08-6659-API-ZCH66
MAY 9, 2008
1.0

Application Programmers Interface for G.711 Decoder and Encoder

ABSTRACT:

Application Programmers Interface for G.711 Decoder and Encoder

KEYWORDS:

Multimedia codecs, speech, G.711

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
1.0	09- May-2008	Qiu Cunshou	Initial Draft

Table of Contents

Introduction

1.1 Purpose

This document gives the details of the Application Programming Interface (API) of G.711 encoder and decoder. The G.711 codec is operating system (OS) independent and do not assume any underlying drivers.

1.2 Scope

This document describes only the functional interface of the G.711 codec. It does not describe the internal design of the codec. Specifically, it describes only those functions that are required for this codec to be integrated in a system.

1.3 Audience Description

The reader is expected to have basic understanding of Speech Signal processing and G.711 codec. The intended audience for this document is the development community who wish to use the G.711 codec in their systems.

1.4 References

1.4.1 Standards

- ITU-T Recommendation G.711.

1.4.2 Freescale Multimedia References

- G.711 Codec Application Programming Interface – g711_codec_api.doc
- G.711 Codec Requirements Book – g711_codec_reqb.doc
- G.711 Codec Test Plan - g711_codec_test_plan.doc
- G.711 Codec Release notes - g711_codec_release_notes.doc
- G.711 Codec Test Results – g711_codec_test_results.doc
- G.711 performance Result – g711_codec_perf_results.doc
- G.711 Interface Decoder Header – g711_dec_api.h
- G.711 Interface Encoder Header – g711_enc_api.h
- G.711 Decoder Application Code – g711_decode_test.c
- G.711 Encoder Application Code – g711_encode_test.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
CNG	Comfort Noise Generation
DTX	Discontinuous Transmission
FSL	Freescale
ITU	International Telecommunication Union
MIPS	Million Instructions per Second
OS	Operating System
PCM	Pulse Code Modulation
SID	Silence Insertion Descriptor
RVDS	ARM RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries
VAD	Voice Activity Detection

1.6 Document Location

docs/g.711

2 API Description Encoder

This section describes the steps followed by the application to call the G.711 encoder. During each step the data structures and the functions used will be explained. Pseudo code is given at the end of each step.

2.1 Encoder API Data Types

The member variables inside the structure are prefixed as G711E or APPE together with data types prefix to indicate if that member variable needs to be initialized by the encoder or application calling the encoder.

Step 1: Allocate memory for Encoder config parameter structure

There are 4 different encoder modes,

- A-law
- µ-law
- A-law to µ-law
- µ-law to A-law

The application has to call the encoder routine according to the encoder mode.

Step 2: Get the encoder memory requirements

The G.711 encoder does not do any dynamic memory allocation.

Step 3: Allocate Data Memory for the encoder

The application needn't allocates data memory for the encoder.

Step 4: Initialization routine

There isn't initialization routine for G.711 encoder.

Step 5: Memory allocation for input buffer

For A-law and µ-law encoder, the application has to allocate (2-byte aligned) the memory needed for the input buffer. It is desirable to have the input buffer allocated in FAST_MEMORY, as this may improve the performance (MHz) of the encoder. Pointer to the input buffer needs to be passed to encode routine.

Example pseudo code for allocating the input buffer

```
/* Allocate memory for input buffer of A-law or µ-law encoder */
inBuf = alloc_fast(BUFFER_SIZE * sizeof(G711_S16));

/* Allocate memory for input buffer of A-law to µ-law or µ-law to A-law
converter */
inBuf = alloc_fast(BUFFER_SIZE * sizeof(G711_U8));
```

Step 6: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the encoded bitstream corresponding to input data (PCM sample or A-law or µ-law compressed data). The pointer to this output buffer needs to be passed to encode function. The application can allocate memory for output buffer in external memory using alloc_slow. Allocating memory in internal memory using alloc_fast will improve the performance (MHz) of the encoder marginally.

Example pseudo code for allocating memory for output buffer

```
/* Allocate memory for output buffer of A-law/µ-law encoder or A-law to
µ-law / µ-law to A-law converter */
outBuf = alloc_fast (BUFFER_SIZE * sizeof(G711_U8));
```

Step 7: Call the encode routine

There are 4 different G.711 encoder functions for 4 different encoder modes, respectively. These functions encodes input PCM sample or converts the A-law/µ-law compressed data and writes bitstream to output buffer.

C prototype:

```
/* C prototype for A-law encoder */
void g711AlawEncode(G711_U16 size, G711_S16 *inBuf, G711_U8 *outBuf);

/* C prototype for µ-law encoder */
void g711MulawEncode(G711_U16 size, G711_S16 *inBuf, G711_U8 *outBuf);

/* C prototype for A-law to µ-law converter */
void g711Alaw2Mulaw(G711_U16 size, G711_U8 *inBuf, G711_U8 *outBuf);

/* C prototype for µ-law to A-law converter*/
void g711Mulaw2Alaw(G711_U16 size, G711_U8 *inBuf, G711_U8 *outBuf);
```

Arguments:

- size The number of input samples to be encoded or converted
- inBuf Pointer to input speech buffer
- outBuf Pointer to output (encoded) buffer

Return value:

- none

Example pseudo codes for calling the main encode routine of the encoder.

```
while (the number of input data >= size)
{
      g711AlawEncode(size, inBuf, outBuf);
      or,
      g711MulawEncode(size, inBuf, outBuf);
      or,
      g711Alaw2Mulaw(size, inBuf, outBuf);
      or,
      g711Mulaw2Alaw(size, inBuf, outBuf);
}
```

Step 8: Free memory

The application should release all the memory it allocated before exiting the encoder.

```
free(outBuf);
free(inBuf);
```

3 API Description Decoder

This section describes the steps followed by the application to call the G.711 Decoder. During each step the data structures and the functions used will be explained. Pseudo code is given at the end of each step.

3.1 Decoder API Data Types

The member variables inside the structure are prefixed as G711D or APPD together with data types prefix to indicate if that member variable needs to be initialized by the decoder or application calling the decoder.

Step 1: Allocate memory for Decoder config parameter structure

There are 2 different decoder modes,

- A-law
- µ-law

The application has to call the decoder routine according to the decoder mode.

Step 2: Get the decoder memory requirements

The G.711 decoder does not do any dynamic memory allocation.

Step 3: Allocate Data Memory for the decoder

The application needn't allocates data memory for the decoder.

Step 4: Initialization routine

There isn't initialization routine for G.711 encoder.

Step 5: Memory allocation for input buffer

The application has to allocate the memory needed for the input buffer. It is desirable to have the input buffer allocated in FAST_MEMORY, as this may improve the performance (MHz) of the encoder. Pointer to the input buffer needs to be passed to encode routine.

Example pseudo code for allocating the input buffer

```
/* Allocate memory for input buffer */
inBuf = alloc_fast(BUFFER_SIZE * sizeof(G711_U8));
```

Step 6: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the encoded bitstream corresponding to input data (PCM sample or A-law or µ-law compressed data). The pointer to this output buffer needs to be passed to encode function. The application can allocate memory for output buffer in external memory using alloc_slow. Allocating memory in internal memory using alloc_fast will improve the performance (MHz) of the encoder marginally.

Example pseudo code for allocating memory for output buffer

```
/* Allocate memory for output buffer */
outBuf = alloc_fast (BUFFER_SIZE * sizeof(G711_S16));
```

Step 7: Call the decode routine

There are 2 different G.711 decoder functions for 2 different decoder modes, respectively. These functions decode the A-law/µ-law compressed data and write the linear PCM to output buffer.

C prototype:

```
/* C prototype for A-law decoder */
void g711AlawDecode(G711_U16 size, G711_U8 *inBuf, G711_S16 *outBuf);

/* C prototype for µ-law decoder */
void g711MulawDecode(G711_U16 size, G711_U8 *inBuf, G711_S16 *outBuf);
```

Arguments:

- size The number of input samples to be decoded
- inBuf Pointer to input (encoded) buffer
- outBuf Pointer to output speech buffer

Return value:

- none

Example pseudo codes for calling the main encode routine of the encoder.

```
while (the number of input data >= size)
{
      g711AlawDecode(size, inBuf, outBuf);
      or,
      g711MulawDecode(size, inBuf, outBuf);
}
```

Step 8: Free memory

The application should release memory before exiting the decoder.

```
free(outBuf);
free(inBuf);
```

4 Example calling Routine

4.1 Example calling routine for G.711 Encoder

Below example code gives guidelines for calling G.711 encoder.

```
BUFFER_SIZE = 64;

void main (FILE *pInFile, FILE *pOutFile, unsigned char mode)
{
      short pcm_samples[BUFFER_SIZE];
      unsigned char coded_samples_in[BUFFER_SIZE];
      unsigned char coded_samples[BUFFER_SIZE];
      unsigned int buffer_count = 0;
      int readCount = 0;

      if (A-law encoding)
      {
            readCount = fread ( pcm_samples, sizeof(short), \
                              BUFFER_SIZE, pInFile);
            while(readCount)
            {
                  g711AlawEncode(readCount, pcm_samples, coded_samples);

                  buffer_count = 0;
                  while( buffer_count < readCount )
                  {
                        fwrite( &coded_samples[buffer_count], \
                              sizeof(char),1,pOutFile );
                        buffer_count++;
                  }
                  readCount = fread ( pcm_samples, sizeof(short), \
                                    BUFFER_SIZE, pInFile );
            }
      }
      else if (Mu-law encoding)
      {
            readCount = fread ( pcm_samples, sizeof(short), \
                              BUFFER_SIZE, pInFile );
            while(readCount)
            {
                  g711MulawEncode(readCount, pcm_samples, coded_samples);

                  buffer_count = 0;
                  while( buffer_count < readCount )
```

```
            {
                  fwrite( &coded_samples[buffer_count], \
                        sizeof(char),1,pOutFile );
                  buffer_count++;
            }
            readCount = fread ( pcm_samples, sizeof(short), \
                              BUFFER_SIZE, pInFile );
      }
}
else if (A-law to Mu-law converter)
{
      readCount = fread (coded_samples_in, sizeof(unsigned char), \
                        BUFFER_SIZE, pInFile );
      while(readCount)
      {
            g711Alaw2Mulaw(readCount, coded_samples_in, \
                        coded_samples);

            buffer_count = 0;
            while( buffer_count < readCount )
            {
                  fwrite( &coded_samples[buffer_count], \
                        sizeof(char),1,pOutFile );
                  buffer_count++;
            }
            readCount = fread (coded_samples_in, sizeof(short), \
                              BUFFER_SIZE, pInFile );
      }
}
else if (Mu-law to A-law converter)
{
      readCount = fread (coded_samples_in, sizeof(unsigned char), \
                        BUFFER_SIZE, pInFile );
      while(readCount)
      {
            g711Mulaw2Alaw(readCount, coded_samples_in, \
                        coded_samples);

            buffer_count = 0;
            while( buffer_count < readCount )
            {
                  fwrite( &coded_samples[buffer_count], \
                        sizeof(char),1,pOutFile );
                  buffer_count++;
            }
            readCount = fread (coded_samples_in, sizeof(short), \
                              BUFFER_SIZE, pInFile );
      }
}
```

4.2 Example calling routine for G.711 Decoder

Example calling guidelines for calling the G.726 decoder is given below.

```
BUFFER_SIZE = 64;

void main (FILE *pInFile, FILE *pOutFile, unsigned char mode)
{
      short pcm_samples[BUFFER_SIZE];
      unsigned char coded_samples[BUFFER_SIZE];
      unsigned int buffer_count = 0;
      int readCount = 0;

      if ( A-law decoding )                // selecting A law
      {
            readCount = fread (coded_samples, sizeof(unsigned char), \
                              BUFFER_SIZE, pInFile );
            while(readCount)
            {
                  g711AlawDecode(readCount, coded_samples, pcm_samples);

                  buffer_count = 0;
                  while( buffer_count < readCount )
                  {
                        fwrite( & pcm_samples [buffer_count], \
                              sizeof(short),1,pOutFile );
                        buffer_count++;
                  }
                  readCount = fread (coded_samples, \
                           sizeof(unsigned char), BUFFER_SIZE, pInFile );
            }
      }
      else if (Mu-law decoding)                  // selecting Mu law
      {
            readCount = fread (coded_samples, sizeof(unsigned char), \
                              BUFFER_SIZE, pInFile );
            while(readCount)
            {
                  g711MulawDecode(readCount, coded_samples, pcm_samples);

                  buffer_count = 0;
                  while( buffer_count < readCount )
                  {
                        fwrite( & pcm_samples [buffer_count], \
                              sizeof(char),1,pOutFile );
                        buffer_count++;
                  }
                  readCount = fread (coded_samples, \
                          sizeof(unsigned char), BUFFER_SIZE, pInFile );
            }
   }
```