08-6969-API-ZCH66
OCT 17, 2008
1.0

Application Programmers Interface for G.729AB Decoder and Encoder

ABSTRACT:

Application Programmers Interface for G.729AB Decoder and Encoder

KEYWORDS:

Multimedia codecs, speech, G.729AB

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	14-Jul-2008	Bing Song	Initial Draft
1.0	17-Oct-08	Qiu Cunshou	Add *s8G729VersionInfo* API

Table of Contents

Introduction

1.1 Purpose

This document gives the details of the Application Programming Interface (API) of G.729AB encoder and decoder. ITU-T G.729AB specifies a coded representation that can be used for compressing the speech of multimedia services at a very low bit rate (8 kbps).

The G.729AB 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.729AB 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.729AB codec. The intended audience for this document is the development community who wish to use the G.729AB codec in their systems.

1.4 References

1.4.1 Standards

- **ITU-T Recommendation G.729 (01/2007)** – Coding of speech at 8 kbit/s using conjugate-structure algebraic-code-excited linear prediction.

1.4.2Freescale Multimedia References

- G.729AB Codec Application Programming Interface – g729_codec_api.doc
- G.729AB Codec Requirements Book – g729_codec_reqb.doc
- G.729AB Codec Test Plan - g729_codec_test_plan.doc
- G.729AB Codec Release notes - g729_codec_release_notes.doc
- G.729AB Codec Test Results – g729_codec_test_results.doc
- G.729AB Codec Performance Results – g729_codec_perf_results.doc
- G.729AB Interface Common Header – g729_com_api.h
- G.729AB Interface Decoder Header – g729_dec_api.h
- G.729AB Interface Encoder Header – g729_enc_api.h
- G.729AB Decoder Application Code – g729_dectest.c
- G.729AB Encoder Application Code – g729_enctest.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.729ab

.

2 API Description of Encoder

This section describes the steps followed by the application to call the G.729AB during each step the data structures and the functions used will be explained. Pseudo code is given at the end of each step.
The G.729AB encoder API currently support PUSH mode. It is the application's duty to supply correct data to the encoder.

2.1 Encoder API Data Types

The member variables inside the structure are prefixed as G729E 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: Print version information

The application can get version information such as version number and build time by calling the below function.

C prototype:
```
G729_S8 * s8G729VersionInfo(void);
```

Arguments:
- None.

Return Value:
- Returns a sting which includes version information

Step 2: Allocate memory for Encoder configuration parameter structure

The application allocates memory for below mentioned structure.
```
/* Encoder parameter structure */
typedef struct
{
      sG729EMemAllocInfoType     sG729EMemInfo;
      G729_Void                 *pvG729EEncodeInfoPtr;
      G729_U8                   u8APPEVADFlag;
} sG729EEncoderConfigType;
```

Description of the encoder parameter structure *sG729EEncoderConfigType*
sG729EMemInfo

This is a memory information structure. The application needs to call the function eG729EQueryMem to get the memory requirements from encoder. The encoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

pvG729EEncodeInfoPtr

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.

u8APPEVADFlag

The application shall set the value of this variable to E_G729E_VAD_DISABLE or E_G729E_VAD_ENABLE before invoking encoder function every time (Refer Appendix for definition of these constants).

Example pseudo code for this step

```
/* Allocate memory for the encoder parameter */
sG729EEncoderConfigType *psEncConfig;

/* Allocate fast memory for encoder config */
psEncConfig = (sG729EEncoderConfigType *)
                  alloc_fast(sizeof(sG729EEncoderConfigType));


/* Allocate memory for encoder to use */
psEncConfig->pvG729EEncodeInfoPtr = NULL;
```

Step 3: Get the encoder memory requirements

The G.729AB encoder does not do any dynamic memory allocation inside the library. The application calls the function *eG729EQueryMem* to get the encoder memory requirements. This function must be called before any other encoder functions are invoked.

The function prototype of eG729EQueryMem is:

C prototype:

```
eG729EReturnType eG729EQueryMem (sG729EEncoderConfigType *psEncConfig);
```

Arguments:

• *psEncConfig*	-	Encoder config pointer.

Return value:

• E_G729E_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 */
      G729_S32                     s32G729ENumMemReqs;
      sG729EMemAllocInfoSubType    asMemInfoSub[G729_MAX_NUM_MEM_REQS];
} sG729EMemAllocInfoType;
```

Description of the structure *G729EMemAllocInfoType*

s32G729ENumMemReqs

The number of memory chunks requested by the encoder.

asMemInfoSub

This structure contains configuration parameters of each memory chunk.

```
typedef struct
 {
      G729_S32    s32G729ESize;          /* Size in bytes */
      G729_U8     u8G729EType;           /* Static or scratch */
      G729_U8     u8G729EMemTypeFs;    /* Fast or Slow memory */
      G729_Void   * pvAPPEBasePtr;
            /* Pointer to the base memory, which will be allocated and
            filled by the application  */
      G729_U8     u8G729EMemPriority;
            /* priority in which memory needs to be allocated in fast
            memory */
} sG729EMemAllocInfoSubType;
```

Description of the structure *sG729EMemAllocInfoSubType*

s32G729ESize

The size of each chunk in bytes

u8G729EType

The memory description field indicates whether requested chunk of memory is static or scratch. Codec will update this flag to G729_STATIC or G729_SCRATCH based on whether the requested memory chunk is used as G729_STATIC or as G729_SCRATCH memory.

u8G729EMemTypeFs

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 G729_SLOW_MEMORY (external memory) or G729_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 G729_FAST_MEMORY for which the application allocates a G729_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.

u8G729EMemPriority

This indicates the priority level of the memory type. The type of memory can be G729_SLOW_MEMORY or external memory, G729_FAST_MEMORY or internal memory. In case the type of memory is G729_FAST_MEMORY then the field u8G729EMemPriority 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 = eG729EQueryMem(psEncConfig);

if (eRetVal != E_G729E_OK)
      return G729_FAILURE;
```

Step 4: Allocate Data Memory for the encoder

In this step the application allocates the memory as required by the G.729AB encoder and fills up the base memory pointer *'pvAPPEBasePtr' of 'sG729EMemAllocInfoSubType'* 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.

```
sG729EMemAllocInfoSubType *psMem;

/* Number of memory chunks requested by the encoder */
s32NumMemReqs = psEncConfig->sG729EMemInfo.s32G729ENumMemReqs;

for(s32i = 0; s32i < s32NumMemReqs; s32i++)
{
      psMem = &( psEncConfig->sG729EMemInfo.asMemInfoSub[s32i]);
      if (psMem->s32G729EMemTypeFs == FAST_MEMORY)
      {
            /* If application does not have enough memory to allocate
               in fast memory, it can check priorty
               of requested chunk (psMem->u8G729EMemPriority) and
               allocate accordingly */
            /* This function allocates memory in internal memory */
            psMem->pvAPPEBasePtr = alloc_fast(psMem->s32G729ESize);
      }
      else
      {
            This function allocates memory in external memory */
            psMem->pvAPPEBasePtr = alloc_slow(psMem->s32G729ESize);
      }
}
```

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

Step 5: Initialization routine

All initializations required for the encoder are done in *eG729EEncodeInit*. This function must be called before the main encode function is called.

C prototype

```
eG729EReturnType eG729EEncodeInit (sG729EEncoderConfigType *psEncConfig);
```

Arguments

- Pointer to encoder configuration structure

Return value

• E_G729E_OK	- Initialization successful.
• Other codes	- Initialization Error

<u>Example pseudo code for calling the initialization routine of the decoder</u>

```
/* Initialize the G729 encoder. */
eRetVal = eG729EEncodeInit(psEncConfig);

if (eRetVal != E_G729E_OK)
      return G729_FAILURE;
```

Step 6: Memory allocation for input buffer

The application has to allocate (aligned to 4-byte boundary) memory needed for the input buffer. It is desirable to have the input buffer allocated in G729_FAST_MEMORY, as this may improve the performance (MHz) of the encoder. The size of input buffer should be equal to speech frame size i.e. 80 words. Pointer to the input buffer needs to be passed to encode frame routine.

<u>Example pseudo code for allocating the input buffer</u>

```
/* Allocate memory for input buffer G729_L_FRAME = 80 */
ps16InBuf = alloc_fast(G729_L_FRAME * sizeof(G729_S16));
```

Step 7: Memory allocation for output buffer

The application has to allocate (aligned to 2-byte boundary) the memory for the output buffers to hold the encoded bitstream corresponding to one frame of speech sample. The pointer to this output buffer needs to be passed to the eG729EEncodeFrame 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.

<u>Example pseudo code for allocating memory for output buffer</u>

```
/* Allocate memory for output buffer G729_CODEC_SIZE = 82 */
ps16OutBuf = alloc_fast((G729_CODEC_SIZE) * sizeof(G729_S16));
```

Step 8: Call the encode routine

The main G.729AB encoder function is eG729EEncode. This function encodes input sample and writes packed bitstream to output buffer.

C prototype:

```
eG729EReturnType eG729EEncodeFrame (
                                    sG729EEncoderConfigType *psEncConfig,
                                    G729_S16 *ps16InBuf,
                                    G729_S16 *ps16OutBuf);
```

Arguments:

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

Return value:

• E_G729E_OK	indicates encoding was successful.
• **Others**	**indicates error**

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

```
while (Not end of file)
{
      psEncConfig->u8APPEVADFlag = E_G729E_VAD_DISABLE;

      eRetVal= eG729EEncodeFrame(psEncConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != E_G729E_OK)
            return G729_FAILURE;
}
```

Step 9: Free memory

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

```
free(ps16OutBuf);
free(ps16InBuf);
for(s32i = 0; s32i < s32NumMemReqs; s32i++)
{
    free(psEncConfig->sG729EMemInfo.asMemInfoSub[s32i].pvAPPEBasePtr);
}
free (psEncConfig);
```

3 API Description of Decoder

This section describes the steps followed by the application to call the G.729AB. During each step the data structures and the functions used will be explained. Pseudo code is given at the end of each step.
The G.729AB decoder API currently support PUSH mode. It is the application's duty to supply correct data to the decoder.

3.1 Decoder API Data Types

The member variables inside the structure are prefixed as G729D 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: Print version information

The application can get version information such as version number and build time by calling the below function.

C prototype:
```
G729_S8 * s8G729VersionInfo(void);
```

Arguments:
- None.

Return Value:
- Returns a sting which includes version information

Step 2: Allocate memory for Decoder config parameter structure

The application allocates memory for below mentioned structure.

```
/* Decoder parameter structure */
typedef struct
 {
      sG729DMemAllocInfoType     sG729DMemInfo;
      G729_Void                * pvG729DDecodeInfoPtr ;
      G729_U8                   u8APPDFrameErasureFlag;
}sG729DDecoderConfigType;
```

Description of the decoder parameter structure *sG729DDecoderConfigType*

sG729DMemInfo

This is a memory information structure. The application needs to call the function eG729DQueryMem to get the memory requirements from decoder. The decoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

pvG729DDecodeInfoPtr

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.

u8APPDFrameErasureFlag

This application needs to set this flag to E_G729D_FR_ERASED or E_G729D_FR_NOTERASED every time before invoking decode function every time

Example pseudo code for this step:

```
/* Allocate memory for the decoder parameter */
sG729DDecoderConfigType *psDecConfig;
psDecConfig = (sG729DDecoderConfigType *)
                              alloc(sizeof(sG729DDecoderConfigType);
/* Allocate memory for decoder to use */
psDecConfig->pvG729DdecodeInfoPtr = NULL;
```

Step 3: Get the decoder memory requirements

The G.729AB decoder does not do any dynamic memory allocation inside the library. The application calls the function *eG729DQueryMem* to get the decoder memory requirements. This function must be called before any other decoder functions are invoked.

The function prototype of *eG729DQueryMem* is:

C prototype

*eG729DReturnType eG729DQueryMem (sG729DDecoderConfigType *psDecConfig);*

Arguments

• psDecConfig	-	Decoder configuration pointer.

Return value

• E_G729D_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 */
      G729_S32                    s32G729DNumMemReqs;
      sG729DMemAllocInfoSubType   asMemInfoSub[G729_MAX_NUM_MEM_REQS];
} sG729DMemAllocInfoType;
```

Description of the structure `sG729DMemAllocInfoType`

s32G729DNumMemReqs

The number of memory chunks requested by the decoder.

asMemInfoSub

This structure contains each description of each memory chunk.

```
typedef struct
{
      G729_S32    s32G729DSize;           /* Size in bytes */
      G729_U8     u8G729DType;           /* Static or scratch */
      G729_U8     u8G729DMemTypeFs;   /* Memory type Fast or Slow */
      G729_Void   *pvAPPDBasePtr;
            /* Pointer to the base memory, which will be  allocated and
            filled
            by the application */
      G729_U8     u8G729DMemPriority;
            /* priority in which memory needs to be allocated in fast
            memory */
} sG729DMemAllocInfoSubType;
```

Description of the structure *G729D_Mem_Alloc_Info_sub*

s32G729DSize

The size of each chunk in bytes.

u8G729DType

The memory description field indicates whether requested chunk of memory is static or scratch. Codec will update this flag to G729_STATIC or G729_SCRATCH based on whether the requested memory chunk is used as G729_STATIC or as G729_SCRATCH memory.

u8G729DMemTypeFs

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 G729_SLOW_MEMORY (external memory) or G729_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 G729_FAST_MEMORY for which the application allocates a G729_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 pvAPPDBasePtr. The application should allocate the memory that is aligned to a 4-byte boundary in any case.

u8G729DMemPriority

This indicates the priority level of the memory type. The type of memory can be G729_SLOW_MEMORY or external memory, G729_FAST_MEMORY or internal memory. In case the type of memory is G729_FAST_MEMORY then the field u8G729DMemPriority 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 = eG729DQueryMem(psDecConfig);

if (eRetVal != E_G729D_OK)
      return G729_FAILURE;
```

Step 4: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by the G.729AB decoder and fills up the base memory pointer *'pvAPPDBasePtr'* of '*sG729DMemAllocInfoSubType*' structure for each chunk of memory requested by the decoder.

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

```
sG729DMemAllocInfoSubType *psMem;

/* Number of memory chunks requested by the decoder */
s32NumMemReqs = psDecConfig->sG729DMemInfo.s32G729DNumMemReqs;

for(s32i = 0; s32i < s32NumMemReqs; s32i++)
{
      psMem = &(psDecConfig->sG729DMemInfo.asMemInfoSub[s32i]);
      if (psMem->s32G729DMemTypeFs == G729_FAST_MEMORY)
      {
            /* If application does not have enough memory to allocate
               in fast memory, it can check priorty
               of requested chunk (psMem->u8G729DMemPriority) and
               allocate accordingly */
            /* This function allocates memory in internal memory */
            psMem->pvAPPDBasePtr = alloc_fast(psMem->s32G729DSize);
      }
      else
      {
            This function allocates memory in external memory */
            psMem->pvAPPDBasePtr = alloc_slow(psMem->s32G729DSize);
      }
}
```

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

Step 5: Initialization routine

All initializations required for the decoder are done in *eG729DDecodeInit*. This function must be called before the main decode function is called.

C prototype:
```
eG729DReturnType eG729DDecodeInit (sG729DDecoderConfigType *psDecConfig);
```

Arguments:

• *psDecConfig*	Pointer to decoder configuration structure

Return value:

• E_G729D_OK	-	Initialization successful.
• Other codes	-	Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the G.729AB decoder. */
eRetVal = eG729DDecodeInit(psDecConfig);
if (eRetVal != E_G729D_OK)
      return G729_FAILURE;
```

Step 6: 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 G729_FAST_MEMORY, as this may improve the performance (MHz) of the decoder. Pointer to the input buffer needs to be passed to decode frame routine.

Example pseudo code for allocating the input buffer

```
/* Allocate memory for input buffer G729_CODEC_SIZE = 82 */
ps16InBuf = alloc_fast((G729_CODEC_SIZE) * sizeof(G729_S16));
```

Step 7: Memory allocation for output buffer

The application has to allocate (4-byte aligned) memory for the output buffers to hold the decoded sample corresponding to one speech frame (80 words). The pointer to this output buffer needs to be passed to the eG729DDecodeFrame 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 G729_L_FRAME = 80 */
ps16OutBuf = alloc_fast (G729_L_FRAME * sizeof (G729_S16));
```

Step 8: Call the decode routine

The main G.729AB decoder function is eG729DDecodeFrame. This function decodes the G.729AB bitstream and writes bitstream to output buffer.

C prototype:

```
eG729DReturnType eG729DDecodeFrame (
                        sG729DDecoderConfigType *psDecConfig,
                        G729_S16  *ps16InBuf,
                        G729_S16 *ps16OutBuf);
```

Arguments:

• psDecConfig	Pointer to decoder configuration structure
• ps16InBuf	Pointer to G.729AB bitstream buffer
• ps16OutBuf	Pointer to output (decoded) buffer

Return value:

E_G729D_OK	indicates decoding was successful.
Others	**indicates error**

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

```
while (Not end of file)
{
      /* frame is not erasued */
      psDecConfig->u8APPDFrameErasureFlag = E_G729D_FR_NOTERASED;

      eRetVal = eG729DDecodeFrame(psDecConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != E_G729D_OK)
            return G729_FAILURE;
}
```

Step 9: Free memory

The application should release memory before exiting the decoder.

```
free (ps16OutBuf);
free (ps16InBuf);
for (s32i=0; s32i<s32NumMemReqs; s32i++)
{
    free(psDecConfig->sG729DMemInfo.asMemInfoSub[s32i].pvAPPDBasePtr);
}
free (psDecConfig);
```

4 Appendix

4.1 Common Header Interface file for G.729AB

The content of g729_common_api.h is given below.

```
#define G729_TRUE                1
#define G729_FALSE               0
#define G729_SUCCESS             0
#define G729_FAILURE             1
#define G729_FAST_MEMORY         0
#define G729_SLOW_MEMORY         1

#define G729_MEM_TYPE            G729_FAST_MEMORY
#define G729_MEM_STATIC          0
#define G729_MEM_SCRATCH         1

#define G729_L_FRAME              80
#define G729_CODEC_SIZE          82
#define G729_MAX_NUM_MEM_REQS          10
#define G729_PRIORITY_LOWEST     255
#define G729_PRIORITY_NORMAL     128
#define G729_PRIORITY_HIGHEST    0

#define G729_WARNING_BASE       11
#define G729_RECOVERROR_BASE    31
#define G729_FATALERROR_BASE    51

typedef char                      G729_S8;
typedef unsigned char             G729_U8;

typedef short                     G729_S16;
typedef unsigned short            G729_U16;

typedef int                       G729_S32;
typedef unsigned int              G729_U32;

typedef void                      G729_Void;

#ifdef      NULL
#undef      NULL
#define NULL      (G729_Void *)0
#endif

#define     G729_NULL NULL
```

4.2 Headers file for G.729AB Encoder Interface

g729_enc_api.h file is given below

```
#include "g729_common_api.h"

#define E_G729_VAD_DISABLE 0
#define E_G729_VAD_ENABLE 1

/***** Encoder return type, other return value to be added ****/
/* Success is assigned to 0.
   As of now there can be 20 warnings, starting from 11 to 30.
   Recoverable errors can be 20, starting from 31 to 50.
   Fatal errors can be 20, starting from 51 to 70.
   Later more error types can be added */
typedef enum
{
     E_G729E_OK = 0,
     E_G729E_WARNING = G729_WARNING_BASE,
     /*Recoverable error*/
     E_G729E_INVALID_MODE = G729_RECOVERROR_BASE,
     /*Recoverable error*/
     E_G729E_INIT_ERROR,
     /*fatal error base*/
     E_G729E_MEMALLOC_ERROR=G729_FATALERROR_BASE,
     E_G729E_ERROR
} eG729EReturnType;

typedef struct
{
    G729_S32   s32G729ESize;      /* Size in bytes */
    G729_U8    u8G729EType;      /* Static or scratch */
    G729_U8    u8G729EMemTypeFs; /* Memory type Fast or Slow */
    G729_Void  *pvAPPEBasePtr;    /* Pointer to the base memory,
                                      which will be allocated and
                                      filled by the application */
    G729_U8  u8G729EMemPriority; /* priority in which memory needs
                             to be allocated in fast memory */
} sG729EMemAllocInfoSubType;

/* Memory information structure array*/
typedef struct
{
    /* Number of valid memory requests */
    G729_S32                   s32G729ENumMemReqs;
    sG729EMemAllocInfoSubType   asMemInfoSub[G729_MAX_NUM_MEM_REQS];
} sG729EMemAllocInfoType;

typedef struct
{
    sG729EMemAllocInfoType sG729EMemInfo;
    G729_Void             *pvG729EEncodeInfoPtr;
    G729_U8             u8APPEVADFlag;
} sG729EEncoderConfigType;


G729_S8 * s8G729VersionInfo();
```

```
 eG729EReturnType eG729EQueryMem(sG729EEncoderConfigType *psEncConfig);
 eG729EReturnType eG729EEncodeInit(sG729EEncoderConfigType *psEncConfig);
 eG729EReturnType eG729EEncodeFrame(
                     sG729EEncoderConfigType *psEncConfig,
                     G729_S16 *ps16InBuf,
                     G729_S16 *ps16OutBuf
                     );
```

4.3 Header Interface File for G.729AB Decoder

g729_dec_api.h file is given below

```
#include "g729_common_api.h"

/***** Decoder return type, other return value to be added ****/
/* As of now there can be 20 warnings, starting from 11 to 30.
   Recoverable errors can be 20, starting from 31 to 50.
   Fatal errors can be 20, starting from 51 to 70.
   Later more error types can be added */
typedef enum
{
     E_G729D_OK = 0,
     E_G729D_WARNING = G729_WARNING_BASE,
     /* Recoverable error */
     E_G729D_INVALID_MODE = G729_RECOVERROR_BASE,
     /* Recoverable error */
     E_G729D_INIT_ERROR,
     /*fatal error base*/
     E_G729D_MEMALLOC_ERROR=G729_FATALERROR_BASE,
     E_G729D_ERROR
} eG729DReturnType;

typedef struct
{
    G729_S32   s32G729DSize;        /* Size in bytes     */
    G729_U8    u8G729DType;         /* Static or scratch */
    G729_U8    u8G729DMemTypeFs;    /* Memory type Fast or Slow */
    G729_Void  *pvAPPDBasePtr;      /* Pointer to the base memory,
                                       which will be allocated and
                                       filled by the application   */
    G729_U8    u8G729DMemPriority;  /* priority in which memory needs
                                       to be allocated in fast memory*/
} sG729DMemAllocInfoSubType;

/* Frame erasure enumeration */
#define     E_G729D_FR_ERASED 1
#define     E_G729D_FR_NOTERASED 0

/* Memory information structure array*/
typedef struct
{
    /* Number of valid memory requests */
    G729_S32                     s32G729DNumMemReqs;
    sG729DMemAllocInfoSubType    asMemInfoSub[G729_MAX_NUM_MEM_REQS];
    G729_U8                     u8APPDFrameErasureFlag;}
sG729DMemAllocInfoType;

typedef struct
{
     sG729DMemAllocInfoType    sG729DMemInfo;
     G729_Void                   *pvG729DDecodeInfoPtr;
} sG729DDecoderConfigType;

G729_S8 * s8G729VersionInfo(void);
eG729DReturnType eG729DQueryMem(sG729DDecoderConfigType *psDecConfig);
eG729DReturnType eG729DDecodeInit(sG729DDecoderConfigType *psDecConfig);
eG729DReturnType eG729DDecodeFrame(
                    sG729DDecoderConfigType *psDecConfig,
                    G729_S16 *ps16InBuf,
                    G729_S16 *ps16OutBuf
                    );
```