08-6911-API-ZCH66
JULY 16, 2008
2.2

Application Programmers Interface for G.723.1 Decoder and Encoder

ABSTRACT:

Application Programmers Interface for G.723.1 Decoder and Encoder

KEYWORDS:

Multimedia codecs, speech, G.723.1

© Freescale Semiconductor, Inc. 2005

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	13-Oct-2004	Tommy Tang	Initial Draft
1.0	15-Oct-2004	Ashok Kumar	Reviewed and re-worked
1.1	19-Nov-2004	Ashok Kumar	Added PCS comments
1.2	07-Jan-2005	Tommy Tang	Update performance requirements table
1.3	12-Jan-2005	Tommy Tang	Updated and removed references to ARM9E
1.4	20-Jan-2006	Kusuma	Updated with changed macros in the API
2.0	06-Feb-2006	Lauren Post	Using new format
2.1	05-Mar-2006	Surendra Jain	Review and Re-work
2.2	16-Jul-2008	Jackiea Pan	Update doc

© Freescale Semiconductor, Inc. 2005

Table of Contents

© Freescale Semiconductor, Inc. 2005

Introduction

1.1 Purpose

This document gives the details of the Application Programming Interface (API) of G.723.1 encoder and decoder. ITU G.723.1 specifies a coded representation that can be used for compressing the speech of multimedia services at a very low bit rate (5.3 kbps or 6.3 kbps).

The G.723.1 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.723.1 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.723.1 codec. The intended audience for this document is the development community who wish to use the G.723.1 codec in their systems.

1.4 References

1.4.1 Standards

- **ITU-T Recommendation G.723.1 (03/96)** –Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbits/s.
- **ITU-T Recommendation G.723.1 Annex A (11/96) –**Silence compression scheme
- **ITU-T Recommendation G.723.1 Implementers Guide (25 October 2002)** – Implementers' Guide for G.723.1
- **ITU-T G.723.1 Test vectors (1996)** –Description of the digital test sequences for the verification of the G.723.1 algorithm.

1.4.2 Freescale Multimedia References

- G.723.1 Codec Application Programming Interface – g723_codec_api.doc
- G.723.1 Codec Requirements Book – g723_codec_reqb.doc
- G.723.1 Codec Test Plan - g723_codec_test_plan.doc
- G.723.1 Codec Release notes - g723_codec_release_notes.doc
- G.723.1 Codec Test Results – g723_codec_test_results.doc
- G.723.1 Codec Performance Results – g723_codec_perf_results.doc
- G.723.1 Codec datasheet – g723_codec_datasheet.doc

© Freescale Semiconductor, Inc. 2005

- G.723.1 Interface Common Header – g723_com_api.h
- G.723.1 Interface Decoder Header – g723_dec_api.h
- G.723.1 Interface Encoder Header – g723_enc_api.h
- G.723.1 Decoder Application Code – g723_dectest.c
- G.723.1 Encoder Application Code – g723_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.723.1

.

© Freescale Semiconductor, Inc. 2005

© Freescale Semiconductor, Inc. 2005

2 API Description of Encoder

This section describes the steps followed by the application to call the G.723.1 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 G723E 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 configuration parameter structure

The application allocates memory for below mentioned structure.

```
/* Encoder parameter structure */
typedef struct
{
      sG723EMemAllocInfoType     sG723EMemInfo;
      G723_Void                 *pvG723EEncodeInfoPtr;
      G723_U8                              * pu8APPEInitializedDataStart;
      G723_U8                              u8APPEHighPassFilter;
      G723_U8                   u8APPEBitRate;
      G723_U8                   u8APPEVADFlag;
} sG723EEncoderConfigType;
```

Description of the encoder parameter structure *sG723EEncoderConfigType*

sG723EMemInfo

This is a memory information structure. The application needs to call the function eG723EQueryMem to get the memory requirements from encoder. The encoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

pvG723EEncodeInfoPtr

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.723 encoder every time the application is invoked.

u8APPEHighPassFilter

The application shall set the value of this variable to E_G723E_HPFILTER_DISABLE or E_G723E_HPFILTER_ENABLE before invoking encoder function every time (Refer Appendix for definition of these constants).

u8APPEBitRate

© Freescale Semiconductor, Inc. 2005

The application shall set the value of this variable to E_G723E_BITRATE_53 or E_G723E_BITRATE_63 corresponding to 5.3 kbps or 6.3 kbps respectively before invoking encoder function every time (Refer Appendix for definition of these constants).

u8APPEVADFlag

The application shall set the value of this variable to E_G723E_VAD_DISABLE or E_G723E_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 */
sG723EEncoderConfigType *psEncConfig;

/* Allocate fast memory for encoder config */
psEncConfig = (sG723EEncoderConfigType *)
                  alloc_fast(sizeof(sG723EEncoderConfigType));


/* Allocate memory for encoder to use */
psEncConfig->pvG723EEncodeInfoPtr = NULL;

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

Step 2: Get the encoder memory requirements

The G.723.1 encoder does not do any dynamic memory allocation inside the library. The application calls the function *eG723EQueryMem* to get the encoder memory requirements. This function must be called before any other encoder functions are invoked.

The function prototype of eG723EQueryMem is:

C prototype:

```
eG723EReturnType eG723EQueryMem (sG723EEncoderConfigType *psEncConfig);
```

Arguments:

- *psEncConfig* - Encoder config pointer.

Return value:

- E_G723E_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 */
```

```
      G723_S32                      s32G723ENumMemReqs;
      sG723EMemAllocInfoSubType     asMemInfoSub[G723_MAX_NUM_MEM_REQS];
} sG723EMemAllocInfoType;
```

Description of the structure *G723EMemAllocInfoType*

s32G723ENumMemReqs

The number of memory chunks requested by the encoder.

asMemInfoSub

This structure contains configuration parameters of each memory chunk.

```
typedef struct
 {
      G723_S32    s32G723ESize;          /* Size in bytes */
      G723_U8     u8G723EType;           /* Static or scratch */
      G723_U8     u8G723EMemTypeFs;    /* Fast or Slow memory */
      G723_Void   * pvAPPEBasePtr;
            /* Pointer to the base memory, which will be allocated and
            filled by the application  */
      G723_U8     u8G723EMemPriority;
            /* priority in which memory needs to be allocated in fast
            memory */
} sG723EMemAllocInfoSubType;
```

Description of the structure *sG723EMemAllocInfoSubType*

s32G723ESize

The size of each chunk in bytes

u8G723EType

The memory description field indicates whether requested chunk of memory is static or scratch. Codec will update this flag to G723_STATIC or G723_SCRATCH based on whether the requested memory chunk is used as G723_STATIC or as G723_SCRATCH memory.

u8G723EMemTypeFs

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 G723_SLOW_MEMORY (external memory) or G723_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 G723_FAST_MEMORY for which the application allocates a G723_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.

u8G723EMemPriority

This indicates the priority level of the memory type. The type of memory can be G723_SLOW_MEMORY or external memory, G723_FAST_MEMORY or internal memory. In case the type of memory is G723_FAST_MEMORY then the field u8G723EMemPriority 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

© Freescale Semiconductor, Inc. 2005

```
/* Query for memory */
eRetVal = eG723EQueryMem(psEncConfig);

if (eRetVal != E_G723E_OK)
      return G723_FAILURE;
```

Step 3: Allocate Data Memory for the encoder

In this step the application allocates the memory as required by the G.723 encoder and fills up the base memory pointer '*pvAPPEBasePtr*' of '*sG723EMemAllocInfoSubType*' 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.

```
sG723EMemAllocInfoSubType *psMem;

/* Number of memory chunks requested by the encoder */
s32NumMemReqs = psEncConfig->sG723EMemInfo.s32G723ENumMemReqs;

for(s32i = 0; s32i < s32NumMemReqs; s32i++)
{
      psMem = &( psEncConfig->sG723EMemInfo.asMemInfoSub[s32i]);
      if (psMem->s32G723EMemTypeFs == FAST_MEMORY)
      {
            /* If application does not have enough memory to allocate
               in fast memory, it can check priorty
               of requested chunk (psMem->u8G723EMemPriority) and
               allocate accordingly */
            /* This function allocates memory in internal memory */
            psMem->pvAPPEBasePtr = alloc_fast(psMem->s32G723ESize);
      }
      else
      {
            This function allocates memory in external memory */
            psMem->pvAPPEBasePtr = alloc_slow(psMem->s32G723ESize);
      }
}
```

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 *eG723EEncodeInit*. This function must be called before the main encode function is called.

C prototype

```
eG723EReturnType eG723EEncodeInit (sG723EEncoderConfigType *psEncConfig);
```

© Freescale Semiconductor, Inc. 2005

Arguments

- Pointer to encoder configuration structure

Return value

- E_G723E_OK - Initialization successful.
- Other codes - Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the G723 encoder. */
eRetVal = eG723EEncodeInit(psEncConfig);

if (eRetVal != E_G723E_OK)
      return G723_FAILURE;
```

Step 5: 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 G723_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. 240 words. Pointer to the input buffer needs to be passed to encode frame routine.

Example pseudo code for allocating the input buffer

```
/* Allocate memory for input buffer G723_L_FRAME = 240 */
ps16InBuf = alloc_fast(G723_L_FRAME/2 * sizeof(G723_S32));
```

Step 6: 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 at maximum supported bitrate i.e. 24 bytes corresponding to 6.3 kbps. The pointer to this output buffer needs to be passed to the eG723EEncodeFrame 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 CODED_FRAMESIZE = 24 */
ps16OutBuf = alloc_fast((CODED_FRAMESIZE/2) * sizeof(G723_S16));
```

Step 7: Call the encode routine

The main G.723.1 encoder function is eG723EEncode. This function encodes input sample and writes packed bitstream to output buffer.

C prototype:

```
eG723EReturnType eG723EEncodeFrame (
```

© Freescale Semiconductor, Inc. 2005

```
                                    sG723EEncoderConfigType *psEncConfig,
                                    G723_S16 *ps16InBuf,
                                    G723_S16 *ps16OutBuf);
```

Arguments:

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

Return value:

- E_G723E_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->u8APPEHighPassFilter = E_G723E_HPFILTER_ENABLE;
      psEncConfig->u8APPEBitRate = E_G723E_BITRATE_53;
      psEncConfig->u8APPEVADFlag = E_G723E_VAD_DISABLE;

      eRetVal= eG723EEncodeFrame(psEncConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != E_G723E_OK)
            return G723_FAILURE;
}
```

Step 8: 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->sG723EMemInfo.asMemInfoSub[s32i].pvAPPEBasePtr);
}
free (psEncConfig);
```

© Freescale Semiconductor, Inc. 2005

3 API Description of Decoder

This section describes the steps followed by the application to call the G.723.1. 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 G723D 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
 {
      sG723DMemAllocInfoType     sG723DMemInfo;
      G723_Void                * pvG723DDecodeInfoPtr ;
      G723_U8                                  *
pu8APPDInitializedDataStart;
      G723_U8                                  u8APPDPostFilter;
      G723_U8           u8APPDFrameErasureFlag;
} sG723DDecoderConfigType;
```

Description of the decoder parameter structure *sG723DDecoderConfigType*

sG723DMemInfo

This is a memory information structure. The application needs to call the function eG723DQueryMem to get the memory requirements from decoder. The decoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

pvG723DDecodeInfoPtr

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 G723.1 decoder every time the application is invoked.

u8APPDPostPassFilter

The application shall set the value of this variable to E_G723D_P_FILTER_DISABLE or E_G723D_P_FILTER_ENABLE before invoking decoder function every time (Refer Appendix for definition of these constants).

u8APPDFrameErasureFlag

© Freescale Semiconductor, Inc. 2005

This application needs to set this flag to E_G723D_FR_ERASED or E_G723D_FR_NOTERASED every time before invoking decode function every time (Refer Appendix for definition of these constants).

Example pseudo code for this step:

```
/* Allocate memory for the decoder parameter */
sG723DDecoderConfigType *psDecConfig;
psDecConfig = (sG723DDecoderConfigType *)
                              alloc(sizeof(sG723DDecoderConfigType);
/* Allocate memory for decoder to use */
psDecConfig->pvG723DdecodeInfoPtr = NULL;
/* Fill up the Relocated data position (not used) */
psDecConfig->pu8APPDInitializedDataStart = NULL;
```

Step 2: Get the decoder memory requirements

The G.723.1 decoder does not do any dynamic memory allocation inside the library. The application calls the function *eG723DQueryMem* to get the decoder memory requirements. This function must be called before any other decoder functions are invoked.

The function prototype of *eG723DQueryMem* is:

C prototype
*eG723DReturnType eG723DQueryMem (sG723DDecoderConfigType *psDecConfig);*

Arguments

- psDecConfig - Decoder configuration pointer.

Return value

- E_G723D_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 */
      G723_S32                    s32G723DNumMemReqs;
      sG723DMemAllocInfoSubType   asMemInfoSub[G723_MAX_NUM_MEM_REQS];
} sG723DMemAllocInfoType;
```

Description of the structure *sG723DMemAllocInfoType*

s32G723DNumMemReqs

The number of memory chunks requested by the decoder.

asMemInfoSub

This structure contains each description of each memory chunk.

© Freescale Semiconductor, Inc. 2005

```
typedef struct
{
      G723_S32    s32G723DSize;           /* Size in bytes */
      G723_U8     u8G723DType;           /* Static or scratch */
      G723_U8     u8G723DMemTypeFs;   /* Memory type Fast or Slow */
      G723_Void   *pvAPPDBasePtr;
            /* Pointer to the base memory, which will be  allocated and
            filled
            by the application */
      G723_U8     u8G723DMemPriority;
            /* priority in which memory needs to be allocated in fast
            memory */
} sG723DMemAllocInfoSubType;
```

Description of the structure *G723D_Mem_Alloc_Info_sub*

s32G723DSize

The size of each chunk in bytes.

u8G723DType

The memory description field indicates whether requested chunk of memory is static or scratch. Codec will update this flag to G723_STATIC or G723_SCRATCH based on whether the requested memory chunk is used as G723_STATIC or as G723_SCRATCH memory.

u8G723DMemTypeFs

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 G723_SLOW_MEMORY (external memory) or G723_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 G723_FAST_MEMORY for which the application allocates a G723_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.

u8G723DMemPriority

This indicates the priority level of the memory type. The type of memory can be G723_SLOW_MEMORY or external memory, G723_FAST_MEMORY or internal memory. In case the type of memory is G723_FAST_MEMORY then the field u8G723DMemPriority 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 = eG723DQueryMem(psDecConfig);

if (eRetVal != E_G723D_OK)
      return G723_FAILURE;
```

© Freescale Semiconductor, Inc. 2005

Step 3: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by the G.723.1 decoder and fills up the base memory pointer *'pvAPPDBasePtr'* of '*sG723DMemAllocInfoSubType*' 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.

```
sG723DMemAllocInfoSubType *psMem;

/* Number of memory chunks requested by the decoder */
s32NumMemReqs = psDecConfig->sG723DMemInfo.s32G723DNumMemReqs;

for(s32i = 0; s32i < s32NumMemReqs; s32i++)
{
      psMem = &(psDecConfig->sG723DMemInfo.asMemInfoSub[s32i]);
      if (psMem->s32G723DMemTypeFs == G723_FAST_MEMORY)
      {
            /* If application does not have enough memory to allocate
               in fast memory, it can check priorty
               of requested chunk (psMem->u8G723DMemPriority) and
               allocate accordingly */
            /* This function allocates memory in internal memory */
            psMem->pvAPPDBasePtr = alloc_fast(psMem->s32G723DSize);
      }
      else
      {
            This function allocates memory in external memory */
            psMem->pvAPPDBasePtr = alloc_slow(psMem->s32G723DSize);
      }
}
```

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 *eG723DDecodeInit*. This function must be called before the main decode function is called.

C prototype:

```
eG723DReturnType eG723DDecodeInit (sG723DDecoderConfigType *psDecConfig);
```

Arguments:

- *psDecConfig* Pointer to decoder configuration structure

Return value:

- E_G723D_OK - Initialization successful.
- Other codes - Initialization Error

© Freescale Semiconductor, Inc. 2005

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the G.723 decoder. */
eRetVal = eG723DDecodeInit(psDecConfig);
if (eRetVal != E_G723D_OK)
      return G723_FAILURE;
```

Step 5: Memory allocation for input buffer

The application has to allocate (2-byte aligned) the memory needed for the input buffer. This should be 24 bytes corresponding to 6.3 kbps. It is desirable to have the input buffer allocated in G723_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 CODED_FRAMESIZE = 24 */
ps16InBuf = alloc_fast((CODED_FRAMESIZE/2) * sizeof(G723_S16));
```

Step 6: 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 (240 words). The pointer to this output buffer needs to be passed to the eG723DDecodeFrame 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 G723_L_FRAME = 240 */
ps16OutBuf = alloc_fast (G723_L_FRAME/2 * sizeof (G723_S32));
```

Step 7: Call the decode routine

The main G.723.1 decoder function is eG723DDecodeFrame. This function decodes the G.723.1 bitstream and writes bitstream to output buffer.

C prototype:

```
eG723DReturnType eG723DDecodeFrame (
                        sG723DDecoderConfigType *psDecConfig,
                        G723_S16  *ps16InBuf,
                        G723_S16 *ps16OutBuf);
```

Arguments:

- psDecConfig — Pointer to decoder configuration structure
- ps16InBuf — Pointer to G.723.1 bitstream buffer
- ps16OutBuf — Pointer to output (decoded) buffer

© Freescale Semiconductor, Inc. 2005

Return value:

- E_G723D_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)
{
      psDecConfig->u8APPDPostPassFilter = E_G723D_P_FILTER_ENABLE;
      /* frame is not erasued */
      psDecConfig->u8APPDFrameErasureFlag = E_G723D_FR_NOTERASED;

      eRetVal = eG723DDecodeFrame(psDecConfig, ps16InBuf, ps16OutBuf);
      if (eRetVal != E_G723D_OK)
            return G723_FAILURE;
}
```

Step 8: Free memory

The application should release memory before exiting the decoder.

```
free (ps16OutBuf);
free (ps16InBuf);
for (s32i=0; s32i<s32NumMemReqs; s32i++)
{
    free(psDecConfig->sG723DMemInfo.asMemInfoSub[s32i].pvAPPDBasePtr);
}
free (psDecConfig);
```

© Freescale Semiconductor, Inc. 2005

4 Appendix

4.1 Common Header Interface file for for G.723.1

The content of g723_common_api.h is given below.

```
#define G723_TRUE               1
#define G723_FALSE              0
#define G723_SUCCESS            0
#define G723_FAILURE            1
#define G723_FAST_MEMORY        0
#define G723_SLOW_MEMORY        1

#define G723_MEM_TYPE           G723_FAST_MEMORY
#define G723_MEM_STATIC         0
#define G723_MEM_SCRATCH        1

#define G723_L_FRAME            240
#define CODED_FRAMESIZE         24
#define G723_MAX_NUM_MEM_REQS   10
#define G723_PRIORITY_LOWEST    255
#define G723_PRIORITY_NORMAL    128
#define G723_PRIORITY_HIGHEST   0

#define G723_WARNING_BASE      11
#define G723_RECOVERROR_BASE   31
#define G723_FATALERROR_BASE   51

typedef char                    G723_S8;
typedef unsigned char           G723_U8;

typedef short                   G723_S16;
typedef unsigned short          G723_U16;

typedef int                     G723_S32;
typedef unsigned int            G723_U32;

typedef void                    G723_Void;
```

4.2 Headers file for G.723.1 Encoder Interface

g723_enc_api.h file is given below

```
#include "g723_common_api.h"

/* this should be set to 0xF to enable all the debug level logs
 * value of 0x1 will enbale - log level 1 only
 * value of 0x2 will enable - log level 2 only
```

© Freescale Semiconductor, Inc. 2005

```
 * value of 0x3 will enable - level 1 and 2 logs
 * value of 0x4 will enable - level 3 logs
 * value of 0x8 will enabel - level 4 log
 */

#define G723E_DEBUG_LVL 0x0

/* this is start of log message */
#define G723E_BEGIN_DBG_MSGID  1000
/* end of message id */
#define G723E_END_DBG_MSGID    1199

/***** 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_G723E_OK = 0,
     E_G723E_WARNING = G723_WARNING_BASE,
     /*Recoverable error*/
     E_G723E_INVALID_MODE = G723_RECOVERROR_BASE,
     /*Recoverable error*/
     E_G723E_INIT_ERROR,
     /*fatal error base*/
     E_G723E_MEMALLOC_ERROR=G723_FATALERROR_BASE,
     E_G723E_ERROR
} eG723EReturnType;

typedef enum
{
      E_G723E_HPFILTER_DISABLE,
      E_G723E_HPFILTER_ENABLE
} eG723EHighPassFilterType;

typedef enum
{
      E_G723_VAD_DISABLE,
      E_G723_VAD_ENABLE
} eG723EVadCngType;

typedef enum
{
      E_G723E_BITRATE_53,
      E_G723E_BITRATE_63
}eG723EBitRateType;

typedef struct
{
    G723_S32   s32G723ESize;      /* Size in bytes */
    G723_U8    u8G723EType;      /* Static or scratch */
    G723_U8    u8G723EMemTypeFs; /* Memory type Fast or Slow */
```

© Freescale Semiconductor, Inc. 2005

```
    G723_Void  *pvAPPEBasePtr;    /* Pointer to the base memory,
                                     which will be allocated and
                                     filled by the application */
    G723_U8  u8G723EMemPriority; /* priority in which memory needs
                                to be allocated in fast memory */
} sG723EMemAllocInfoSubType;

/* Memory information structure array*/
typedef struct
{
    /* Number of valid memory requests */
    G723_S32                   s32G723ENumMemReqs;
    sG723EMemAllocInfoSubType   asMemInfoSub[G723_MAX_NUM_MEM_REQS];
} sG723EMemAllocInfoType;

typedef struct
{
    sG723EMemAllocInfoType sG723EMemInfo;
    G723_Void              *pvG723EEncodeInfoPtr;
    G723_U8             *pu8APPEInitializedDataStart;
    G723_U8             u8APPEHighPassFilter;
    G723_U8             u8APPEBitRate;
    G723_U8             u8APPEVADFlag;
} sG723EEncoderConfigType;

extern eG723EReturnType eG723EQueryMem(sG723EEncoderConfigType
                              *psEncConfig);
extern eG723EReturnType eG723EEncodeInit(sG723EEncoderConfigType
                              *psEncConfig);
extern eG723EReturnType eG723EEncodeFrame(
                     sG723EEncoderConfigType *psEncConfig,
                     G723_S16 *ps16InBuf,
                     G723_S16 *ps16OutBuf
                     );
const char *G723E_get_version_info(void);
```

4.3 Header Interface File for G.723.1 Decoder

g723_dec_api.h file is given below

```
#include "g723_common_api.h"

/* this should be set to 0xF to enable all the debug level logs
 * value of 0x1 will enbale - log level 1 only
 * value of 0x2 will enable - log level 2 only
 * value of 0x3 will enable - level 1 and 2 logs
 * value of 0x4 will enable - level 3 logs
 * value of 0x8 will enabel - level 4 log
 */
#define G723D_DEBUG_LVL 0x0
```

© Freescale Semiconductor, Inc. 2005

```
/* this is start of log message */
#define G723D_BEGIN_DBG_MSGID   1200
#define G723D_END_DBG_MSGID     1300


/***** 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_G723D_OK = 0,
     E_G723D_WARNING = G723_WARNING_BASE,
     /* Recoverable error */
     E_G723D_INVALID_MODE = G723_RECOVERROR_BASE,
     /* Recoverable error */
     E_G723D_INIT_ERROR,
     /*fatal error base*/
     E_G723D_MEMALLOC_ERROR=G723_FATALERROR_BASE,
     E_G723D_ERROR
} eG723DReturnType;

/* Post filter enumeration */
typedef enum
{
      E_G723D_P_FILTER_DISABLE,
      E_G723D_P_FILTER_ENABLE
} eG723DPostFilterType;

/* Frame erasure enumeration */
typedef enum
{
      E_G723D_FR_ERASED,
      E_G723D_FR_NOTERASED
} eG723DFrameErasureType;

typedef struct
{
    G723_S32   s32G723DSize;        /* Size in bytes     */
    G723_U8    u8G723DType;         /* Static or scratch */
    G723_U8    u8G723DMemTypeFs;    /* Memory type Fast or Slow */
    G723_Void  *pvAPPDBasePtr;      /* Pointer to the base memory,
                                       which will be allocated and
                                       filled by the application   */
    G723_U8    u8G723DMemPriority;  /* priority in which memory needs
                                       to be allocated in fast memory*/
} sG723DMemAllocInfoSubType;

/* Memory information structure array*/
typedef struct
{
    /* Number of valid memory requests */
```

© Freescale Semiconductor, Inc. 2005

```
    G723_S32                    s32G723DNumMemReqs;
    sG723DMemAllocInfoSubType   asMemInfoSub[G723_MAX_NUM_MEM_REQS];
} sG723DMemAllocInfoType;

typedef struct
{
     sG723DMemAllocInfoType    sG723DMemInfo;
     G723_Void                   *pvG723DDecodeInfoPtr;
     G723_U8                   *pu8APPDInitializedDataStart;
     G723_U8                   u8APPDPostFilter;
     G723_U8                          u8APPDFrameErasureFlag;
} sG723DDecoderConfigType;

extern eG723DReturnType eG723DQueryMem(sG723DDecoderConfigType
                  *psDecConfig);
extern eG723DReturnType eG723DDecodeInit(sG723DDecoderConfigType
                                  *psDecConfig);
extern eG723DReturnType eG723DDecodeFrame(
                   sG723DDecoderConfigType *psDecConfig,
                   G723_S16 *ps16InBuf,
                   G723_S16 *ps16OutBuf

                   );
const char *G723D_get_version_info(void);
```