10ms
9.65% corruption	1	80	5.1	6.1	5.1	5.1	4.8	Frame=7.5ms
9.65% corruption	1	320	7.8	9.8	8.6	8.5	8.2	Frame=7.5ms

Note The above performance numbers are measured with RI-2023.11 tools and XT-CLANG compiler.

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

Note The MCPS numbers for HiFi 3z/HiFi 4/HiFi 5 are obtained by running the test recompiled from the HiFi 3 optimized source code in the HiFi 3z/HiFi 4/HiFi 5 configuration. No specific optimization is performed for HiFi 3z/HiFi 4/HiFi 5. Some specific optimizations are also made for HiFi 1.

Note The input and output buffer sizes above are for 16-bit PCM samples.

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 the following diagram.

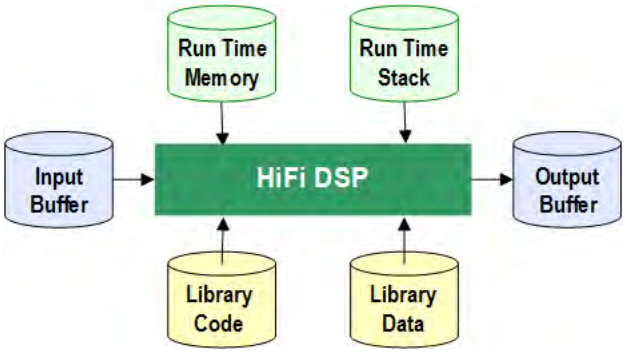

Figure 1 HiFi Speech Codec Interfaces

Section 2.1 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 must 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 used freely 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

An overview of the codec flow is shown in Figure 1. The speech codec API consists of query, initialization, and execution functions. In the naming scheme below, `<codec>` is either the codec name (for example, `LC3`), or the codec name with an `_enc` or `_dec` suffix for encoder- and decoder-specific functions, respectively.

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.

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.

Execution Functions: `xa_<codec>`

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

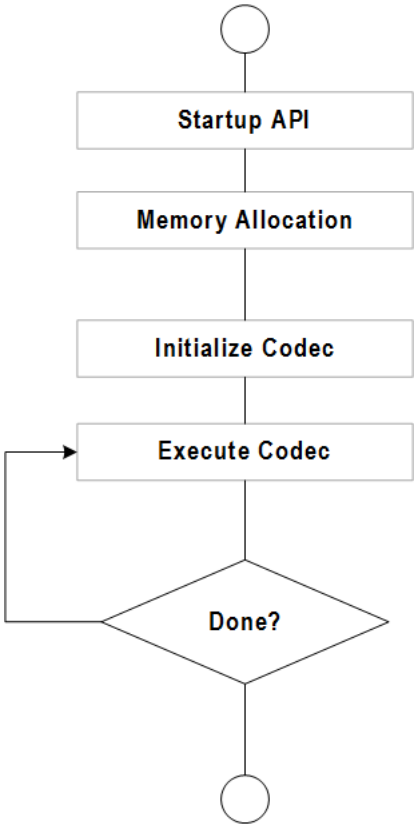

Figure 2 Speech Codec Flow Overview

2.3 Generic API Errors

Speech codec API functions return error code of type `XA_ERRORCODE`, which 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 must 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

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 LC3 Codec API

The HiFi DSP LC3 codec conforms to the generic speech codec API. The supported I/O formats, bit rates, as well as API functions, and files specific to the LC3 codec are described in the following sections.

3.1 Files Specific to the LC3 Codec

The LC3 codec parameter header file (`include/lc3_codec`):

- `xa_lc3_codec_api.h`

The LC3 codec library(`lib`):

- `xa_lc3_codec.a`

3.2 I/O Formats

The input for the encoder is a block of 16- or 24-bit speech samples representing speech frame data. The encoder can accept input signals of sampling frequency from 8 kHz, 16 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz.

The decoder processes the encoded bitstream frame by frame. Under normal conditions, the decoder requires one complete packet in the input buffer, and it outputs frames composed of 16 or 24-bit PCM samples. The decoder can generate output signals of sampling frequency from 8 kHz, 16 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz. The encoder generates (or Decoder processes) bitstream in packed or G192 bitstream format.

3.3 Control Structure for LC3 Codec

Encoder control structure `xa_lc3_enc_init_cfg_t` is used to communicate various parameters (listed in Table 3-1) between the codec library and the application.

Table 3-1 Encoder Control Structure xa_lc3_enc_init_cfg_t Parameters

Parameter	Input/ Output	Default Value	Description
`sampleRate`	input	8000	Input signal sampling rate in Hz. It is described in Section 1.3. Data type: UWORD32
`nChannels`	input	1	Number of channels of the input signal; 1 or 2 Data type: WORD16
`Bitrate`	input	32000	Bitrate in bps. It is described in Section 1.3. Data type: WORD32
`bipsIn`	input	16	Input data pcm-width; 16 or 24 bits per sample. Data type: WORD16

The decoder control structure `xa_lc3_dec_init_cfg_t` is used to communicate various parameters (listed in Table 3-2) between the codec library and the application.

Table 3-2 Decoder Control Structure xa_lc3_dec_init_cfg_t Parameters

Parameter	Input/ Output	Default Value	Description
`sampleRate`	input	8000	Output signal sampling rate in Hz. It is described in Section 1.3. Data type: UWORD32
`nChannels`	input	1	Number of channels of the output signal; 1 or 2. Data type: WORD16
`bipsOut`	input	16	Output data pcm_width; 16 or 24 bits per sample. Data type: WORD32

3.4 API Functions

The following sections provide the LC3 Codec API functions relevant to each stage in the codec flow.

3.4.1 Startup Stage

You can use the API startup functions described in Table 3-3 to get the various identification strings from the codec library. These functions are for information purpose 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_lc3_get_lib_name_string	Gets the name of the library.
xa_lc3_get_lib_version_string	Gets the version of the library.
xa_lc3_get_lib_api_version_string	Gets the version of the API.

Example

```
const char *name = xa_lc3_get_lib_name_string();
const char *ver = xa_lc3_get_lib_version_string();
const char *apiver = xa_lc3_get_lib_api_version_string();
```

Errors

- None

3.4.2 Memory Allocation Stage

During the memory allocation, the application must reserve the necessary memory for the LC3 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. The functions take a pointer of type `xa_lc3_enc_init_cfg_t` and `xa_codec_handle_t` for encoder and `xa_lc3_dec_init_cfg_t` and `xa_codec_handle_t` for decoder, and return `WORD32`.

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

Table 3-4 Memory Management Functions

Function	Description
xa_lc3_enc_get_handle_byte_size	Returns the LC3 Encoder API handle (persistent state) size in bytes.
xa_lc3_dec_get_handle_byte_size	Returns the LC3 Decoder API handle (persistent state) size in bytes.
xa_lc3_enc_get_scratch_byte_size	Returns the LC3 Encoder scratch buffer size.
xa_lc3_dec_get_scratch_byte_size	Returns the LC3 Decoder scratch buffer size.

Example

```
WORD32 enc_handle_size, dec_handle_size;
WORD32 enc_scratch_size, dec_scratch_size;
xa_lc3_enc_init_cfg_t enc_control;
xa_lc3_dec_init_cfg_t dec_control;


enc_handle_size = xa_lc3_enc_get_handle_byte_size(
                          &enc_control);
dec_handle_size = xa_lc3_dec_get_handle_byte_size(
                          &dec_control);
enc_scratch_size = xa_lc3_enc_get_scratch_byte_size(
                          encoder_state);
dec_scratch_size = xa_lc3_dec_get_scratch_byte_size(
                          decoder_state);
```

Errors

- XA_LC3_CONFIG_FATAL_SAMP_FREQ_NOT_SUPPORTED

  The sampling frequency is not supported.

- XA_LC3_CONFIG_FATAL_NUM_CHANNEL_NOT_SUPPORTED

  The number of the channel not supported.

3.4.3 Initialization Stage

In the initialization stage, the application points the LC3 codec to its API handle and scratch buffer. The application also specifies various other parameters related to the codec's operation and places it in its initial state. The API functions for LC3 Encoder and Decoder initialization are defined in Table 3-5 and Table 3-6, respectively.

LC3 Encoder Initialization Function

Table 3-5 LC3 Encoder Initialization Function

Function	`xa_lc3_enc_init`

Syntax	`XA_ERRORCODE`
	`xa_lc3_enc_init (xa_codec_handle_t handle,`
	`pWORD32 scratch_init,`
	`xa_lc3_enc_init_cfg_t *enc_control,`
	`WORD16 frame_dms);`

Description	Resets the encoder API handle to its initial state. Sets up the encoder to run using the supplied scratch buffer and the specified initial configuration parameters.
Parameters	• Input: `handle` Pointer to the Encoder handle (persistent state) Required size: see xa_lc3_enc_get_handle_byte_size Required alignment: 8 bytes • Input: `scratch_init` This is the max encoder scratch init memory used during initialization. *xa_lc3_enc_init()* calculates the actual scratch size required for the *xa_lc3_enc_process()* function. You can free this memory after initialization. You must then make a call to *xa_lc3_enc_get_scratch_byte_size()*, to get the actual scratch size required for encoder process function and allocate the memory accordingly. • Input: `frame_dms` Input frame size value in DMS (100 or 75) • Input: `enc_control` Pointer to the encoder control structure Refer to *xa_lc3_enc_init_cfg_t* described in section 3.3.

Example

```
xa_codec_handle_t handle =
      (xa_codec_handle_t)malloc(handle_size);
pWORD32 scratch_init = (pWORD32)malloc(scratch_size);
xa_lc3_enc_init_cfg_t enc_control;

error_code = xa_lc3_enc_init (handle,
                              scratch_init,
                              &enc_control, frame_dms);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_API_FATAL_MEM_ALIGN
- XA_LC3_CONFIG_FATAL_SAMP_FREQ_NOT_SUPPORTED

  The sampling frequency is not supported.
- XA_LC3_CONFIG_FATAL_NUM_CHANNEL_NOT_SUPPORTED

  The number of channels is not supported.
- XA_LC3_CONFIG_FATAL_ENC_BITRATE_NOT_SUPPORTED

  Bitrate is not supported.
- XA_LC3_CONFIG_FATAL_BITDEPTH_NOT_SUPPORTED

  Bitdepth is not supported.

LC3 Decoder Initialization Function

Table 3-6 LC3 Decoder Initialization Function

Function	`xa_lc3_dec_init`

Syntax	`XA_ERRORCODE`
	`xa_lc3_dec_init (`
	`xa_codec_handle_t handle,`
	`pWORD32 scratch_init,`
	`xa_lc3_dec_init_cfg_t *dec_control,`
	`WORD32 out_samples, WORD16 frame_dms)`

Description	Resets the decoder API handle to its initial state. Sets up the decoder to run using the supplied scratch buffer and specified initial configuration parameters.
Parameters	• Input: `handle` Pointer to the decoder handle (persistent state). Required size: see *xa_lc3_dec_get_handle_byte_size* Required alignment: 8 bytes • Input: `dec_control` Pointer to the decoder control structure. Refer to *xa_lc3_dec_init_cfg_t* described in section 3.3. • Input `scratch_init` This is the max decoder scratch init memory during initialization. `xa_lc3_dec_init()` calculates the actual scratch size required for the `xa_lc3_dec_process()` function. You can free this memory after initialization. You must then make a call to `xa_lc3_dec_get_scratch_byte_size()`, to get the actual scratch size required for the decoder-process function and allocate the memory accordingly. • Input: `frame_dms` Input frame size in DMS (100 or 75). • Input: `out_samples` Number of output samples per channel.

Example

```
xa_codec_handle_t handle =
      (xa_codec_handle_t)malloc(handle_size);
pWORD32 scratch = (pWORD32)malloc(scratch_size);
xa_lc3_dec_init_cfg_t dec_control;
error_code = xa_lc3_dec_init( handle,
                              scratch_init
                              &dec_control, out_samples,
frame_dms);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_API_FATAL_MEM_ALIGN
- XA_LC3_CONFIG_FATAL_SAMP_FREQ_NOT_SUPPORTED

  Not valid setting of sampling frequency.
- XA_LC3_CONFIG_FATAL_NUM_CHANNEL_NOT_SUPPORTED

  The number of channels is not supported.
- XA_LC3_CONFIG_FATAL_BITDEPTH_NOT_SUPPORTED

  Bit-depth is not supported.

3.4.4 Execution Stage

The LC3 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.

The execution stage uses the following macros to calculate I/O buffer sizes and to configure LC3.

Table 3-7 Macros used to calculate I/O buffer size

Defines with Values	Description
XA_LC3_MAX_CHANNELS 2	Maximum number of channels
XA_LC3_MAX_SAMPLES 480	Maximum number of samples in a frame
XA_LC3_MAX_BYTES 870	Maximum number of bytes in a frame

The syntax of the LC3 Encoder and Decoder execution functions are specified in Table 3-8 and Table 3-9, respectively.

LC3 Encoder Execution Function

Table 3-8 LC3 Encoder Execution Function

Function	`xa_lc3_enc_process`

Syntax	`XA_ERRORCODE`
	`xa_lc3_enc_process (`
	`xa_codec_handle_t handle,`
	`pVOID p_in_data,`
	`pVOID p_out_data,`
	`pWORD32 scratch,`
	`pUWORD32 p_in_samples,`
	`pUWORD32 p_out_bytes,`
	`WORD32 scratch_size);`

Description	Encodes one frame of data.
Parameters	• Input: `handle` Pointer to the encoder handle (persistent state). • Input: `p_in_data` Pointer to the input speech PCM samples. You must specify at least one frame of data as input. Required alignment: 8 bytes • Output: `p_out_data` Pointer to the encoded data. Required alignment: 8 bytes • Input: `scratch` Pointer to the scratch_buffer. • Input/Output: `p_in_samples` Pointer to the number of samples in the input buffer. • Input/Output: `p_out_bytes` Pointer to the number of bytes in the output buffer generated by the encoder.
Note	None

Example

```
xa_codec_handle_t handle;
pVOID inp_buf;
pVOID out_buf;
pUWORD32 p_in_samples;
pUWORD32 p_out_bytes;
WORD32 scratch_size;
pWORD32 scratch;

error_code = xa_lc3_enc_process(handle,
                        inp_buf,
                        out_buf,
                        scratch,
                        p_in_samples,
                        p_out_bytes,
                        scratch_size);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_API_FATAL_MEM_ALIGN
- XA_LC3_EXECUTE_FATAL_NOT_INITIALIZED

  LC3 not initialized.

LC3 Decoder Execution Function

Table 3-9 LC3 Decoder Execution Function

Function	`xa_lc3_dec_process`

Syntax	`XA_ERRORCODE`
	`xa_lc3_dec_process (`
	`xa_codec_handle_t handle,`
	`pVOID p_in_data,`
	`pVOID p_out_data,`
	`pWORD32 scratch,`
	`pUWORD32 p_in_bytes,`
	`pUWORD32 p_out_samples,`
	`WORD32 scratch_size,`
	`WORD32 bfi_ext);`

Description	Decodes one frame of data
Parameters	• Input: `handle` Pointer to the decoder handle (persistent state). • Input: `p_in_data` Pointer to the input bytes of a bitstream. Required alignment: 8 bytes • Output: `p_out_data` Pointer to the decoded PCM data. Required alignment: 8 bytes • Input: `scratch` Pointer to the scratch buffer. • Input/Output: `p_in_bytes` Pointer to the number of bytes in the input buffer. • Input/Output: `p_out_samples` Pointer to the number of samples in the output buffer. • Input: `bfi_ext` Bad frame indicator value given from testbench.
Note	None

Example

```
xa_codec_handle_t handle;
pVOID inp_buf;
pVOID out_buf;
pUWORD32 p_in_bytes;
pUWORD32 p_out_samples;
WORD32 scratch_size;
WORD32 bfi_ext;
pWORD32 scratch;

error_code = xa_lc3_dec_process(handle,
                        inp_buf,
                        out_buf,
                        scratch,
                        p_in_bytes,
                        p_out_samples,
                        scratch_size,
                        bfi_ext);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_API_FATAL_MEM_ALIGN
- XA_LC3_EXECUTE_FATAL_NOT_INITIALIZED

  LC3 is not initialized.
- XA_LC3_EXECUTE_FATAL_DEC_NUMBYTES_ERROR

  The invalid number of bytes.
- XA_LC3_EXECUTE_NONFATAL_DECODE_ERROR

  Frame failed to decode, hence concealed.

LC3 Encoder Set Parameter Function

Table 3-10 LC3 Encoder Set Parameter Function

Function	`xa_lc3_enc_set_param`

Syntax	`XA_ERRORCODE`
	`xa_lc3_enc_set_param (`
	`xa_codec_handle_t handle,`
	`xa_lc3_enc_param_id_t param_id,`
	`pVOID p_param_value);`

Description	Sets the parameter value specified by `param_id` to the value passed in the buffer pointed by `p_param_value`.
Parameters	• Input: `handle` Pointer to the encoder handle (persistent state). • Input: `param_id` Identifies the parameter to be written. • Input: `p_param_value` Pointer to the buffer that contains the parameter value.
Note	None

Example

```
xa_codec_handle_t handle;
xa_lc3_enc_param_id_t param_id;
pVOID p_param_value;

error_code = xa_lc3_enc_set_param (handle,
                                   param_id,
                                   p_param_value);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_LC3_CONFIG_FATAL_ENC_BITRATE_NOT_SUPPORTED

  Invalid bitrate.
- XA_LC3_FRAMEMS_NOT_SUPPORTED
- XA_LC3_CONFIG_NONFATAL_INVALID_PARAM

  Parameter identifier (`param_id`) is not valid.

Table 3-11 LC3 Encoder Set Parameter param_id

param_id	Description
XA_LC3_ENC_PARAM_BITRATE	Sets the encoder bitrate
XA_LC3_ENC_PARAM_FRAME_DMS	Sets the encoder frame size in DMS.

LC3 Encoder Get Parameter Function

Table 3-12 LC3 Encoder Get Parameter Function

Function	xa_lc3_enc_get_param
Syntax	XA_ERRORCODE xa_lc3_enc_get_param (xa_codec_handle_t handle, xa_lc3_enc_param_id_t param_id, pVOID p_param_value);
Description	Gets the parameter value specified by param_id in the buffer pointed by p_param_value.
Parameters	• Input: handle Pointer to the encoder handle (persistent state). • Input: param_id Identifies the parameter to be written. • Output: p_param_value Pointer to the buffer that contains the parameter value.
Note	None

Example

```
xa_codec_handle_t handle;
xa_lc3_enc_param_id_t param_id;
pVOID p_param_value;

error_code = xa_lc3_enc_get_param (handle,
                                   param_id,
                                   p_param_value);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_LC3_CONFIG_NONFATAL_INVALID_PARAM

  Parameter identifier (`param_id`) is not valid.
- XA_LC3_EXECUTE_FATAL_NOT_INITIALIZED

  LC3 not initialized.

Table 3-13 Encoder Get Parameter `param_id`

param_id	Description
XA_LC3_ENC_PARAM_FRAME_LENGTH	Gets the encoder frame length
XA_LC3_ENC_PARAM_REAL_BITRATE	Gets the encoder real bitrate
XA_LC3_ENC_PARAM_DELAY_COMPENSATION	Gets the delay compensation in the encoder.

LC3 Decoder Get Parameter Function

Table 3-14 LC3 Decoder Get Parameter Function

Function	`xa_lc3_dec_get_param`

Syntax	`XA_ERRORCODE`
	`xa_lc3_dec_get_param (`
	`xa_codec_handle_t handle,`
	`xa_lc3_dec_param_id_t param_id,`
	`pVOID p_param_value);`

Description	Gets the parameter value specified by `param_id` in the buffer pointed by `p_param_value`.
Parameters	• Input: `handle` Pointer to the decoder handle (persistent state). • Input: `param_id` Identifies the parameter to be written. • Output: `p_param_value` Pointer to the buffer that contains the parameter value.
Note	None

Example

```
xa_codec_handle_t handle;
xa_lc3_dec_param_id_t param_id;
pVOID p_param_value;

error_code = xa_lc3_dec_get_param (handle,
                                   param_id,
                                   p_param_value);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_LC3_CONFIG_NONFATAL_INVALID_PARAM

  Parameter identifier (`param_id`) is not valid.
- XA_LC3_EXECUTE_FATAL_NOT_INITIALIZED

  LC3 not initialized.

Table 3-15 LC3 Decoder Get Parameter `param_id`

param_id	Description
XA_LC3_DEC_PARAM_DELAY_COMPENSATION	Gets the delay compensation in the decoder.
XA_LC3_DEC_PARAM_FRAME_LENGTH	Gets the decoder frame length

LC3 Decoder Set Parameter Function

Table 3-16 LC3 Decoder Set Parameter Function

Function	`xa_lc3_dec_set_param`

Syntax	`XA_ERRORCODE`
	`xa_lc3_dec_set_param (`
	`xa_codec_handle_t handle,`
	`xa_lc3_dec_param_id_t param_id,`
	`pVOID p_param_value);`

Description	Sets the parameter value specified by `param_id` to the value passed in the buffer pointed by `p_param_value`.
Parameters	• Input: `handle` Pointer to the decoder handle (persistent state). • Input: `param_id` Identifies the parameter to be written.

	• Input: `p_param_value` Pointer to the buffer that contains the parameter value.
Note	None

Example

```
xa_codec_handle_t handle;
xa_lc3_dec_param_id_t param_id;
pVOID p_param_value;

error_code = xa_lc3_dec_set_param (handle,
                                   param_id,
                                   p_param_value);
```

Errors

- XA_API_FATAL_MEM_ALLOC
- XA_LC3_FRAMEMS_NOT_SUPPORTED
- XA_LC3_CONFIG_NONFATAL_INVALID_PARAM

  Parameter identifier (`param_id`) is not valid.

Table 3-17 LC3 Decoder Set Parameter `param_id`

param_id	Description
XA_LC3_DEC_PARAM_FRAME_DMS	Sets the decoder frame size in DMS.

4. Introduction to the Example Test Bench

The LC3 codec library provides two sample test bench applications: one for the encoder and one for the decoder. The LC3 codec library is released as a .tgz file for Linux/makefile-based usage and an .xws file for Xtensa Xplorer-based usage. The supplied test benches consist of the following files:

- Test bench source files (`test/src`)
  - `xa_lc3_decoder_sample_testbench.c`
  - `xa_lc3_encoder_sample_testbench.c`
  - `xa_lc3_codec_error_handler.c`
- Makefile to build the executables (`test/build`)
  - `makefile_testbench_sample`

4.1 Making the Executable

4.1.1 Building LC3 Applications

Build Encoder and Decoder Testbenches

To build the applications from the codec .tgz based release package, follow these steps:

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 encoder example test bench `xa_lc3_enc_test` and the decoder example test bench `xa_lc3_dec_test` are built.

Build Either an Encoder or a Decoder Testbench

To build only encoder or only decoder application from the codec .tgz based release package, follow these steps:

1. In the command prompt, navigate to the `test/build` directory.
2. Enter the following command:
   ```
   xt-make –f makefile_testbench_sample clean [lc3_dec|lc3_enc]
   ```

The encoder example test bench `xa_lc3_dec_test` or the decoder example test bench `xa_lc3_enc_test` is built.

4.1.2 Building LC3 Applications with Advanced PLC

Build Encoder Testbench and Decoder with Advanced PLC Testbench

To build the application from the codec with Advanced PLC .tgz based release package, follow these steps:

1. In the command prompt, navigate to the `test/build` directory.
2. Enter the following command:
   `xt-make –f makefile_testbench_sample clean all_aplc`

The encoder example test bench `xa_lc3_enc_test` and the decoder with Advanced PLC example test bench `xa_lc3_aplcdec_test` are built.

Build Either an Encoder Testbench or a Decoder with Advanced PLC Testbench

To build only encoder or decoder with Advanced PLC based application from the codec with Advanced PLC .tgz based release package, follow these steps:

1. In the command prompt, navigate to the `test/build` directory.
2. Enter the following command:
   `xt-make –f makefile_testbench_sample clean [lc3_aplcdec|lc3_enc]`

The decoder example test bench `xa_lc3_aplcdec_test` or the encoder example test bench `xa_lc3_enc_test` is built.

Note If you have source code distribution for the codec package, you must build the LC3 codec library before building the test bench.

4.1.3 Building Libraries

Build LC3 Codec Library

To build the LC3 codec library, follow these steps:

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

The LC3 library `xa_lc3_codec.a` is built and copied to the `lib` directory.

Build Either an Encoder or a Decoder Library

To build the LC3 decoder or encoder library, follow these steps:

1. In the command prompt, navigate to the `build` directory.
2. Enter the following command:
   `xt-make clean [lc3_dec|lc3_enc] install`.

The LC3 library `xa_lc3_dec.a` or `xa_lc3_enc.a` is built and copied to the `lib` directory respectively.

Note If you have source code distribution for the codec with Advanced PLC package, you must build the LC3 codec with Advanced PLC library before building the testbench.

4.1.4 Building Libraries with Advanced PLC

Build Codec Libraries with Advanced PLC

To build the LC3 Advanced PLC decoder library, follow these steps:

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

The LC3 library `xa_lc3_codec_aplc.a` is built and copied to the `lib` directory.

Build Either an Encoder or a Decoder Library with Advanced PLC

To build the LC3 decoder with Advanced PLC /LC3 encoder library, follow these steps:

1. In the command prompt, navigate to the `build` directory.
2. Enter the following command:
   `xt-make clean [lc3_aplcdec/lc3_enc] install.`

The LC3 library `xa_lc3_aplc_dec.a` or `xa_lc3_enc.a` is built and copied to the `lib` directory.

Note To build and execute the application from .xws based release package, refer to the readme.html file available in the imported application project.

4.2 Usage

You can run the executable sample application using command-line options or a parameter file. For more information about executing the sample application from Xtensa Xplorer workspace, refer to the readme.html file available in the imported project.

4.2.1 Encoder

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

```
xt-run xa_lc3_enc_test [OPTIONS] <INPUT> <OUTPUT> <BITRATE>
```

Where:

`< INPUT >:`	Input WAV file (1 or 2 channels, 16 or 24 bit-depth, sample-rate: 8, 16, 24, 32, 44.1 or 48 kHz)
`< OUTPUT >:`	Output encoded bitstream file
`< BITRATE >:`	Target bitrate, specified in bits per second (real bitrate would be an approximation of target bitrate as per Table 1-1). Alternatively, you can specify a switching file.
`[OPTIONS]:`	
`-frame_ms:`	NUM Frame length in ms. NUM must be 10 (default) or 7.5
`-formatG192:`	Activate G192 bitstream format. It uses .cfg file to store/load decoder info.
`-cfgG192 FILE:`	Specify a configuration file for G192 bitstream format.
`-epf FILE:`	Enable packet loss simulation using error pattern from FILE
`-dc NUM:` 0:	Do not use delay compensation
1:	Compensate delay in the decoder (default)
2:	Split delay equally between encoder and decoder
`-ept:`	Use together with `-epf FILE` to create bitstream triggering PLC via the special value of lastnz.
`-y:`	StartFrame: frame number where encoding/decoding shall start.
`-z:`	StopFrame: frame number where encoding/decoding shall stop.

4.2.2 Decoder

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

```
xt-run xa_lc3_dec_test [OPTIONS] <INPUT> <OUTPUT>
```

Where:

`< INPUT >:`	Input encoded bitstream file
`< OUTPUT >:`	Output decoded WAV file
`[OPTIONS]:`	

`-bps NUM:`		Output bits per sample. NUM must be 16 (default) or 24
`-formatG192:`		Activate G192 bitstream format. It uses a .cfg to store/load decoder info.
`-cfgG192 FILE:`		Specify a configuration file for G192 bitstream format.
`-dc NUM:`	0:	Do not use delay compensation
	1:	Compensate delay in the decoder (default)
	2:	Split delay equally between encoder and decoder
`-epf FILE:`		Enable packet loss simulation using error pattern from FILE
`-edf FILE:`		Write error pattern as detected to FILE
`-y:`		StartFrame: frame number where encoding/decoding shall start.
`-z:`		StopFrame: frame number where encoding/decoding shall stop.

Note An invalid input argument (starting with an alphabet) to `-dc NUM` results in no delay compensation.

4.2.3 Decoder with Advanced PLC

This decoder package supports Advanced Packet Loss Concealment that uses concealment technique based on the Time Domain designed by Fraunhoffer. It is best suited for monophonic signals with a periodic structure, such as voiced speech.

The command-line options for Decoder with Advanced PLC are same as the normal decoder as described in the Decoder section.

Note Use the `xa_lc3_aplcdec_test` binary name for the LC3 Decoder with Advanced PLC.

If you do not provide command line arguments, the encoder or decoder application reads the commands from the parameter file `paramfilesimple_encode.txt` or `paramfilesimple_decode.txt`, respectively.

The following is the syntax for writing the `paramfilesimple`:

```
@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
```

Using different command lines, you can run the LC3 encoder and decoder for multiple test files. 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 must be at the first column of a line except the command @New_line.

Note All the @<command>s are case-sensitive. Use the command @New_line to span the command across lines in the parameter file.

For example,

```
<command line part 1> @ New_line
<command line part 1>
```

Note It ignores blank lines.

Note To comment an individual line, use "//" at the beginning of the line.

5. Reference

[1] *LC3 Specification version 1.0:*
https://www.bluetooth.com/specifications/specs/low-complexity-communication-codec-1-0/

[2] *LC3 Test suite version 1.0,6:*
https://www.bluetooth.com/specifications/specs/low-complexity-communication-codec-1-0/

[3] *LC3 Reference binary*
https://www.bluetooth.org/DocMan/DocInfo.aspx?doc_id=497700

[4] *LC3 with Advanced PLC reference binary*
This reference binary is provided by Fraunhoffer with the version: v1.6.6.