# G.722 Codec

## Programmer's Guide

For:
- Xtensa HiFi Audio Engine

Tensilica, Inc.
3393 Octavius Drive.
Santa Clara, CA 95054

Version 1.0
January 2013
PD-12-8612-10-00

# Contents

# Figures

# Tables

# 1. Introduction to the HiFi G.722 Codec

The HiFi Audio Engine G.722 Codec implements the speech codec standard specified as ITU-T Recommendation G.722 (11/1988)[1], including Erratum 1 (05/2003)[2]. This codec supports packet-loss concealment according to G.722 Appendix IV (11/2009)[3].

## 1.1 G.722 Background

G.722 is an ITU standard codec that provides 7 kHz wide-band audio with  a data rate of 64 kbit/s.  It offers a significant improvement in speech quality over narrow-band codecs such as G.711.The coding system uses sub-band adaptive differential pulse code modulation (SB-ADPCM) with a bit rate of 64 kbit/s.

The system is henceforth referred to as 64 kbit/s (7 kHz) audio coding. In the SB-ADPCM technique used, the frequency band is split into two sub-bands (higher and lower) and the signals in each sub-band are encoded using ADPCM.

Packet loss concealment algorithm, also known as frame erasure concealment algorithms, hide transmission losses in audio systems where the input signal is encoded and packateised. When packaet loss is informed to the G.722 decoder, it performas the analysis of the past lower-band reconstructed signal and extrapolates the missing signal using linear predictive coding (LPC), pitch-synchronous period repetations and adaptive muting. Once good frame is received, the decoded signal is cross faded with the extrapolated signal. In the higher band, decoder reapets the previous frame pith synchronously, with adaptive muting and high pass post-processing. The ADPCM states are updated after each frame erasure.

## 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.

## 1.3 HiFi G.722 Codec Specifications

The HiFi Audio Engine G.722 Codec from Tensilica implements the following features:

- ITU-T G.722 compliant encoder and decoder
- Runtime option for G.722 Appendix IV: A low-complexity algorithm for packet-loss concealment with ITU-T G.722
- Sample rates: 16 kHz
- Bit rate supported : 64 kbit/s
- PCM sample size: 16 bit.

The HiFi G.722 codec implementation is GPL free. The base codec implementation has been verified with the reference vectors provided by ITU G.722 implementation in G.191. The PLC implementation was verified using the test vectors provided with G.722 Annex IV[3].

## 1.4 HiFi G.722 Codec Performance

The HiFi Audio Engine G.722 Codec from Tensilica was characterized on the 5-stage HiFi Audio Engine processor core. The memory usage and performance figures are provided for design reference.

### 1.4.1 Memory

| Text (Kbytes) | | | Data |
|---|---|---|---|
| HiFi Mini | HiFi 2 | HiFi 3 | Kbytes |
| 11.7 | 13.8 | 14.5 | 0.5 |

#### Encoder

| Run Time Memory (Bytes) | | | | |
|---|---|---|---|---|
| Persistent | Scratch | Stack | Input | Output |
| 196 | 0 | 288 | NS*BPS | NS/2 |

#### Decoder without PLC

| Run Time Memory (Bytes) | | | | |
|---|---|---|---|---|
| Persistent | Scratch | Stack | Input | Output |
| 1956 | 1000 | 368 | BS | BS*2*BPS |

## Decoder with PLC

| Frame Size | Run Time Memory (Bytes) | | | | |
|---|---|---|---|---|---|
| msec | Persistent | Scratch | Stack | Input | Output |
| 10 | 1956 | 1000 | 368 | 80 | 320 |
| 20 | 1956 | 1000 | 368 | 160 | 640 |

**Note** NS: Number of PCM sample to be encoded. Each encoded sample is 4 bits.

**Note** BS: Number of bytes to be decoded by the decoder.

**Note** BPS: Number of bytes per PCM sample, which is 2.

Input and output buffers are allocated by the test-bench and buffer sizes are calculated using equations specified in above tables.

### 1.4.2 Timings

#### Encoder

| Data Rate | Frame size | Average CPU Load (MHz) | | |
|---|---|---|---|---|
| kbps | msec | HiFi Mini | HiFi 2 | HiFi 3 |
| 64 | 10 | 5.3 | 5.0 | 5.0 |

#### Decoder

| Data Rate | Frame size | PLC | Average CPU Load (MHz) | | |
|---|---|---|---|---|---|
| kbps | msec | | HiFi Mini | HiFi 2 | HiFi 3 |
| 64 | 10 | off | 4.5 | 4.3 | 4.4 |
| 64 | 10 | on | 4.8 | 4.7 | 4.7 |
| 64 | 20 | on | 4.6 | 4.4 | 4.5 |

**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-filled cache model.

# 2. Generic HiFi Speech Codec API

This chapter 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.

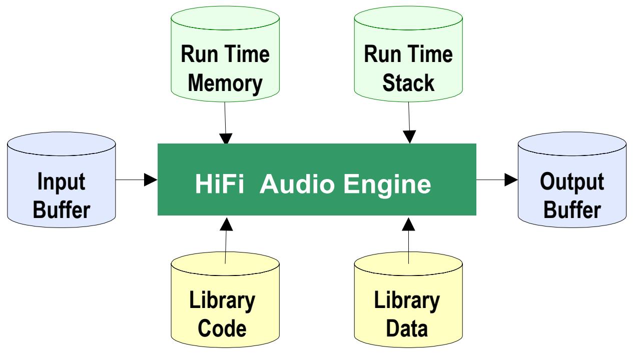

**Figure 1 HiFi Speech Codec Interfaces**

The following section, 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.

### API Handle / Persistent Memory

The codec API stores persistent state information in a structure that is referenced via a 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.

### 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.

### Input Buffer

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

### Output Buffer

This is the buffer in which the algorithm writes the output. This buffer needs to 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, `g722`), 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.722 Codec sequence, as well as the functions associated with each stage, are described in detail in Section 3.

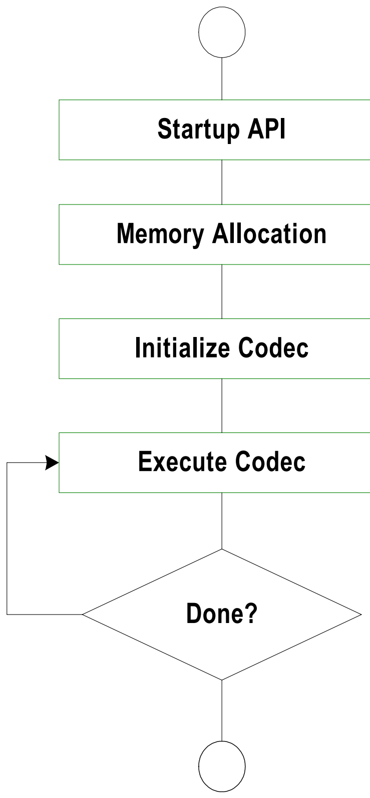

**Figure 2 Speech Codec Flow Overview**

## 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 are 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 sub divided into those that are fatal, which require the restarting of the whole 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

All 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

### 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 Audio Engine G.722 Codec

The HiFi Audio Engine G.722 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.722 Codec are described below.

## *3.1 I/O Formats*

The G.722 encoder takes as input 16 bit PCM samples sampled at 16KHz, and generates compressed output bit stream. Number of samples to be encoded are passed to the encoder execute function. Each PCM sample is encoded to 4 bits. Number of samples to be encoded per execute call should be multiple of 2. Encoder application should decide input buffer size and corresponding output buffer size and allocates the buffers  depending on the number of PCM samples to be encoded per execute API call. G.722 encoded bit stream contains 2 encoded PCM samples per byte..

The G.722 Decoder accepts compressed bitstream and generates PCM  output. Input and output buffer requirements differ depends on whether PLC is enabled at the decoder initialization time.

If the decoder is initialized with PLC disabled, the decoder execute function accepts number of compressed bytes to be decoded. Each compressed byte corresponds to 2 decoded PCM samples. The size of the input and the output buffers should be decided by the decoder application depending on number of bytes to be decoded per execute API call.

If PLC option is enabled, then the input buffer of the decoder must be either 80 or 160 bytes, representing 10ms or 20ms frame length respectively. Accordingly, the decoder outputs 160 or 320 PCM samples that corresponds to 10 ms or 20 ms speech frame respectively.

## *3.2 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_g722_frame_type_e`. Table 3-1 describes each frame type.

Table 3-1 Frame Types

| Frame Type | Description |
|---|---|
| `XA_G722_GOOD_FRAME` | Normal frame. |
| `XA_G722_BAD_FRAME` | Erased (lost) frame. It only has an effect when PLC is enabled. |

## 3.3 API Functions

The G.722 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 below 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 3-2 Library Identification Functions

| Function | Description |
|---|---|
| `xa_g722_get_lib_name_string` | Get the name of the library. |
| `xa_g722_get_lib_version_string` | Get the version of the library. |
| `xa_g722_get_lib_api_version_string` | Get the version of the API. |

#### Example

```
const char *name = xa_g722_get_lib_name_string();
const char *ver = xa_g722_get_lib_version_string();
const char *apiver = xa_g722_get_lib_api_version_string();
```

#### Errors

- None.

### 3.3.2 Memory Allocation Stage

During the memory allocation stage, the application needs to reserve the necessary memory for the G.722 Encoder and Decoder API handles (persistent state) and scratch buffers. The required alignment of the handles and the scratch buffers is 4 bytes. The application can use the functions listed in Table 3-3 to query the codec library for the required size of each buffer. The functions take no input arguments and return `WORD32`**.**

While input and output 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 Execution Stage.

Table 3-3 Memory Management Functions

| Function | Description |
|---|---|
| `xa_g722_enc_get_handle_byte_size` | Returns the size of the G.722 Encoder API handle (persistent state) in bytes. |
| `xa_g722_dec_get_handle_byte_size` | Returns the size of the G.722 Decoder API handle (persistent state) in bytes. |
| `xa_g722_enc_get_scratch_byte_size` | Returns the size of the G.722 Encoder scratch memory. |
| `xa_g722_dec_get_scratch_byte_size` | Returns the size of the G.722 Decoder scratch memory. |

## Example

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

enc_handle_size  = xa_g722_enc_get_handle_byte_size();
dec_handle_size  = Xa_g722_dec_get_handle_byte_size();
enc_scratch_size = xa_g722_enc_get_scratch_byte_size();
dec_scratch_size = xa_g722_dec_get_scratch_byte_size();
```

## Errors

- None

### 3.3.3 Initialization Stage

In the initialization stage, the application points the G.722 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.722 Encoder and Decoder initialization are specified in Table 3-4 and Table 3-5, respectively.

Table 3-4 G.722 Encoder Initialization Function

| **Function** | `xa_g722_enc_init` |
|---|---|
| **Syntax** | `XA_ERRORCODE xa_g722_enc_init(`<br>`          xa_codec_handle_t handle,`<br>`          pVOID scratch)` |
| **Description** | Reset the encoder API handle into its initial state. |
| **Parameters** | **Input:** `handle`<br>Pointer to the Encoder persistent memory. This is an opaque handle.<br>Required size: see `xa_g722_enc_get_handle_byte_size`.<br>Required alignment: 4 bytes.<br><br>**Input:** `scratch`<br>Pointer to the Encoder scratch buffer.<br>Required size: see `xa_g722_enc_get_scratch_byte_size`.<br>Required alignment: None |

#### Example

```
xa_codec_handle_t enc_handle = NULL;
pVOID scratch_encoder = NULL;
enc_handle = (xa_codec_handle_t)malloc(handle_size);
error_code = xa_g722_enc_init(enc_handle,
                              scratch_encoder);
```

#### Errors

- XA_API_FATAL_MEM_ALLOC
  - `handle` is `NULL`.
- XA_API_FATAL_MEM_ALIGN
  - `handle` does not meet the alignment requirements.

Table 3-5 G.722 Decoder Initialization Function

| | |
|---|---|
| **Function** | `xa_g722_dec_init` |
| **Syntax** | `XA_ERRORCODE xa_g722_dec_init(`<br>`        xa_codec_handle_t handle,`<br>`        pVOID scratch,`<br>`        WORD32 plc_enable)` |
| **Description** | Reset the decoder API handle into its initial state. |
| **Parameters** | **Input:** `handle`<br>Pointer to the Decoder persistent memory. This is an opaque handle.<br>Required size: see `xa_g722_dec_get_handle_byte_size`.<br>Required alignment: 4 bytes.<br><br>**Input:** `scratch`<br>Pointer to the decoder scratch buffer.<br>Required size: see `xa_g722_dec_get_scratch_byte_size`.<br>Required alignment: 2 bytes.<br><br>**Input:** `plc_enable`<br>Configuration of the PLC<br>0: core g722 decoder without PLC<br>1: Configure g722 decoder with PLC |

## Example

```
xa_codec_handle_t dec_handle = NULL;
pVOID scratch_decoder = NULL;
WORD32 plc_option;

dec_handle = (xa_codec_handle_t)malloc(handle_size);
error_code = xa_g722_dec_init(
            dec_handle,
            scratch_decoder,
            plc_option);
```

## Errors

- XA_API_FATAL_MEM_ALLOC
  - `handle` is `NULL`.
- XA_API_FATAL_MEM_ALIGN
  - `handle` or `scratch` do not meet alignment requirements.

### 3.3.4 Execution Stage

The G.722 encoder takes as input 16KHz speech samples, and generates compressed bit stream as output. Number of PCM samples to be encoded per execute call is supplied to the encoder execute function. Number of PCM samples to be encoded must be multiple of 2. Please refer to section 3.1 for detils on how to calculate input output buffer sizes for the enocoder.

Similarly, G.722 decoder  accepts compressed bit stream and generates PCM samples. Number of bytes to be decoded per execute call is supplied to the decoder execute function. If the decoder is initialized with PLC mode enabled, then number of bytes to process must be either 80 or 160 bytes, corresponding to 10ms or 20ms frame respetively. This restriction is imposed by the G.722 PLC algorithm which works on 10 or 20ms frames only. Refer to section 3.1 for details on input, output buffer allocaton for the decoder.

The following macros that specify the relevant I/O buffer sizes for the encoder and decoder:

- `XA_G722_MAX_NUM_BITS_PER_ENCODED_SAMPLE: 4`
- `XA_G722_NUM_BITS_PER_PCM_SAMPLE: 16`

The syntax of the G.722 Encoder and Decoder execution functions are specified in Table 3-6 and Table 3-7 respectively.

Table 3-6 G.722 Encoder Execution Function

| | |
|---|---|
| **Function** | `xa_g722_enc` |
| **Syntax** | `XA_ERRORCODE xa_g722_enc (`<br>`      xa_codec_handle_t handle,`<br>`      pWORD16 inp_speech,`<br>`      pWORD8 enc_speech`<br>`      WORD32 num_samples);` |
| **Description** | Encode speech samples. |
| **Parameters** | **Input:** `handle`<br>The opaque Encoder handle.<br>Required alignment: 4 bytes.<br><br>**Input:** `inp_speech`<br>A pointer to the input speech frame.<br>Required alignment: 2 bytes.<br><br>**Input**: `num_samples`<br>Number of PCM sample to be encoded.<br>Restriction: Number of PCM sample to be encoded must be multiple of 2.<br><br>**Output:** `enc_speech`<br>A pointer to the encoded speech frame.<br>This is the output buffer.<br>Required alignment: 1 byte. |

## Example

```
error_code = xa_g722_enc (
            encoder_state,
            inp_speech,
            enc_speech,
            num_samples);
```

## Errors

- XA_API_FATAL_MEM_ALLOC
  - One of the input pointers is `NULL`.
- XA_API_FATAL_MEM_ALIGN
  - `handle` or `inp_speech` do not meet the alignment requirements.
- XA_G722_EXECUTE_FATAL_INVALID_NUM_SAMPLES_TO_PROCESS
  - Number of PCM samples to be encoded should be multiple of 2.

Table 3-7 G.722 Decoder Execution Function

| | |
|---|---|
| **Function** | `xa_g722_dec` |
| **Syntax** | `XA_ERRORCODE xa_g722_dec (`<br>`      xa_codec_handle_t handle,`<br>`      pUWORD8 enc_speech,`<br>`      pWORD16 synth_speech,`<br>`      WORD32 num_bytes,`<br>`      xa_g722_frame_type_e frame_type);` |
| **Description** | Decode number of compressed bytes of speech, depending on the frame type. |
| **Parameters** | **Input:** `handle`<br>The opaque Decoder handle.<br>Required alignment: 4 bytes.<br><br>**Input:** `enc_speech`<br>A pointer to the encoded speech frame.<br>Required alignment: 1 byte.<br><br>**Input:** `num_bytes`<br>Number of input bytes given to the decoder to process.<br><br>**Output:** `synth_speech`<br>Pointer to the synthesized speech.<br>Required alignment: 2 bytes.<br><br>**Input:** `frame_type`<br>According to use of packet-loss concealment , the decoder need to specify the type of frame (See Table 3.1). |

## Example

```
error_code = xa_g722_dec (
            decoder_state,
            inp_speech,
            dec_speech,
            num_bytes,
            frame_type);
```

## Errors

- XA_API_FATAL_MEM_ALLOC
  - One of the input pointers is `NULL`.
- XA_API_FATAL_MEM_ALIGN
  - `handle` is not 4-byte aligned. `synth_speech` is not 2-byte aligned.

- XA_G722_EXECUTE_FATAL_INSUFFICIENT_BYTES_TO_PROCESS
  - This error is generated if the decoder is configured to run with PLC enabled and recives num_bytes are not equal to 80 or 160 bytes.

## 3.4 Files Specific to the G.722 Codec

### The G.722 codec API header file (`include/g722_codec`)

- `xa_g722_codec_api.h`

### The G.722 codec library (`lib`)

- `xa_g722_codec.a`

# 4. Introduction to Example Test Bench

The G.722 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_g722_enc_sample_testbench.c`
- `xa_g722_dec_sample_testbench.c`
- `xa_g722_codec_error_handler.c`

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

- `makefile_testbench_sample`

## 4.1 Making Executables

To build the applications, follow these steps:

1. From the directory: `test/build`.
2. At the command-line prompt, type:

```
xt-make –f makefile_testbench_sample clean xa_g722_codec_test
```

This will build the encoder example test bench `xa_g722_enc_test` and the decoder example test bench `xa_g722_dec_test`.

**Note:**

If you have source code distribution, you must build the G.722 library before you can build the testbench. Follow these steps:

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

The G.722 library will be built and copied 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_g722_enc_test <speech_file> <bitstream_file>
```

Where:

`<speech_file>`

- Input speech file.

`<bitstream_file>`

- Output speech file encoded in the ITU-T conformance format.

### Decoder Test Bench

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

```
xt-run xa_g722_dec_test
            <bitstream_file> <synth_file> [-plc]
            [<loss_pattern_file>] [-20ms]
```

Where:

`<bitstream_file>`

- input speech file encoded in the ITU-T conformance format

`<synth_file>`

- output synthesized speech file.

`-plc`

- This is an optional parameter.
- If supplied, the decode does the PLC according to frame erasure pattern specified by G.192 compatable loss pattern file.
- If not supplied, decoder operates in core decode mode without PLC.

`[loss_pattern_file]`

- This must be specified if –plc option is supiled to decoder. If supplied, a G.192-compliant pattern loss file to simulate frame loss patterns.

`[-20ms]`

- This is an optional parameter. If supplied, the decode use 20ms frames. If not supplied, the testbench assumes 10ms frames.

.

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

The syntax for writing the `paramfilesimple` file is:

```
@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.722 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.

Eg.

```
<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. Reference

[1] International Telecommunication Union, "7 kHz Audio Coding Within 64 KBIT/S." ITU-T Recommendation G.722 (11/88).
http://www.itu.int/rec/T-REC-G.722-198811-I/en

[2] International Telecommunication Union, "Erratum 1 (05/2003) to ITU-T Recommendation G.722 (11/1988)" ITU-T Recommendation G.722 (1988) Erratum 1 (05/03).
http://www.itu.int/rec/T-REC-G.722-200305-I!Err1/en

[3] International Telecommunication Union, *"Appendix IV: A low-complexity algorithm for packet-loss concealment with ITU-T G.722."* ITU-T Recommendation G.722 (1988) Appendix IV (11/09).
*http://www.itu.int/rec/T-REC-G.722-200911-I!AppIV/en*