cadence®

G.711 Codec

Programmer's Guide

For HiFi DSPs and Fusion F1 DSP

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

Version 1.2
August 2018

Contents

Figures

Tables

Document Change History

Version	Changes
1.1	• Updated memory data and timing data in Section 1.4. • Updated parameters information in Table 3-5, Table 3-6, and Table 3-7.
1.2	• Added performance numbers for HiFi 3z/HiFi 4 and Fusion F1 in Section 1.4.

1. Introduction to the HiFi G.711 Speech Codec

The HiFi DSP G.711 Codec implements the speech codec standard specified as ITU-T recommendation G.711 [1], and includes support for Packet Loss Concealment described in Appendix I [2].

For this document, HiFi DSPs include Fusion F1 DSP.

1.1 G.711 Description

G.711 is a speech codec commonly used in Voice over IP telephony, video conferencing, wireless GPRS EDGE systems, and others. The G.711, also known as Pulse Code Modulation (PCM) of voice frequencies, is commonly used in waveform codec. The G.711 codec processes speech streams with a sample rate of 8 kHz. The G.711 encoder compresses input samples from 16 bits per sample to 8 bits per sample, achieving a compressed bandwidth of 64 Kbit/s. The G.711 decoder performs the reverse process.

Two variants of the G.711 codec exist: A-Law and µ-Law. These variants are inherited from the switching telephone networks around the world. The two variants are equivalent in coding efficiency and have similar computational requirements.

A Packet Loss Concealment option in the decoder can be used to improve the perceived audio quality under the presence of packet loss. Lost frames are filled by a synthetic signal with similar audio characteristics as the preceding speech. The use of the PLC will increase the CPU consumption of the G.711 decoder.

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; these are described after the introduction. Section 3 covers all the features and information particular to the HiFi G.711 Codec. Section 4 describes the example test bench.

1.3 HiFi G.711 Codec Specifications

The HiFi DSP G.711 Codec from Cadence Tensilica implements the following features:

- Cadence Speech Codec API is used
- ITU-T G.711 compliant encoder and decoder
- Runtime option for G.711 Appendix I packet loss concealment2
- Sample rate: 8 kHz
- Bit rate: 64 Kbit/s

This codec implementation has been verified to be bit exact with the ITU-T G.711 conformance test sequences. The PLC implementation has been verified to produce equivalent audio quality measurements as the reference implementation from ITU.

1.4 HiFi G.711 Codec Performance

The HiFi DSP G.711 Codec from Cadence Tensilica was characterized on the 5-stage HiFi DSP core. The memory usage and performance figures are provided for design reference.

1.4.1 Memory

Text (Kbytes)						Data Kbytes
HiFi Mini	HiFi 2	Fusion F1	HiFi 3	HiFi 3z	HiFi 4	
3.5	3.9	4.2	4.8	4.2	4.8	0.1

	Run Time Memory (Kbytes)				
	Persistent	Scratch	Stack	Input	Output
Encoder	0.1	0	0.1	0.2	0.1
Decoder	1.9	0.2	0.3	0.1	0.2

1.4.2 Timings

Encoder

Rate kHz	nch	Bit Rate kbps	Average CPU Load (MHz)						
			HiFi Mini	HiFi 2	Fusion F1	HiFi 3	HiFi 3z	HiFi 4	
8	1	64	0.17	0.17	0.14	0.16	0.13	0.13	A law
8	1	64	0.18	0.17	0.14	0.15	0.12	0.12	μ law

Decoder

Rate kHz	nch	Bit Rate kbps	Average CPU Load (MHz)						
			HiFi Mini	HiFi 2	Fusion F1	HiFi 3	HiFi 3z	HiFi 4	
8	1	64	0.16	0.16	0.11	0.16	0.11	0.10	A law, PLC=off
8	1	64	0.16	0.16	0.11	0.16	0.11	0.10	μ law, PLC=off
8	1	64	0.44	0.41	0.26	0.37	0.26	0.25	A law, PLC=on
8	1	64	0.44	0.41	0.27	0.37	0.26	0.25	μ law, PLC=on

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 is equivalent to running with all code and data in local memories or using an infinite-size, pre-filled cache model. The performance numbers for HiFi Mini/HiFi 3/HiFi 3z/HiFi 4/Fusion F1 are obtained by running the test that is recompiled from the HiFi 2 source code in the HiFi Mini/HiFi 3/HiFi 3 z /HiFi 4/Fusion F1 configuration. No specific optimization is done for HiFi Mini/HiFi 3/HiFi 3 z /HiFi 4/Fusion F1.

Note The CPU load with PLC depends on the frequency of frame erasures.

2. Generic HiFi Speech Codec API

This chapter describes the API that is common to all the HiFi audio codec libraries. The API facilitates any codec that works in the overall method shown in the following diagram.

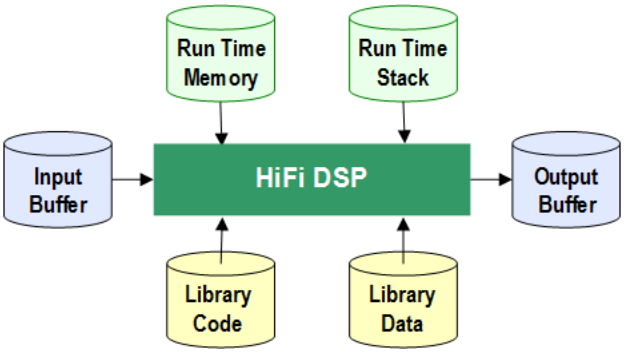

Figure 1 HiFi Audio Codec Interfaces

Section 0 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 persistent state information in a structure that is referenced via an opaque handle. The handle is passed by the application for each API call. This object contains all state and history information that is maintained from one codec frame 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 during a single frame processing call. The contents of this memory region should not be changed if the actual codec execution process is active, *i.e.*, 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 2. The speech codec API consists of query, initialization, and execution functions. In the naming scheme below, `<codec>` is either the codec name (for example, `g711`), 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.

The G.711 Codec sequence, as well as the functions associated with each stage, is described in detail in Section 3.

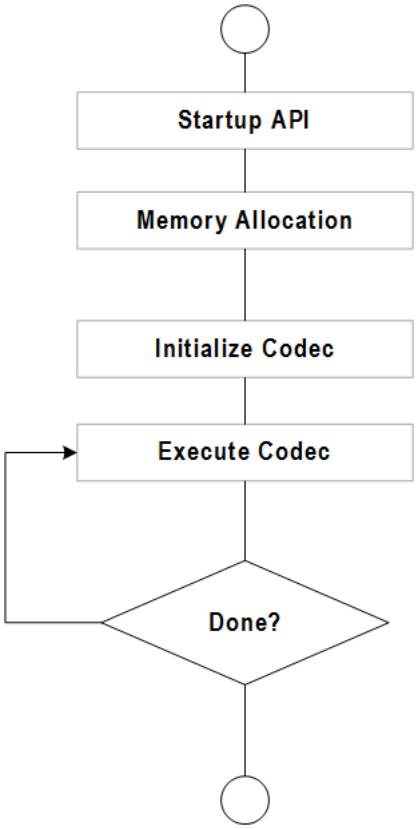

Figure 2 Speech Codec Flow Overview

2.3 Generic API Errors

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

Table 2-1 Error Codes Format

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 API, Config, or Execution. The API 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

All these errors are fatal and should not be encountered during a 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`

  The macros and definition for error handler

- `xa_type_def.h`

  All the types required for the API calls

3. HiFi DSP G.711 Codec API

The HiFi DSP G.711 Codec conforms to the generic speech codec API and the codec flow shown in Figure 2. The supported I/O formats and frame types, as well as the API functions and the files specific to the G.711 Codec are described below.

3.1 Files Specific to the G.711 Codec

The G.711 codec API header file (`include/g711_codec`)

- `xa_g711_codec_api.h`

The G.711 codec library (`lib`)

- `xa_g711_codec.a`

3.2 I/O Formats

The input for the encoder is a block of 80 speech samples, representing 10ms of speech data. The samples should be produced using an 8 KHz sampling rate and must be the 13-bit uniform PCM. Each sample should occupy one 16-bit word, thus a total of 80 endian-dependent 16-bit words should be given as an input. Similarly, the decoder will output a block of 80 speech samples at a 8 KHz sampling rate. Each sample will occupy one 16-bit word. Each PCM sample uses the lower 13 bits of the 16-bit word.

If the decoder is enabled with PLC, then the output PCM samples from the decoder will experience a 30 samples delay as specified in Appendix I2 contributing to 3.75ms delay. Otherwise no delay will be observed in the decoded PCM output.

The encoded bitstream format is a bit-packed version of the conformance format. The packed bitstream is endian-independent—the first bit of the stream is always the most significant bit of the first byte.

3.2.1 Encoding Law

The G.711 encoder and decoder must be configured for operation either as A-law or as µ-law. The law type is defined by an enumeration type `xa_g711_law_t`, listed in Table 3-1.

Table 3-1 Encoding Laws

Law Type	Description
`XA_G711_LAW_A`	A-law encode/decode
`XA_G711_LAW_U`	µ-law encode/decode

3.2.2 Frame Types

When PLC is enabled, each input frame to the decoder must be identified as being either a normal frame, or an erased (lost) frame. This indication is done using the enumeration `xa_g711_frame_type_t`. Table 3-2 describes each frame type.

Table 3-2 Frame Modes

Frame Type	Description
`XA_G711_FRAME_NORMAL`	Normal frame.
`XA_G711_FRAME_SID`	Reserved for future use.
`XA_G711_FRAME_ERASURE`	Erased (lost) frame. It only has an effect when PLC is enabled.

3.3 API Functions

The G.711 Codec API functions relevant to each stage in the codec flow are specified in the following sections.

3.3.1 Startup 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_g711_get_lib_name_string`	Get the name of the library.
`xa_g711_get_lib_version_string`	Get the version of the library.
`xa_g711_get_lib_api_version_string`	Get the version of the API.

Example

```
const char *name = xa_g711_get_lib_name_string();
const char *ver = xa_g711_get_lib_version_string();
const char *apiver = xa_g711_get_lib_api_version_string();
```

Errors

- None

3.3.2 Memory Allocation Stage

During the memory allocation stage, the application must reserve the necessary memory for the G.711 encoder and decoder API handles (persistent state) and scratch buffers. The required alignment of the scratch buffers is 4 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 no input arguments and return WORD32. Note that encoder do not require scratch memory so return value of `xa_g711_dec_get_scratch_byte_size` function is zero.

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 3.3.4.

Table 3-4 Memory Management Functions

Function	Description
`xa_g711_enc_get_handle_byte_size`	Return the size of the G.711 Encoder API handle (persistent state) in bytes.
`xa_g711_dec_get_handle_byte_size`	Return the size of the G.711 Decoder API handle (persistent state) in bytes.
`xa_g711_enc_get_scratch_byte_size`	Return the size of the G.711 Encoder scratch memory.
`xa_g711_dec_get_scratch_byte_size`	Return the size of the G.711 Decoder scratch memory.

Example

```
WORD32 enc_handle_size, dec_handle_size;
WORD32 enc_scratch_size, dec_scratch_size;

enc_handle_size = xa_g711_enc_get_handle_byte_size();
dec_handle_size = xa_g711_dec_get_handle_byte_size();
enc_scratch_size = xa_g711_enc_get_scratch_byte_size();
dec_scratch_size = xa_g711_dec_get_scratch_byte_size();
```

Errors

- None

3.3.3 Initialization Stage

In the initialization stage, the application points the G.711 codec to its API handle and scratch buffer. 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 G.711 encoder and decoder initialization are specified in Table 3-5 and Table 3-6, respectively.

Table 3-5 G.711 Encoder Initialization Function

Function	`xa_g711_enc_init`

Syntax	`XA_ERRORCODE`
	`xa_g711_enc_init (`
	`xa_codec_handle_t handle,`
	`pVOID scratch )`

Description	Reset the encoder API handle into its initial state. Set up the encoder to run using the supplied scratch buffer in the specified encoded speech format.
Parameters	**Input:** `handle` Pointer to the Encoder persistent memory. This is the opaque handle. Required size: see `xa_g711_enc_get_handle_byte_size`. Required alignment: 4 bytes. **Input:** `scratch` Pointer to the Encoder scratch buffer. Required size: Encoder does not need scratch memory Required alignment: None

Example

```
xa_codec_handle_t enc_handle =
      (xa_codec_handle_t)malloc(enc_handle_size);
pVOID enc_scratch = (pVOID)malloc(enc_scratch_size);
res = xa_g711_enc_init(enc_handle,
                       enc_scratch );
```

Errors

- XA_API_FATAL_MEM_ALLOC

  handle is `NULL`

- XA_API_FATAL_MEM_ALIGN

  handle is not 4-byte aligned

Table 3-6 G.711 Decoder Initialization Function

Function	`xa_g711_dec_init`

Syntax	`XA_ERRORCODE`
	`xa_g711_dec_init (`
	`xa_codec_handle_t handle,`
	`pVOID scratch,`
	`WORD32 plc_enable)`

Description	Reset the decoder API handle into its initial state. Setup the decoder to run using the supplied scratch buffer in the specified encoded speech format.
Parameters	**Input:** `handle` Pointer to the Decoder persistent memory. This is the opaque handle. Required size: see `xa_g711_dec_get_handle_byte_size`. Required alignment: 4 bytes. **Input:** `scratch` Pointer to the Decoder scratch buffer. Required size: see `xa_g711_dec_get_scratch_byte_size`. Required alignment: 4 bytes. **Input:** `plc_enable` Configuration of the PLC . 0 - to disable PLC 1 - to enable PLC

Example

```
xa_codec_handle_t dec_handle =
      (xa_codec_handle_t)malloc(dec_handle_size);
WORD32 plc_enable = 0;
pVOID dec_scratch = (pVOID)malloc(dec_scratch_size);
res = xa_g711_dec_init(dec_handle, dec_scratch,
                       plc_enable);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  handle or scratch is `NULL`.

- XA_API_FATAL_MEM_ALIGN

  scratch is not 4-byte aligned

3.3.4 Execution Stage

The G.711 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 frame as output.

The G.711 encoder takes as input one 8 kHz 10 ms speech frame containing 16-bit samples. Since the input frame contains 80 samples, the input buffer of the G.711 encoder is 160 bytes. The encoder generates 80 bytes of encoded speech codewords per frame. Correspondingly, the G.711 decoder's output buffer is 160 bytes, and the input buffer is 80 bytes. The alignment information of these I/O buffers is given in the description of the execution functions.

The following macros specify the relevant I/O buffer sizes:

- `XA_G711_SAMPLES_PER_FRAME: 80`
- `XA_G711_MAX_NUM_BITS_PER_FRAME: 640`
- `XA_G711_MAX_NUM_BYTES_PER_FRAME:`

  `(XA_G711_MAX_NUM_BITS_PER_FRAME >> 3)`

The G.711 encoder execution function requires as input the encoding law (A-Law or µ-Law) for every frame. The G.711 decoder execution function requires as input the encoding law, as well as the frame type (normal or erasure).

The syntax of the G.711 encoder and decoder execution functions are specified in Table 3-7 and Table 3-8 respectively.

Table 3-7 G.711 Encoder Execution Function

Function	`xa_g711_enc`

Syntax	`XA_ERRORCODE`
	`xa_g711_enc (`
	`xa_codec_handle_t handle,`
	`pWORD16 inp_speech,`
	`pUWORD8 enc_speech,`
	`xa_g711_law_t law,`
	`xa_g711_frame_type_t *p_frame_type)`

Description	Encode one frame of speech using the given encoding law.
Parameters	Input: `handle` The opaque Encoder handle. Input: `inp_speech` A pointer to the input speech frame containing 80 16-bit samples. This is the input buffer. Required alignment: 4 bytes. Input: `law` Encoding law (A-Law or µ-Law). Output: `enc_speech` A pointer to the encoded speech frame, containing 80 bytes. This is the output buffer. Required alignment: Byte align. Output: `p_frame_type` Always return `XA_G711_FRAME_NORMAL`.

Example

```
xa_g711_frame_type_t frame_type;
res = xa_g711_enc(enc_handle,
                  inp_speech, enc_speech,
                  XA_G711_LAW_A, &frame_type);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  One of the input pointers is `NULL`

- XA_API_FATAL_MEM_ALIGN

  `inp_speech` is not 4-byte aligned

Table 3-8 G.711 Decoder Execution Function

Function	`xa_g711_dec`

Syntax	`XA_ERRORCODE`
	`xa_g711_dec (`
	`xa_codec_handle_t handle,`
	`pUWORD8 enc_speech,`
	`pWORD16 synth_speech,`
	`xa_g711_law_t law,`
	`xa_g711_frame_type_t frame_type)`

Description	Decode one frame of speech using the specified encoding law and frame type.
Parameters	Input: `handle` The opaque Decoder handle. Input: `enc_speech` A pointer to the encoded speech frame containing 80 bytes. This is the input buffer. Required alignment: Byte align. Input: `law` Encoding law (A-Law or µ-Law). Input: `frame_type` The type of frame (normal or erased). Refer to Table 3.3 Output: `synth_speech` Pointer to the synthesized speech containing 80 16-bit samples. This is the output buffer. Required alignment: 4 bytes.

Example

```
res = xa_g711_dec(dec_handle, enc_speech, synth_speech,
                  XA_G711_LAW_A, XA_G711_FRAME_NORMAL);
```

Errors

- XA_API_FATAL_MEM_ALLOC

  Any of the input pointers is `NULL`

- XA_API_FATAL_MEM_ALIGN

  `synth_speech` is not 4-byte aligned

4. Introduction to the Example Test Bench

The G.711 Codec library is provided with two sample ITU-T conformance test bench applications: one for the encoder and one for the decoder. The supplied test benches contain the following files:

Test bench source files (`test/src`)

- `xa_g711_enc_sample_testbench.c`
- `xa_g711_dec_sample_testbench.c`
- `xa_g711_codec_error_handler.c`

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

- `makefile_testbench_sample`

4.1 Making Executables

To build the application, follow these steps:

1. Go to `test/build`.
2. At the prompt, type
   `xt-make -f makefile_testbench_sample clean all`

This will build the encoder example test bench `xa_g711_enc_test` and the decoder example test bench `xa_g711_dec_test`.

Note: If you have source code distribution, you must build the G.711 library before you can build the testbench. Follow these steps:

1. Go to `build`.
2. At the command prompt, type: `make clean all install`

This builds the G.711 library and copies it to the `lib` directory.

4.2 Usage

Encoder Test Bench

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

```
xt-run xa_g711_enc_test <speech_file> <bitstream_file> <law>
```

Where:

`<speech_file>` Input speech file (80 16-bit samples per frame)

`<bitstream_file>` Output speech file encoded in the ITU-T conformance format (each 8-bit codeword is expanded to 16 bits).

`<law>` A for A-law
U for µ-Law.

Decoder Test Bench

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

```
xt-run xa_g711_dec_test <bitstream_file> <synth_file> <law>
            [-plc] [loss_pattern_file]
```

Where:

`<bitstream_file>` Input speech file encoded in the ITU-T conformance format (each 8-bit codeword is expanded to 16-bits)

`<synth_file>` Output synthesized speech file (80 16-bit samples per frame)

`<law>` A for A-law
U for µ-Law

`-plc`

- This is an optional parameter that forces the G.711 decoder to perform packet-loss concealment. For packets lost according to the pattern-loss file, the output will be filled with silence.
- If the –plc option is provided, the output PCM file will observe 3.75ms 3- PCM sample delay as specified in Appendix-I22.
- If this option is not provided, the decoder library act as simple A or µ law decode and no delay will be observed in output PCM file.

`[loss_pattern_file]`

- This is an optional parameter. If supplied, a G.192-compliant pattern loss file simulates frame loss patterns. If not supplied, the testbench assumes no frame erasures.

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 G.711 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 `@New_line` command.

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

```
<command line part 1> @New_line

<command line part 2>.
```

Note Blank lines will be ignored.

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

5. References

[1] ITU-T Recommendation G.711-Pulse Code Modulation of voice frequencies.

[2] ITU-T Recommendation Appendix-I: A High Quality low Complexity Algorithm for Packet Loss Concealment with G.711