08-6795-API-ZCH66
JUNE 17, 2008
2.1

Application Programmers Interface for G.726 Decoder and Encoder

ABSTRACT:

Application Programmers Interface for G.726 Decoder and Encoder

KEYWORDS:

Multimedia codecs, speech, G.726

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	30-Jun-2004	Tommy Tang	Initial Draft
0.2	02-Jul-2004	Tommy Tang	Add comments after review
0.3	13-July-2004	Tommy Tang	Added review comments
1.0	03- Aug-2004	Ashok Kumar	Added PCS generic comments
1.1	27-Sep-2004	Tommy Tang	Added comments for ARM1136J-S
1.2	30-Sep-2004	Ashok Kumar	Updated doc
2.0	06-Feb-2006	Lauren Post	Using new format
2.1	17-Jun-2008	Qiu Cunshou	Update doc

Table of Contents

Introduction

1.1 Purpose

This document gives the details of the Application Programming Interface (API) of G.726 encoder and decoder. ITU-T G.726 is a low complexity variable rate (40, 32, 24 or 16 kbps) speech codec based on the principle of Adaptive Differential Pulse Code Modulation (ADPCM).

The G.726 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.726 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.726 codec. The intended audience for this document is the development community who wish to use the G.726 codec in their systems.

1.4 References

1.4.1 Standards

- **ITU-T Recommendation G.726 (1990)** – 40, 32, 24, 16 kbit/s Adaptive differential pulse code modulation (ADPCM).
- **ITU-T Recommendation G.711 (1988)** –Pulse code modulation (PCM) of voice frequencies.
- **G.726 Appendix II Test vectors (1991)** –Description of the digital test sequences for the verification of the G.726 40,32,24 and 16 kbps ADPCM algorithm.

1.4.2 Freescale Multimedia References

- G.726 Codec Application Programming Interface – g726_codec_api.doc
- G.726 Codec Requirements Book – g726_codec_reqb.doc
- G.726 Codec Test Plan - g726_codec_test_plan.doc
- G.726 Codec Release notes - g726_codec_release_notes.doc
- G.726 Codec Test Results – g726_codec_test_results.doc
- G.726 Codec Test Results – g726_codec_perf_results.doc
- G.726 Codec Datasheet – g726_codec_datasheet.doc
- G.726 Interface Common Header – g726_com_api.h
- G.726 Interface Decoder Header – g726_dec_api.h

- G.7261 Interface Encoder Header – g726_enc_api.h
- G.726 Decoder Application Code – g726_dectest.c
- G.726 Encoder Application Code – g726_enctest.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
ADPCM	Adaptive Differential Pulse Code Modulation
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.726

2 API Description Encoder

This section describes the steps followed by the application to call the G.726 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 G726E 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

The application allocates memory for below mentioned structure.

```
/* Encoder parameter structure */
typedef struct
{
      sG726EMemAllocInfoType  sG726EMemInfo;
      G726_Void               * pvG726EEncodeInfoPtr;
      G726_U8                 * pu8APPEInitializedDataStart;
      G726_S32                s32APPEBitRate;
      G726_S32                s32APPEPcmFormat;
      G726_S32                s32APPESampleNum;
} sG726EEncoderConfigType;
```

Description of the encoder parameter structure *sG726EEncoderConfigType*

sG726EMemInfo

This is a memory information structure. The application needs to call the function eG726EQueryMem to get the memory requirements from encoder. The encoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

pvG726EEncodeInfoPtr

This is a void pointer. The encoder will initialize this pointer to a structure during the initialization routine. The structure contains the pointers to tables, buffers and symbols used by the encoder.

pu8APPEInitializedDataStart

The application has to assign this pointer with the symbol supplied in the header file. This symbol is the start address of the initialized data that the encoder uses. The encoder needs to know this as the OS can relocate the data tables of the G.726 encoder every time the application is invoked.

s32APPEBitRate

The application shall set the value of this variable to BIT_RATE_16KBPS, BIT_RATE_24KBPS, BIT_RATE_32KBPS or BIT_RATE_40KBPS corresponding to 16 kbps, 24 kbps, 32 kbps or 40 kbps respectively before invoking encoder function every time(Refer **Error! Reference source not found.** for definition of these constants).

s32APPEPcmFormat

The encoder can process linear PCM, 8-bit A-law and 8-bit μ-law input data. The application shall set the value of this variable to PCM_LINEAR, PCM_ALAW or PCM_ULAW before invoking encoder function every time(Refer **Error! Reference source not found.** for definition of these constants).

s32APPESampleNum

This value is used to identify number of input samples to be encoded. The application shall assign a value equal to number of samples to be encoded before invoking encoder function.

Example pseudo code for this step:

```
/* Allocate memory for the encoder parameter */
sG726EEncoderConfigType *psEncConfig;

/* Allocate fast memory for encoder config */
psEncConfig = (sG726EEncoderConfigType *)
                  alloc_fast (sizeof(sG726EEncoderConfigType));

/* Allocate memory for encoder to use */
psEncConfig->pvG726EencodeInfoPtr = NULL;

/* Fill up the relocated data position (not used) */
psEncConfig->pu8APPEInitializedDataStart = NULL;
```

Step 2: Get the encoder memory requirements

The G.726 encoder does not do any dynamic memory allocation. The application calls the function *eG726EqueryMem* to get the encoder memory requirements. This function must be called before any other encoder functions are invoked.

The function prototype of eG726EqueryMem is:

C prototype:

```
eG726EReturnType eG726EQueryMem (sG726EEncoderConfigType *psEncConfig);
```

Arguments:

- *psEncConfig* - Encoder config pointer.

Return value:

- G726E_OK - Memory query successful.
- Other codes - Error (For other error codes refer to appendix).

This function populates the memory information structure, which is described below:

```
/* Memory information structure array */
typedef struct
{
      /* Number of valid memory requests */
      G726_S32                      s32G726ENumMemReqs;
      sG726EMemAllocInfoSubType     asMemInfoSub[MAX_NUM_MEM_REQS];
} sG726EMemAllocInfoType;
```

Description of the structure *sG726EMemAllocInfoType*

s32G726ENumMemReqs

The number of memory chunks requested by the encoder.

asMemInfoSub

This structure contains each chunk's memory config parameters.

```
typedef struct
 {
      G726_S32    s32G726ESize;        /* Size in bytes */
      G726_S32    s32G726EType;        /* Memory type Fast or Slow */
      G726_S32    s32G726EMemTypeFs;   /* Static or scratch */
      G726_Void   * pvAPPEBasePtr;
            /* Pointer to the base memory, which will be allocated and
            filled by the application  */
      G726_U8     u8G726EMemPriority;
            /* priority in which memory needs to be allocated in fast
            memory */
}sG726DmemAllocInfoSubType;
```

Description of the structure *sG726DmemAllocInfoSubType*

s32G726ESize

The size of each chunk in bytes

s32G726EType

The memory description field indicates whether requested chunk of memory is static or scratch. Codec will update this flag to STATIC or SCRATCH based on whether the requested memory chunk is used as STATIC or as SCRATCH memory.

s32G726EMemTypeFs

The type of the memory indicates if the requested chunk of memory needs to be allocated in external or internal memory. The type of memory can be SLOW_MEMORY (external memory) or FAST_MEMORY (internal memory). In targets where there is no internal memory, the application can allocate memory in external memory.

(Note: If the encoder requests for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the encoder will still encode, but the performance (MHz) will suffer.)

pvAPPEBasePtr

This will be initialized by the application. The application will allocate the memory for each chunk depending on the requested size and the type, and then assign the base address of this chunk of memory to *pvAPPEBasePtr*. The application should allocate the memory that is aligned to a 4-byte boundary in any case.

u8G726EMemPriority

This indicates the priority level of the memory type. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In case the type of memory is FAST_MEMORY then the field u8G726EMemPriority indicates the importance or the priority of the request. A priority value of zero indicates highest priority and 255 indicates lowest priority.

Example pseudo code for the memory information request

```
/* Query for memory */

eRetVal = eG726EQueryMem(psEncConfig);

if (eRetVal != G726E_OK)
      return G726_FAILURE;
```

Step 3: Allocate Data Memory for the encoder

In this step the application allocates the memory as required by the G.726 encoder and fills up the base memory pointer *'pvAPPEBasePtr' of 'sG726EMemAllocInfoSubType'* structure for each chunk of memory requested by the encoder.

Example pseudo code for the memory allocation and filling the base memory pointer by the application is given below.

```
sG726EMemAllocInfoSubType *psMem;

/* Number of memory chunks requested by the encoder */
s16NumMemReqs = psEncConfig-> sG726EMemInfo.s32G726ENumMemReqs;

for(s16i = 0; s16i < s16NumMemReqs; s16i++)
{
      psMem = &( psEncConfig-> sG726EMemInfo.asMemInfoSub[s16i]);
      if (psMem->s32G726EMemTypeFs == FAST_MEMORY)
      {
            /* If application does not have enough memory to allocate
               in fast memory, it can check priorty
               of requested chunk (psMem->u8G726EMemPriority) and
               allocate accordingly */
            /* This function allocates memory in internal memory */
            psMem-> pvAPPEBasePtr = alloc_fast(psMem->s32G726ESize);
      }
      else
      {
            This function allocates memory in external memory */
            psMem-> pvAPPEBasePtr = alloc_slow(psMem->s32G726ESize);
      }
}
```

The functions alloc_fast and alloc_slow are required to allocate 4-byte aligned fast and slow memory.

Step 4: Initialization routine

All initializations required for the encoder are done in g726e_*encode_init*. This function must be called before the main encode function is called.

C prototype:

```
eG726EReturnType eG726EEncodeInit(
                    sG726EEncoderConfigType *psEncConfig);
```

Arguments:

- Pointer to encoder configuration structure

Return value:

- G726E_OK - Initialization successful.
- Other codes - Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the G726 encoder. */
eRetVal = eG726EEncodeInit(psEncConfig);

if (eRetVal != G726E_OK)
      return G726_FAILURE;
```

Step 5: Memory allocation for input buffer

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 */
ps16InBuf = alloc_fast( L_BUFFER * sizeof(G726_S16));
```

Step 6: Memory allocation for output buffer

The application has to allocate (2-byte aligned) memory for the output buffers to hold the encoded bitstream corresponding to input PCM sample. The pointer to this output buffer needs to be passed to the g726e_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.

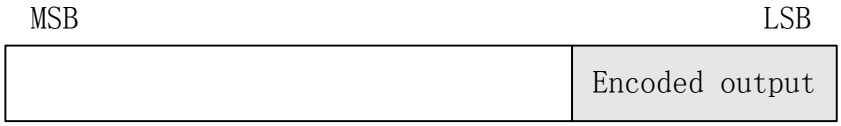
Example pseudo code for allocating memory for output buffer

```
/* Allocate memory for output buffer */
```

```
ps16OutBuf = alloc_fast (L_BUFFER * sizeof(G726_S16));
```

Special Consideration

Encoder output can be 5, 4, 3 or 2 bits per word corresponding to bit rate of 40, 32, 24 and 16 kbps respectively and placed at least significant bits position of the word with zero extension as shown below.



Step 7: Call the encode routine

The main G.726 encoder function is eG726EEncode. This function encodes input PCM sample and writes bitstream to output buffer.

C prototype:

```
eG726EReturnType eG726EEncode(
                            sG726EEncoderConfigType *psEncConfig,
                            G726_S16 *ps16InBuf,
                            G726_S16 *ps16OutBuf);
```

Arguments:

- psEncConfig Pointer to encoder config structure
- ps16InBuf Pointer to input speech buffer
- ps16OutBuf Pointer to output (encoded) buffer

Return value:

- G726E_OK indicates encoding was successful.
- **Others** **indicates error**

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

```
while (G726_TRUE)
{
      /* Initialize the input sample numbers to be encoded */
      psEncConfig->s32APPESampleNum = 64;
      /* Initialize the rate field. */
      psEncConfig->s32APPEBitRate = BIT_RATE_24KBPS;
      /* Encode format */
      psEncConfig->s32APPEPcmFormat = PCM_ALAW;

      eRetVal= eG726EEncode(psEncConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != G726E_OK)
            return G726_FAILURE;
}
```

Step 8: Free memory

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

```
free(ps16OutBuf);
free(ps16InBuf);
for(s16i = 0; s16i < s16NumMemReqs; s16i++)
{
    free(psEncConfig->sG726EMemInfo.asMemInfoSub[s16i].pvAPPEBasePtr);
}
free (psEncConfig);
```

3 API Description Decoder

This section describes the steps followed by the application to call the G.726 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 G726D 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

The application allocates memory for below mentioned structure.

```
/* Decoder parameter structure */
typedef struct
 {
      sG726DMemAllocInfoType  sG726DMemInfo;
      G726_Void               * pvG726DDecodeInfoPtr ;
      G726_U8                 * pu8APPDInitializedDataStart;
      G726_S32                s32APPDBitRate;
      G726_S32                s32APPDPcmFormat;
      G726_S32                s32APPDSampleNum;
} sG726DDecoderConfigType;
```

Description of the decoder parameter structure *sG726DDecoderConfigTyp*

sG726DMemInfo

This is a memory information structure. The application needs to call the function eG726DqueryMem to get the memory requirements from decoder. The decoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

pvG726DDecodeInfoPtr

This is a void pointer. The decoder will initialize this pointer to a structure during the initialization routine. This structure contains the pointers to tables, buffers and symbols used by the decoder.

pu8APPDInitializedDataStart

The application has to assign this pointer with the symbol supplied in the header file. This symbol is the start address of the initialized data that the decoder uses. The decoder needs to know this as the OS can relocate the data tables of the G726 decoder every time the application is invoked.

s32APPDBitRate

The application shall set the value of this variable to BIT_RATE_16KBPS, BIT_RATE_24KBPS, BIT_RATE_32KBPS or BIT_RATE_40KBPS corresponding to 16

kbps, 24 kbps, 32 kbps or 40 kbps respectively before invoking decoder function every time (Refer **Error! Reference source not found.** for definition of these constants).

s32APPDPcmFormat

The encoder can generate linear PCM, 8-bit A-law and 8-bit μ-law output data. The application shall set the value of this variable to PCM_LINEAR, PCM_ALAW or PCM_ULAW before invoking decoder function every time (Refer Appendix for definition of these constants).

s32APPDSampleNum

This value is used to identify number of input samples to be decoded. The application shall assign a value to this member before invoking decoder function every time.

Example pseudo code for this step:

```
/* Allocate memory for the decoder parameter */
sG726DdecoderConfigType *psDecConfig;
psDecConfig = (sG726DdecoderConfigType *)
                              alloc(sizeof(sG726DDecoderConfigType);
/* Allocate memory for decoder to use */
psDecConfig->pvG726DdecodeInfoPtr = NULL;
/* Fill up the Relocated data position (not used) */
psDecConfig->pu8APPDInitializedDataStart = NULL;
```

Step 2: Get the decoder memory requirements

The G.726 decoder does not do any dynamic memory allocation. The application calls the function *eG726DqueryMem* to get the decoder memory requirements. This function must be called before any other decoder functions are invoked.

The function prototype of *eG726DqueryMem* is:

C prototype

eG726DReturnType eG726DQueryMem(
*sG726DDecoderConfigType *psDecConfig);*

Arguments

- psDecConfig - Decoder configuration pointer.

Return value

- G726D_OK - Memory query successful.
- Other codes - Error (For other error codes refer to appendix).

This function populates the memory information structure, which is described below:

Memory information structure array

```
typedef struct
{
      /* Number of valid memory requests */
      G726_S32                      s32G726DNumMemReqs;
      sG726DMemAllocInfoSubType     asMemInfoSub[MAX_NUM_MEM_REQS];
```

```
} sG726DMemAllocInfoType;
```

Description of the structure *sG726DMemAllocInfoType*

s32G726DNumMemReqs

The number of memory chunks requested by the decoder.

asMemInfoSub

This structure contains each chunk's memory configuration parameters.

```
typedef struct
{
      G726_S32    s32G726DSize;           /* Size in bytes */
      G726_S32    s32G726DType;          /* Static or scratch */
      G726_S32    s32G726DMemTypeFs;  /* Memory type Fast or Slow */
      G726_Void   *pvAPPDBasePtr;
            /* Pointer to the base memory, which will  be  allocated and
           filled by the application */
      G726_U8     u8G726DMemPriority;
           /* priority in which memory needs to be allocated in fast
           memory */
}sG726DmemAllocInfoSubType;
```

Description of the structure *G726D_Mem_Alloc_Info_sub*

s32G726DSize

The size of each chunk in bytes.

s32G726DType

The memory description field indicates whether requested chunk of memory is static or scratch. Codec will update this flag to STATIC or SCRATCH based on whether the requested memory chunk is used as STATIC or as SCRATCH memory.

s32G726DMemTypeFs

The type of the memory indicates if the requested chunk of memory needs to be allocated in external or internal memory. The type of memory can be SLOW_MEMORY (external memory) or FAST_MEMORY (internal memory). In targets where there is no internal memory, the application can allocate memory in external memory.
(Note: If the decoder requests for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the decoder will still decode, but the performance (MHz) will suffer.)

pvAPPDBasePtr

This will be initialized by the application. The application will allocate the memory for each chunk depending on the requested size and the type, and then assign the base address of this chunk of memory to *appd_base_ptr.* The application should allocate the memory that is aligned to a 4-byte boundary in any case.

u8G726DMemPriority

This indicates the priority level of the memory type. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In case the type of memory is FAST_MEMORY then the field u8G726DMemPriority indicates the importance or the priority of the request. A priority value of zero indicates highest priority and 255 indicates lowest priority.

Example pseudo code for the memory information request

```
/* Query for memory */
eRetVal = eG726DQueryMem(psDecConfig);
```



```
if (eRetVal != G726D_OK)
      return G726_FAILURE;
```

Step 3: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by the G.726 decoder and fills up the base memory pointer '*appd_base_ptr*' of '*G726D_Mem_Alloc_Info_sub*' structure for each chunk of memory requested by the decoder.

Example pseudo code for the memory allocation and filling the base memory pointer by the application is given below.

```
sG726DMemAllocInfoSubType *psMem;

/* Number of memory chunks requested by the decoder */
s16NumMemReqs = psDecConfig->sG726DMemInfo.s32G726DNumMemReqs;

for(s16i = 0; s16i < s16NumMemReqs; s16i++)
{
      psMem = &(psDecConfig->sG726DMemInfo.asMemInfoSub[s16i]);
      if (psMem->s32G726DMemTypeFs == FAST_MEMORY)
      {
            /* If application does not have enough memory to allocate
               in fast memory, it can check priorty
               of requested chunk (psMem-> u8AMRDMemPriority) and
               allocate accordingly */
            /* This function allocates memory in internal memory */
            psMem->pvAPPDBasePtr = alloc_fast(psMem->s32G726DSize);
      }
      else
      {
            This function allocates memory in external memory */
            psMem->pvAPPDBasePtr = alloc_slow(psMem->s32G726DSize);
      }
}
```

The functions alloc_fast and alloc_slow are required to allocate the memory aligned to 4-byte boundary.

Step 4: Initialization routine

All initializations required for the decoder are done in *eG726DDecodeInit*. This function must be called before the main decode function is called.

C prototype:

```
eG726DReturnType eG726DDecodeInit(
            sG726DDecoderConfigType *psDecConfig);
```

Arguments:

- *psDecConfig* Pointer to decoder configuration structure

Return value:

- G726D_OK - Initialization successful.
- Other codes - Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the G.726 decoder. */
eRetVal = eG726DDecodeInit(psDecConfig);
if (eRetVal != G726D_OK)
      return G726_FAILURE;
```

Step 5: Memory allocation for input buffer

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 decoder. Pointer to the input buffer needs to be passed to decode routine.

Example pseudo code for allocating the input buffer

```
/* Allocate memory for input buffer */
ps16InBuf = alloc_fast(L_BUFFER * sizeof (G726_S16));
```

Step 6: Memory allocation for output buffer

The application has to allocate (2-byte aligned) memory for the output buffers to hold the decoded PCM sample corresponding to G.726 encoded bitstream. The pointer to this output buffer needs to be passed to the g726d_decode 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 decoder marginally. It would be desirable to allocate the buffer in the slow memory.

Example pseudo code for allocating memory for output buffer

```
/* Allocate memory for output buffer */
ps16OutBuf = alloc_fast (L_BUFFER * sizeof (G726_S16));
```

Special Consideration

Input of decoder is 5, 4, 3 or 2 bits per word corresponding to bit rate of 40, 32, 24 and 16 kbps respectively and decoded output can be linear PCM, 8-bit A-law or 8-bit μ-law data.

Step 7: Call the decode routine

The main G.726 decoder function is eG726DDecode. This function decodes the G.726 bitstream and writes bitstream to output buffer.

C prototype:

```
eG726DReturnType eG726DDecode(
                        sG726DDecoderConfigType *psDecConfig,
                        G726_S16 *ps16InBuf,
                        G726_S16 *ps16OutBuf);
```

Arguments:

- psDecConfig Pointer to decoder configuration structure
- ps16InBuf Pointer to G.726 bitstream buffer
- ps16OutBuf Pointer to output (decoded) buffer

Return value:

- G726D_OK indicates decoding was successful.
- **Others** **indicates error**

Example pseudo codes for calling the main decode routine of the decoder.

```
while (G726_TRUE)
{
      /* Initialize the input sample number to be decoded */
      psDecConfig->s32APPDSampleNum = 64;
      /* Initialize the rate field. */
      psDecConfig->s32APPDBitRate = BIT_RATE_24KBPS;
      /* Decode format */
      psDecConfig->s32APPDPcmFormat = PCM_ALAW;

      eRetVal = eG726DDecode(psDecConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != G726D_OK)
            return G726_FAILURE;
}
```

Step 8: Free memory

The application should release memory before exiting the decoder.

```
free (ps16OutBuf);
free (ps16InBuf);
for (s16i=0; s16i<s16NumMemReqs; s16i++)
{
    free(psDecConfig->sG726DMemInfo.asMemInfoSub[s16i].pvAPPDBasePtr);
}
free (psDecConfig);
```

4 Example calling Routine

4.1 Example calling routine for G.726 Encoder

Below example code gives guidelines for calling G.726 encoder.

```
/*****************************************************************
 *                          INCLUDE FILES
 *****************************************************************/

#include "g726_enc_api.h"

/*****************************************************************
 *        MAIN PROGRAM
 *****************************************************************/

G726_S32 main(G726_S32 s32Argc, G726_S8 *ps8Argv[])
{
      eG726EReturnType eRetVal;
      /* Pointer to new speech data*/
      G726_S16 *ps16InBuf;
      /* Output bitstream buffer */
      G726_S16 *ps16OutBuf;
      G726_S16  s16i;
      G726_S16  s16NumMemReqs;
      SG726EMemAllocInfoSubType*psMem;
      SG726EEncoderConfigType *psEncConfig;

      /* Allocate memory for encoder configuration structure */
      psEncConfig = (SG726EEncoderConfigType *)
                        alloc_fast(sizeof(SG726EEncoderConfigType));

      /* allocate memory for encoder to use */
      psEncConfig->pvG726EEncodeInfoPtr = NULL;
      /* Not Use */
      psEncConfig->pu8APPEInitializedDataStart = NULL;

      /* Query for memory */
      eRetVal = eG726EQueryMem(psEncConfig);
      if (eRetVal != G726E_OK)
      {
            /* Deallocate memory allocated for encoder config */
            free(psEncConfig);
            return G726_FAILURE;
      }
      /* Number of memory chunk requests by the encoder */
      s16NumMemReqs = psEncConfig->sG726EMemInfo.s32G726ENumMemReqs;

      /* Allocate memory requested by the encoder*/
```

```
for(s16i = 0; s16i <s16NumMemReqs; s16i++)
{
      psMem = &(psEncConfig->sG726EMemInfo.asMemInfoSub[s16i]);
      if (psMem-> s32G726EMemTypeFs == FAST_MEMORY)
      {
            /* Check for priority and memory description can be
               added here */
            psMem->pvAPPEBasePtr=alloc_fast(psMem->s32G726ESize);
      }
      else
      {
            psMem->pvAPPEBasePtr=alloc_slow(psMem->s32G726ESize);
      }
}

/* Initialize the G726 encoder */
eRetVal = eG726EEncodeInit (psEncConfig);
if (eRetVal != G726E_OK)
{
      /* Free all the memory allocated for encoder
            config param */
      for(s16i = 0; s16i < s16NumMemReqs; s16i++)
      {
            free(psEncConfig->sG726EMemInfo.\
                  asMemInfoSub[s16i].pvAPPEBasePtr);
      }
      free (psEncConfig);
      return G726_FAILURE;
}
/* Allocate memory for input buffer */
if ((ps16InBuf = alloc_fast (L_BUFFER *\
                              sizeof (G726_S16))) == NULL)
{
      /*Free all the memory allocated for encoder config param*/
      for(s16i = 0; s16i < s16NumMemReqs; s16i++)
      {
            free(psEncConfig->sG726EMemInfo.\
                  asMemInfoSub[s16i].pvAPPEBasePtr);
      }
      free (psEncConfig);
      return G726_FAILURE;
}
if ( (ps16OutBuf = alloc_fast(L_BUFFER *\
                              sizeof(G726_S16))) ==   NULL)
{
      /* Free all the memory allocated for encoder config param
      */
      for(s16i = 0; s16i < s16NumMemReqs; s16i++)
      {
            free(psEncConfig->sG726EMemInfo.\
                  asMemInfoSub[s16i].pvAPPEBasePtr);
      }
      free (psEncConfig);
```

```
            free(ps16InBuf);
            return G726_FAILURE;
      }
      while (G726_TRUE)
      {
            if(exist input sample)
                  Get sample from input buffer;
            else
                  break;

            /* Initialize the input sample numbers to be encoded */
            psEncConfig->s32APPESampleNum = 64;
            /* Initialize the rate field. */
            psEncConfig->s32APPEBitRate = BIT_RATE_24KBPS;
            /* Encode format */
            psEncConfig->s32APPEPcmFormat = PCM_ALAW;

            eRetVal= eG726EEncode(psEncConfig, ps16InBuf, ps16OutBuf);
            if (retval != G726E_OK)
            {
                  free (ps16OutBuf);
                  free (ps16InBuf);
                  for(s16i = 0; s16i < s16NumMemReqs; s16i++)
                  {
                        free(psEncConfig->sG726EMemInfo.\
                              asMemInfoSub[s16i].pvAPPEBasePtr);
                  }
                  free (psEncConfig);
                  return G726_FAILURE;
            }
      }

      /*************************************************************
       *Closedown speech coder
       *************************************************************/
      free(ps16OutBuf);
      free(ps16InBuf);
      for(s16i = 0; s16i < s16NumMemReqs; s16i++)
      {
            free(psEncConfig->sG726EMemInfo.\
                        asMemInfoSub[s16i].pvAPPEBasePtr);
      }
      free (psEncConfig);
      return G726_SUCCESS;
}
```

4.2 Example calling routine for G.726 Decoder

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

```
/*************************************************************************
* Include Files
*************************************************************************/
#include "g726_dec_api.h"

/*************************************************************************
* Main Program
*************************************************************************/
G726_S32 main(G726_S32 s32Argc, G726_S8 *ps8Argv[])
{
      eG726DReturnType eRetVal;
      /* Pointer to bitstream buffer */
      G726_S16 *ps16InBuf;
      /* decoded PCM sample buffer */
      G726_S16 *ps16OutBuf;
      G726_S16  s16i;
      G726_S16  s16NumMemReqs;
      SG726DMemAllocInfoSubType*psMem;
      SG726DDecoderConfigType *psDecConfig;

      /* Allocate memory for decoder configuration structure */
      psDecConfig =( sG726DDecoderConfigType *)
                        alloc_fast(sizeof(sG726DDecoderConfigType));

      /* allocate memory for decoder to use */
      psDecConfig->pvG726DDecodeInfoPtr = NULL;
      /* Not Use */
      psDecConfig->pu8APPDInitializedDataStart = NULL;

      /* Query decoder for its memory requirement */
      eRetVal = eG726DQueryMem(psDecConfig);
      if (eRetVal != G726D_OK)
      {
            /* De-allocate memory allocated to psDecConfig */
            free(psDecConfig);
            return G726_FAILURE;
      }

      /* Number of memory chunk requested by the decoder */
      s16NumMemReqs = psDecConfig->sG726DMemInfo.s32G726DNumMemReqs;

      /* allocate memory requested by the decoder*/
      for(s16i = 0; s16i <s16NumMemReqs; s16i++)
      {
            psMem = &(psDecConfig->sG726DMemInfo.asMemInfoSub[s16i]);
            if (psMem-> s32G726DMemTypeFs == FAST_MEMORY)
            {
                  /* If needed check for priority and memory
```

```
                description can be added here */
            psMem->pvAPPDBasePtr=alloc_fast(psMem->s32G726DSize);
      }
      else
      {
            psMem->pvAPPDBasePtr=alloc_slow(psMem->s32G726DSize);
      }
}

/* Initialize decoder */
eRetVal = eG726DDecodeInit(psDecConfig);
if (eRetVal =! G726D_OK)
{
      for (s16i=0; s16i<s16NumMemReqs; s16i++)
      {
            free(psDecConfig->sG726DMemInfo.\
                  asMemInfoSub[s16i].pvAPPDBasePtr);
      }
      free (psDecConfig);
      return G726_FAILURE;
}
/* Allocate memory for input buffer */
in_buf = alloc_fast(L_BUFFER * sizeof (G726_S16));

/* allocate memory for output buffer */
out_buf = alloc_fast(L_BUFFER * sizeof (G726_S16));

while(G726_TRUE)
{
      if(exist input sample)
            Get sample from input buffer;
      else
            break;

      /* Initialize the input sample number to be decoded */
      psDecConfig->s32APPDSampleNum = 64;
      /* Initialize the rate field. */
      psDecConfig->s32APPDBitRate = BIT_RATE_24KBPS;
      /* Decode format */
      psDecConfig->s32APPDPcmFormat = PCM_ALAW;

      /* Decode speech */
      eRetVal = eG726DDecode(psDecConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != G726D_OK)
      {
            free(in_buf);
            free(out_buf);
            for (s16i=0; s16i<s16NumMemReqs; s16i++)
            {
                  free(psDecConfig->sG726DMemInfo.\
                        asMemInfoSub[s16i].pvAPPDBasePtr);
            }
            free (psDecConfig);
            return G726_FAILURE;
```

```
            }
        }
        /*************************************************************
         * Closedown speech coder
         *************************************************************/
        free (ps16OutBuf);
        free (ps16InBuf);
        for (s16i=0; s16i<s16NumMemReqs; s16i++)
        {
              free(psDecConfig->sG726DMemInfo.\
                    asMemInfoSub[s16i].pvAPPDBasePtr);
        }
        free (psDecConfig);
        return G726_SUCCESS;
}
```