MOTOROLA

Embedded SDK (Software Development Kit)

G.729AB Vocoder Library

SDK139/D
Rev. 1, 07/19/2002

MOTOROLA intelligence everywhere™ | digital dna™

Contents

List of Tables

List of Figures

List of Examples

About This Document

This manual describes the G.729AB Vocoder algorithm for use with Motorola's Embedded Software Development Kit, (SDK).

Audience

This document targets software developers implementing the G.729AB Vocoder within software applications.

Organization

This manual is arranged in the following sections:

- **Chapter 1, Introduction**--provides a brief overview of this document
- **Chapter 2, Directory Structure**--provides a description of the required core directories
- **Chapter 3, G.729AB Vocoder Library Interface**--describes all of the G.729AB Library functions
- **Chapter 4, Building the G.729AB Vocoder Library**--tells how to execute the system library project build
- **Chapter 5, Linking Applications with the G.729AB Vocoder Library**--describes the organization of the G.729AB Library
- **Chapter 6, G.729AB Vocoder Applications**--describes the use of the G.729AB Library through test/demo applications
- **Chapter 7, License**--provides the license required to use this product

Suggested Reading

We recommend that you have a copy of the following references:

- *DSP56800E Reference Manual*, DSP56800ERM/D
- *DSP5685x User's Manual*, DSP5685xUM/AD
- *Inside CodeWarrior: Core Tools*, Metrowerks Corp.

Conventions

This document uses the following notational conventions:

Typeface, Symbol or Term	Meaning	Examples
`Courier Monospaced Type`	Code examples	`//Process command for line flash`
Italic	Directory names, project names, calls, functions, statements, procedures, routines, arguments, file names, applications, variables, directives, code snippets in text	...and contains these core directories: *applications* contains applications software... ...CodeWarrior project, *3des.mcp* is... ...the *pConfig* argument.... ...defined in the C header file, *aec.h*....
Bold	Reference sources, paths, emphasis	...refer to the **Targeting DSP56F80x Platform** manual.... ...see: **C:\Program Files\Motorola\Embedded SDK\help\tutorials**
Blue Text	Linkable on-line	...refer to Chapter 7, License....
Number	Any number is considered a positive value, unless preceded by a minus symbol to signify a negative value	3V -10 DES^{-1}
ALL CAPITAL LETTERS	# defines/ defined constants	# define INCLUDE_STACK_CHECK
Brackets [...]	Function keys	...by pressing function key [F7]
Quotation marks, "..."	Returned messages	...the message, "Test Passed" is displayed.... ...if unsuccessful for any reason, it will return "NULL"...

Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document. As this template develops, this list will be generated from the document. As we develop more group resources, these acronyms will be easily defined from a common acronym dictionary. Please note that while the acronyms are in solid caps, terms in the definition should be initial capped ONLY IF they are trademarked names or proper nouns.

ADPCM	Adaptive Differential Pulse Code Modulation
CNG	Comfort Noise Generation
DSP	Digital Signal Processor or Digital Signal Processing
DTX	Discontinuous Transmission
I/O	Input/Output
IDE	Integrated Development Environment
LSB	Least Significant Byte
MIPS	Million Instructions Per Second
MSB	Most Significant Byte
OnCE™	On-Chip Emulation
OMR	Operating Mode Register
PC	Program Counter
SDK	Software Development Kit
SID	Silence Insertion Descriptor
SP	Stack Pointer
SPI	Serial Peripheral Interface
SR	Status Register
SRC	Source
VAD	Voice Activity Detection

References

The following sources were referenced to produce this book:

[1] *DSP56800E Reference Manual*, DSP56800ERM/D

[2] *DSP5685x User's Manual*, DSP5685xUM/AD

[3] *Embedded SDK Programmer's Guide*, SDK101/D

[4] *ITU-T Recommendation G.729AB*

Chapter 1
Introduction

Welcome to Motorola's Family of Digital Signal Processors, (DSPs). This document describes the G.729AB Vocoder Library, which is a part of Motorola's comprehensive Software Development Kit, (SDK), for its DSPs. In this document, you will find all the information required to use and maintain the G.729AB Vocoder Library interface and algorithms.

Motorola provides these algorithms to you for use on Motorola Digital Signal Processors to expedite your application development and reduce the time it takes to bring your own products to market.

Motorola's G.729AB Vocoder Library is licensed for your use on Motorola processors. Please refer to the standard Software License Agreement in **Chapter 7** for license terms and conditions; please consult with your Motorola representative for premium product licensing.

1.1 Quick Start

Motorola's Embedded SDK is targeted to a large variety of hardware platforms. To take full advantage of a particular hardware platform, use **Quick Start** from the appropriate **Targeting Motorola DSP568xx Platform** documentation.

For example, the **Targeting Motorola DSP56852 Platform** manual provides more specific information and examples about this hardware architecture. If you are developing an application for a DSP56852EVM board or any other DSP56852 development system, refer to the **Targeting Motorola DSP56852 Platform** manual for **Quick Start** or other DSP56852-specific information.

1.2 Overview of G.729AB Vocoder

1.2.1 Background

The ITU-T G.729 Recommendation describes an algorithm for speech coding at 8kbps using the Conjugate-Structure Algebraic-Code-Excited Linear-Prediction model. The coder operates on speech frames of 80 samples at a sampling rate of 8000 samples per second (each frame corresponds at 10ms of speech).

The ITU-T G.729 Annex A is a less-complicated version of the G.729 vocoder. The block diagram of the G.729 Annex A vocoder is similar to that of the original G.729 Recommendation. Most blocks are identical; only a few blocks have different implementations.

The ITU-T G.729 Annex B is used to reduce the transmission rate during silent periods of speech. Each frame first passes a voice activity detection (VAD) block that decides if the frame is “speech” or “silent”. This decision is transmitted to the encoder and the decoder via the bit-stream, activating either the “Active Voice” block or “Non-Active Voice” block in both the encoder and decoder.

The “Active Voice” block is invoked for “speech” frames and represents the G.729A vocoder. In this case, the bit-stream that results by encoding with G.729A is 80 bits long.

The “Non-Active Voice” block is called for “silent” frames and means discontinuous transmission (DTX) in the encoder and comfort noise generation (CNG) in the decoder. The bit-stream that results by encoding a “silent” frame is named Silence Insertion Descriptor (SID) and is 15 or 16 bits long, depending on the transmission mode (bit or octet).

Annex B of ITU-T G.729 Recommendation contains a complete description of the VAD algorithm and DTX and CNG blocks.

1.2.2 Features and Performance

For details on Memory and MIPS for a particular DSP, refer to the **Libraries** chapter of the appropriate Targeting manual.

Chapter 2
Directory Structure

2.1 Required Core Directories

Figure 2-1 details required platform directories:

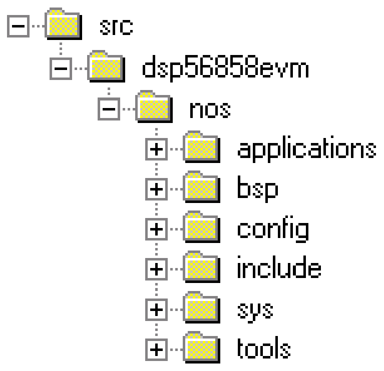
```
src
  dsp56858evm
    nos
      applications
      bsp
      config
      include
      sys
      tools
```

Figure 2-1. Core Directories

As shown in Figure 2-1, DSP56858EVM has no operating system (nos) support. This platform contains these core directories:

- ***applications*** contains applications software that can be exercised on this platform
- ***bsp*** contains board support package specific for this platform
- ***config*** contains default hardware/software configurations for this platform
- ***include*** contains SDK header files which define the Application Programming Interface
- ***sys*** contains required system components
- ***tools*** contains utilities used by system components

There are also optional directories that include domain-specific libraries.

2.2 Optional (Domain-Specific) Directories

Figure 2-2 demonstrates how the G.729AB Vocoder (*g729ab*) is encapsulated in the domain-specific directory *telephony*.

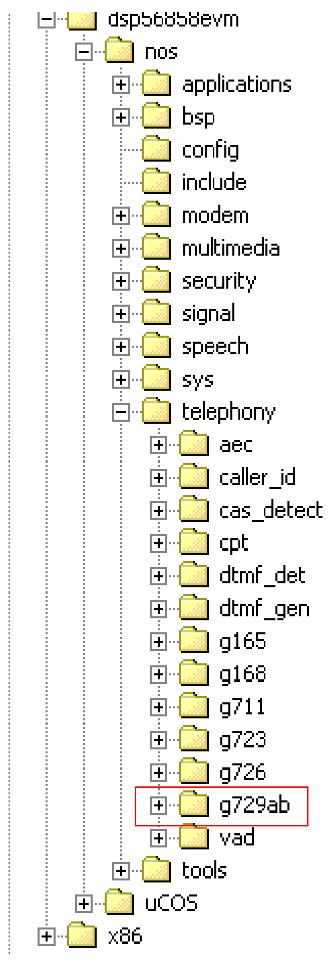

Figure 2-2. DSP56858 Directories

Figure 2-3 shows the *g729ab* directory structure.

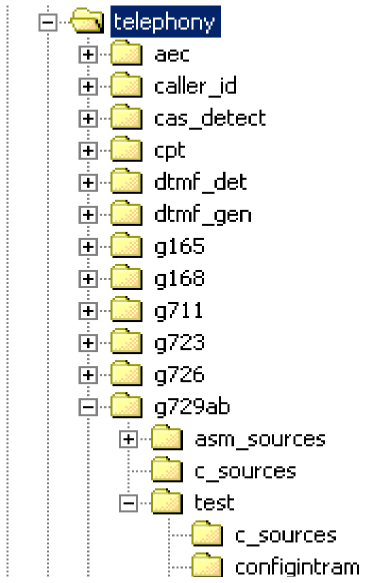

Figure 2-3. *g729ab* Vocoder Directory Structure

The ***g729ab*** directory contains the G.729AB library as well as the following sub-directories:

- ***asm_sources*** includes assembly source files for G.729AB
- ***c_sources*** includes "C" source files for G.729AB
- ***test*** includes a test project, *c_source* files, and configuration necessary for testing the G.729AB library modules
  - ***configintram*** contains configuration files *appconfig.c*, *appconfig.h* and *linker.cmd* specific to G.729ab Encoder/Decoder testing
  - ***c_Sources*** contains test source code for encoder/decoder testing

2.3 Demo Directory Structure

Figure 2-4 demonstrates how the components of the G.729AB vocoder demo (*loopback_vocoders* and *recorder_player*) are encapsulated in the *telephony* directory under *applications*.

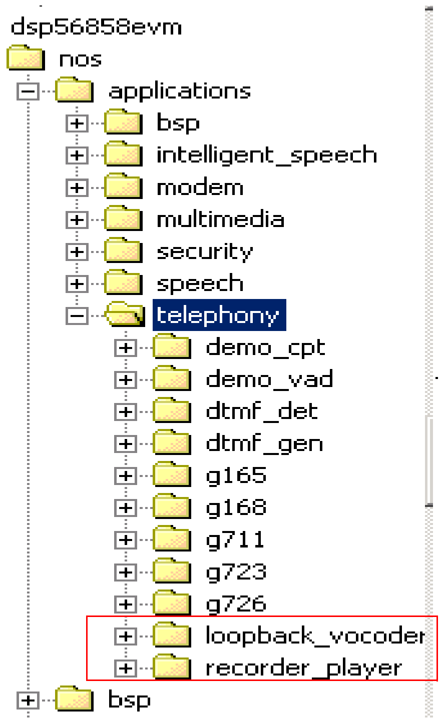

Figure 2-4. *telephony* Directory Structure

The *loopback_vocoders* directory includes G.729AB, G.711, and G.726 vocoder demo-specific algorithms. **Figure 2-5** illustrates the *loopback_vocoders* directory structure.

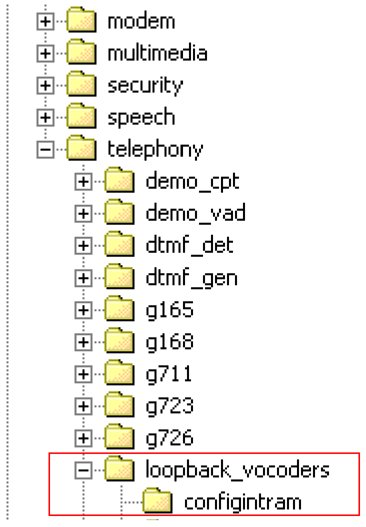

Figure 2-5. *loopback_vocoders* Directory Structure

- ***loopback_vocoders*** includes c sources and configuration necessary for testing the demo modules for G.729AB, G.726 and G.711 vocoders
  — ***configintram*** contains configuration files *appconfig.c*, *appconfig.h* and *linker.cmd* specific to the demo for G.729AB, G.726, and G.711 vocoders

The *recorder_player* directory includes G.729AB and G.711 vocoder demo-specific algorithms. Figure 2-6 shows the *recorder_player* directory structure.

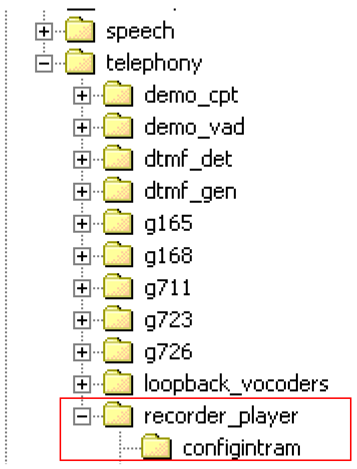

Figure 2-6. *recorder_player* Directory Structure

- ***recorder_player*** includes "C" sources and configuration necessary for testing the demo modules for G.729AB and G.711 vocoders
  - — ***configintram*** contains configuration files *appconfig.c*, *appconfig.h* and *linker.cmd* specific to the demo for G.729AB and G.711 vocoders

Chapter 3
G.729AB Vocoder Library Interface

3.1 G.729AB Vocoder Services

The principal application of G.729AB vocoder is for Voice over IP (VoIP) telephony. It codes speech at 8kbps using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction model. The coder operates on speech frames of 80 samples, at a sampling rate of 8000 samples per second (each frame corresponds at 10ms of speech).

The vocoder provides a compression rate of 10:1, and even better for silent frames.

3.2 Interface

The C interface for the G.729 vocoder library is defined in the C header files *g729ab.h*, shown in **Code Example 3-1**.

Code Example 3-1. C Header File *g729ab.h*

```
/*==============================================================================

    g729ab.h

    Application Programming Interface for G729AB library

    Platform: DSP56800E

================================================================================
                             Motorola DSP Center Romania
                 (c) Copyright Motorola 2001, All Rights Reserved
==============================================================================*/


#ifdef __cplusplus
extern "C" {
#endif
```

```
#ifndef __G729AB_H
#define __G729AB_H

/*============================================================================
                         INCLUDE FILES
==========================================================================*/

#include "port.h"                            /* platform specifics           */

/*============================================================================
                         CONSTANTS
==========================================================================*/
#define G729AB_SIZE_ENCODER_CH_DATA 708     /* Size of encoder channel data */
#define G729AB_SIZE_DECODER_CH_DATA 430     /* Size of decoder channel data */
#define G729AB_L_FRAME              80      /* Frame size.                  */
#define G729AB_BITSTREAM_SIZE       (2+80)  /* Size of bitstream buffer     */

#define IN                                  /* input parameters             */
#define OUT                                 /* output parameters            */
#define IN_OUT                              /* in/out parameters            */

/*============================================================================
                         TYPEDEFS (STRUCTURES, UNIONS, ENUMS)
==========================================================================*/

typedef struct
{
    Word32 vector[G729AB_SIZE_ENCODER_CH_DATA/2];
} g729ab_sEncoderChannelData;

typedef struct
{
    Word32 vector[G729AB_SIZE_DECODER_CH_DATA/2];
} g729ab_sDecoderChannelData;


/*============================================================================
                         GLOBAL FUNCTIONS
==========================================================================*/

/*============================================================================

FUNCTION: g729abEncoderCreate()

DESCRIPTION:
    Allocates one encoder channel data structure.

ARGUMENTS PASSED:
    None

RETURN VALUE:
    Pointer to new encoder channel data object.
    NULL if memory allocation failed.

PRE-CONDITIONS:
    None
```

```
POST-CONDITIONS:
    None

IMPORTANT NOTES:
    None

==============================================================================*/
g729ab_sEncoderChannelData * g729abEncoderCreate(void);

/*==============================================================================

FUNCTION: g729abEncoderInit()

DESCRIPTION: Initialize the encoder channel data

ARGUMENTS PASSED:
    *pEncChData    (OUT)        -pointer to encoder channel data struct

RETURN VALUE:
    None

PRE-CONDITIONS:
    Encoder channel data struct should be allocated by the caller.

POST-CONDITIONS:
    None

IMPORTANT NOTES:
    This function must be called one time per channel.

==============================================================================*/
void g729abEncoderInit(
    OUT     g729ab_sEncoderChannelData *pEncChData
);

/*==============================================================================

FUNCTION: g729abEncoder()

DESCRIPTION: Encode a speech frame

ARGUMENTS PASSED:
    *pSpeechBuffer  (IN)        -pointer to the speech buffer (PCM coded)
    *pEncParm       (OUT)       -pointer to the output buffer
    *pEncChData     (IN/OUT)    -pointer to encoder channel data struct
    enable_vad      (IN)        -flag to enable voice activity detection module

RETURN VALUE:
    None

PRE-CONDITIONS:
    Encoder channel data struct should be initialized before this
    function call (using g729abEncoderInit).

POST-CONDITIONS:
    None
```

```
IMPORTANT NOTES:
    None

=============================================================================*/
void g729abEncoder(
    IN      Word16 *pSpeechBuffer,
    OUT     Word16 *pEncParm,
    IN_OUT  g729ab_sEncoderChannelData *pEncChData,
    IN      Word16 enable_vad
);

/*=============================================================================

FUNCTION: g729abEncoderDestroy()

DESCRIPTION:
    Deallocates a encoder channel data structure.

ARGUMENTS PASSED:
    Pointer to encoder channel data object.

RETURN VALUE:
    None

PRE-CONDITIONS:
    None

POST-CONDITIONS:
    None

IMPORTANT NOTES:
    None

=============================================================================*/
void g729abEncoderDestroy(
    IN      g729ab_sEncoderChannelData * pEncChData
);
/*=============================================================================

FUNCTION: g729abDecoderCreate()

DESCRIPTION:
    Allocates one decoder channel data structure.

ARGUMENTS PASSED:
    None

RETURN VALUE:
    Pointer to new decoder channel data object.
    NULL if memory allocation failed.

PRE-CONDITIONS:
    None

POST-CONDITIONS:
    None
```

```
IMPORTANT NOTES:
    None

==============================================================================*/
g729ab_sDecoderChannelData * g729abDecoderCreate(void);

/*==============================================================================

FUNCTION: g719abDecoderInit()

DESCRIPTION: Initialize the decoder channel data

ARGUMENTS PASSED:
    *pDecChData     (OUT)       -pointer to decoder channel data struct

RETURN VALUE:
    None

PRE-CONDITIONS:
    Decoder channel data struct should be allocated by the caller.

POST-CONDITIONS:
    None

IMPORTANT NOTES:
    This function must be called one time per channel.

==============================================================================*/
void g729abDecoderInit(
    OUT     g729ab_sDecoderChannelData *pDecChData
);

/*==============================================================================

FUNCTION: g729abDecoder()

DESCRIPTION: Decode a speech frame

ARGUMENTS PASSED:
    *pEncParm       (IN)        -pointer to the parameters' array
    *pDecodedSpeech (OUT)       -pointer to the reconstructed speech buffer
                                 (PCM coded)
    *pDecChData     (IN/OUT)    -pointer to decoder channel data struct

RETURN VALUE:
    None

PRE-CONDITIONS:
    Decoder channel data struct should be initialized before this
    function call (using g729abDecoderInit).

POST-CONDITIONS:
    None

IMPORTANT NOTES:
    None
```

```
=============================================================================*/
void g729abDecoder(
    IN      Word16 *pEncParm,
    OUT     Word16 *pDecodedSpeech,
    IN_OUT  g729ab_sDecoderChannelData *pDecChData
);

/*==============================================================================

FUNCTION: g729abDecoderDestroy()

DESCRIPTION:
    Deallocates a decoder channel data structure.

ARGUMENTS PASSED:
    Pointer to decoder channel data object.

RETURN VALUE:
    None

PRE-CONDITIONS:
    None

POST-CONDITIONS:
    None

IMPORTANT NOTES:
    None

=============================================================================*/
void g729abDecoderDestroy(
    IN      g729ab_sDecoderChannelData * pDecChData
);
/*============================================================================*/
#endif __G729AB_H

#ifdef __cplusplus
}
#endif
```

3.3 Specifications

The following pages describe the G.729AB vocoder library functions.

Function arguments for each routine are described as *in*, *out*, or *inout*. An *in* argument means that the parameter value is an input only to the function. An *out* argument means that the parameter value is an output only from the function. An *inout* argument means that a parameter value is an input to the function, but the same parameter is also an output from the function.

Typically, *inout* parameters are input pointer variables in which the caller passes the address of a preallocated data structure to a function. The function stores its results within that data structure. The actual value of the *inout* pointer parameter is not changed.

3.3.1 *g729abEncoderCreate*

Call(s):

```
g729ab_sEncoderChannelData * g729abEncoderCreate(void);
```

Required Header: *g729ab.h*

Arguments: None

Description: The *g729abEncoderCreate()* function creates an instance of G.729AB encoder. This means it allocates a data structure, *g729ab_sEncoderChannelData*, which is used by the encoder to pass the status information for a specific channel from one frame to the next.

The *g729ab_sEncoderChannelData* structure has G729AB_SIZE_ENCODER_CH_DATA words and is double-word aligned. The internal structure of the encoder channel data is transparent for the user, so it is declared as a vector of Word32 values (Word32 means the structure must be double-word aligned), For additional information, see **Code Example 3-1**.

The *g729abEncoderCreate*() function allocates memory dynamically using the *mem* library routines, and it does not initialize the allocated structure. To initialize the encoder channel data, the user must call *g729abEncoderInit()*. If a *g729abEncoderCreate()* function is called to create an instance, then *g729abEncoderDestroy()* should be used to destroy the instance.

Alternatively, the user can allocate memory statically, simply by defining a variable of type *g729ab_sEncoderChannelData*. In this case, the user can call the *g729abEncoderInit()* function directly, bypassing the *g729abEncoderCreate()* function call.

Returns: Upon successful completion, *g729abEncoderCreate()* returns a pointer of type *g729ab_sEncoderChannelData*. If data memory is not available, *g729abEncoderCreate()* will return "NULL".

Special Considerations: None

3.3.2 *g729abEncoderInit*

Call(s):

```
void g729abEncoderInit(
    OUT     g729ab_sEncoderChannelData *pEncChData
);
```

Required Header: *g729ab.h*

Arguments:

Table 3-1. ***g729abEncoderInit()* Arguments**

pEncChData	*out*	A pointer to a structure of type *g729ab_sEncoderChannelData*; the structure must be allocated before the function call

Description: The *g729abEncoderInit()* function initializes the G.729AB encoder algorithm. During initialization, all channel data are set to their initial values in preparation for G.729AB encoder operation. The *g729abEncoderInit()* function must be called only once per channel before the first call to the *g729abEncoder()* function.

Returns: None

Special Considerations: None

3.3.3 g*729abEncoder*

Call(s):

```
void g729abEncoder(
    IN      Word16 *pSpeechBuffer,
    OUT     Word16 *pEncParm,
    IN_OUT  g729ab_sEncoderChannelData *pEncChData,
    IN      Word16 enable_vad
);
```

Required Header: *g729ab.h*

Arguments:

Table 3-2. *g729abEncoder()* Arguments

pSpeechBuffer	*in*	Pointer to speech input buffer; this buffer has G729AB_L_FRAME words.
pEncParm	*out*	Pointer to encoded parameters buffer, which has G729AB_BITSTREAM_SIZE words. It must be allocated before the call to the *g729abEncoder()* function.
pEncChData	*inout*	Pointer to the *g729ab_sEncoderChannelData* structure
enable_vad	*in*	The flag that enables (1) or disables (0) the voice activity detection (VAD) block.

Description: When the entire *pSpeechBuffer* is filled (80 samples), the *g729abEncoder()* must be called for each speech frame. It encodes the input data and generates the parameters that are transmitted for one frame. The first word of the *pEncParm* buffer is a synchronization word, the second word represents the encoded bit-stream length. Each word of the remaining of *pEncParm* buffer codes one bit of the G.729AB bit-stream: 0x007F means a zero-bit, 0x0081 means a one-bit.

The *g729abEncoder()* function uses information from the previous frame, stored in the *g729ab_sEncoderChannelData* structure, and updates information needed for the next frame processing in the same structure. The parameter *enable_vad* activates the G.729 annex B contribution (VAD, CNG) when it is non-zero.

In a multichannel scheme, each channel keeps channel information in its own *g729ab_sEncoderChannelData* object.

Returns: None

Special Considerations: None

3.3.4 g729abEncoderDestroy

Call(s):

```
void g729abEncoderDestroy(
    IN      g729ab_sEncoderChannelData * pEncChData
);
```

Required Header: *g729ab.h*

Arguments:

Table 3-3. *g729abEncoderDestroy()* Arguments

pEncChData	*in*	Pointer to an instance of G.729AB encoder generated by a call to *g729abEncoderCreate()*

Description: The *g729abEncoderDestroy()* function destroys the instance of the G.729AB encoder originally created by a call to *g729abEncoderCreate()*.

If the *g729ab_sEncoderChannelData* object was statically created, the *g729abEncoderDestroy()* function should not be called.

Returns: None

Special Considerations: None

3.3.5 *g729abDecoderCreate*

Call(s):

```
g729ab_sDecoderChannelData * g729abDecoderCreate(void)
```

Required Header: *g729ab.h*

Arguments: None

Description: The *g729abDecoderCreate()* function creates an instance of G.729AB decoder. This means it allocates a data structure, *g729ab_sDecoderChannelData*, which is used by the decoder to pass the status information for a specific channel from one frame to the next.

The *g729ab_sDecoderChannelData* structure has G729AB_SIZE_DECODER_CH_DATA words and is double-word aligned. The internal structure of the decoder channel data is transparent for the user, so it is declared as a vector of Word32 values (Word32 means the structure must be double-word aligned). For more information, see **Code Example 3-1**.

The *g729abDecoderCreate*() function allocates memory dynamically using the *mem* library routines, and it does not initialize the allocated structure. To initialize the decoder channel data, the user must call *g729abDecoderInit()*. If a *g729abDecoderCreate()* function is called to create an instance, then *g729abDecoderDestroy()* should be used to destroy the instance.

Alternatively, the user can allocate memory statically, simply by defining a variable of type *g729ab_sDecoderChannelData*. In this case, the user can call the *g729abDecoderInit()* function directly, bypassing the *g729abDecoderCreate()* function.

Returns: Upon successful completion, *g729abDecoderCreate()* returns a pointer of type *g729ab_sDecoderChannelData*. If data memory is not available, *g729abDecoderCreate()* will return "NULL".

Special Considerations: None

3.3.6 *g729abDecoderInit*

Call(s):

```
void g729abDecoderInit(
    OUT     g729ab_sDecoderChannelData *pDecChData
);
```

Required Header: *g729ab.h*

Arguments:

Table 3-4. *g729abDecoderInit()* Arguments

pDecChData	*out*	A pointer to a structure of type *g729ab_sDecoderChannelData*. This structure must be allocated before the function call.

Description: The *g729abDecoderInit()* function initializes the G.729AB decoder algorithm. During initialization, all channel data are set to their initial values in preparation for G.729AB decoder operation. The *g729abDecoderInit()* function must be called only once per channel before the first call to the *g729abDecoder()* function.

Returns: None

Special Considerations: None

3.3.7 *g729abDecoder*

Call(s):

```
void g729abDecoder(
    IN      Word16 *pEncParm,
    OUT     Word16 *pDecodedSpeech,
    IN_OUT  g729ab_sDecoderChannelData *pDecChData
);
```

Required Header: “g729ab.h”

Arguments:

Table 3-5. *g729abDecoder()* Arguments

pEncParm	*in*	Pointer to encoded parameters buffer received; this buffer has G729AB_BITSTREAM_SIZE words
pDecodedSpeech	*out*	Pointer to reconstructed speech buffer, which has G729AB_L_FRAME words. It must be allocated before the call to the *bg29abDecoder* function
pDecChData	*inout*	Pointer to the *g729ab_sEncoderChannelData* structure

Description: The *g729abDecoder()* function must be called for each set of encoded parameters received. It decodes these parameters and provides the corresponding synthesized speech for each frame in PCM format. It uses information from the previous frame, stored in the *g729ab_sDecoderChannelData* structure and updates information needed for the next frame processing in that structure.

In a multichannel scheme, each channel keeps channel information in its own *g729ab_sDecoderChannelData* object.

Returns: None

Special Considerations: None

3.3.8 *g729abDecoderDestroy*

Call(s):

```
void g729abDecoderDestroy(
    IN      g729ab_sDecoderChannelData * pDecChData
);
```

Required Header: *g729ab.h*

Arguments:

Table 3-6. *g729abDecoderDestroy()* Arguments

pDecChData	IN	Pointer to an instance of G.729AB decoder generated by a call to *g729abDecoderCreate()*

Description: The *g729abDecoderDestroy()* function destroys the instance of the G.729AB decoder originally created by a call to *g729abDecoderCreate*().

If the *g729ab_sDecoderChannelData* object was statically created, the *g729abDecoderDestroy()* function should not be called.

Returns: None

Special Considerations: None

Chapter 4
Building the G.729AB Vocoder Library

4.1 Building the G.729AB Vocoder Library

The G.729AB Vocoder Library combines all of the components described in the previous sections into two libraries: *g729ab_Enc.lib* and *g729ab_Dec.lib*. This library's code is not provided with the SDK; therefore, it cannot be built from the SDK. The G.729AB Vocoder Library is provided in the **...\nos\telephony\g729ab** directory of the SDK directory structure.

Chapter 5
Linking Applications with the G.729AB Vocoder Library

5.1 Calling G.729AB Vocoder Library

The G.729AB library provides eight entry points, described in **Chapter 3**. To invoke G.729AB vocoder, the entry points must be called in the order shown in **Table 5-1** and **Table 5-2**.

Table 5-1. Calling Order for Encoder Interfaces

Calling Order	Function Name	Notes
1	*g729abEncoderCreate()*	• Dynamically creates an instance of encoder • Optional; alternatively, an instance can be created statically
2	*g729abEncoderInit()*	• Initializes the encoder instance • Must be called once per channel
3	*g729abEncoder()*	• Encodes a frame of speech • Called multiple times
4	*g729abEncoderDestroy()*	• Destroys an instance created by *g729abEncoderCreate()*

Table 5-2. Calling Order for Decoder Interfaces

Calling Order	Function Name	Notes
1	*g729abDecoderCreate()*	• Dynamically creates an instance of decoder • Optional; alternatively, an instance can be create statically
2	*g729abDecoderInit()*	• Initializes the decoder instance • Must be called once per channel
3	*g729abDecoder()*	• Encodes a frame of speech • Called multiple times
4	*g729abDecoderDestroy()*	• Destroys an instance created by *g729abDecoderCreate()*

5.2 Recommended Memory Map

A sample linker command file, *linker.cmd*, used in the test application for encoder and decoder, is shown in **Code Example 5-1**.

Code Example 5-1. *linker.cmd* File

```
#*******************************************************************************
#
# Linker.cmd file for DSP56858 Internal RAM
#    using only internal program and data memory.
#
#*******************************************************************************

MEMORY {

       .pInterruptVector   (RWX) : ORIGIN = 0x000000, LENGTH = 0x00008C
       .pIntRAM            (RWX) : ORIGIN = 0x00008C, LENGTH = 0x009F74

       .pIntROM            (RX)  : ORIGIN = 0x1F0000, LENGTH = 0x000400

       .xIntRAM            (RW)  : ORIGIN = 0x000100, LENGTH = 0x004F00
       .xStack             (RW)  : ORIGIN = 0x005000, LENGTH = 0x001000

       .xPeripherals       (RW)  : ORIGIN = 0x1FFC00, LENGTH = 0x000400

       .xCoreRegisters     (RW)  : ORIGIN = 0xFFFF00, LENGTH = 0x000100

}

#*******************************************************************************

FORCE_ACTIVE {FconfigInterruptVector}

#*******************************************************************************

SECTIONS {

   #****************************************************************************

    .ApplicationInterruptVector :
       {
              vector.c (.text)

       } > .pInterruptVector

   #****************************************************************************

       .ApplicationCode :
       {
      # Place all code into Program RAM
```

```
        * (.text)
        * (rtlib.text)
        * (fp_engine.text)
        * (user.text)
        * (COMMON.text)
        * (ENCODER.text)
        * (DECODER.text)

        # Place all data into Program RAM

        F_Pdata_start_addr_in_ROM = 0;
        F_Pdata_start_addr_in_RAM = .;
           pramdata.c (.data)
        F_Pdata_ROMtoRAM_length = 0;

     F_Pbss_start_addr = .;

           _P_BSS_ADDR = .;
           pramdata.c (.bss)

        F_Pbss_length = . - _P_BSS_ADDR;

  } > .pIntRAM

#*****************************************************************************

  .ApplicationData :
  {
        # Define variables for C initialization code

        F_Xdata_start_addr_in_ROM = .;
        F_StackAddr               = ADDR(.xStack);
        F_StackEndAddr            = ADDR(.xStack) + SIZEOF(.xStack) - 1;
        F_Xdata_start_addr_in_RAM = .;

           * (.const.data)
           * (.data)
           * (fp_state.data)
           * (rtlib.data)
        * (G729AB_TABLE_LD8A.data)
        * (G729AB_TABLE_DTX.data)


        F_Xdata_ROMtoRAM_length = 0;

        F_Xbss_start_addr = .;
        _X_BSS_ADDR = .;

           * (rtlib.bss.lo)
           * (.bss)

        F_Xbss_length = . - _X_BSS_ADDR;  # Copy DATA

  } > .xIntRAM
```

```
   #****************************************************************************

       FArchIO              = 0x0000;
       FArchCore            = ADDR(.xCoreRegisters);
       FArchInterrupts      = ADDR(.pInterruptVector);

}
```

5.3 Special Requirements

Data sections must be placed in the first 32kwords of memory.

Chapter 6
G.729AB Vocoder Applications

To verify the G.729AB algorithm, test and demo applications have been developed. Refer to the **Targeting Motorola DSP568xx Platform** Manual for the DSP you are using to see if the test and demo applications are available for your target.

Chapter 7
License

7.1 Limited Use License Agreement

LIMITED USE LICENSE AGREEMENT

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THIS SOFTWARE. BY USING OR COPYING THE SOFTWARE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.

The software in either source code form ("Source") or object code form ("Object") (cumulatively hereinafter "Software") is provided under a license agreement ("Agreement") as described herein. Any use of the Software including copying, modifying, or installing the Software so that it is usable by or accessible by a central processing unit constitutes acceptance of the terms of the Agreement by the person or persons making such use or, if employed, the employer thereof ("Licensee") and if employed, the person(s) making such use hereby warrants that they have the authority of their employer to enter this license agreement,. If Licensee does not agree with and accept the terms of this Agreement, Licensee must return or destroy any media containing the Software or materials related thereto, and destroy all copies of the Software.

The Software is licensed to Licensee by Motorola Incorporated ("Motorola") for use under the terms of this Agreement. Motorola retains ownership of the Software. Motorola grants only the rights specifically granted in this Agreement and grants no other rights. Title to the Software, all copies thereof and all rights therein, including all rights in any intellectual property including patents, copyrights, and trade secrets applicable thereto, shall remain vested in Motorola.

For the Source, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to use, copy, and make derivatives of the Source solely in a development system environment in order to produce object code solely for operating on a Motorola semiconductor device having a central processing unit ("Derivative Object").

For the Object and Derivative Object, Motorola grants Licensee a personal, non-exclusive, non-assignable, revocable, royalty-free right to copy, use, and distribute the Object and the Derivative Object solely for operating on a Motorola semiconductor device having a central processing unit.

Licensee agrees to: (a) not use, modify, or copy the Software except as expressly provided herein, (b) not distribute, disclose, transfer, sell, assign, rent, lease, or otherwise make available the Software, any derivatives thereof, or this license to a third party except as expressly provided herein, (c) not remove obliterate, or otherwise defeat any copyright, trademark, patent or proprietary notices, related to the Software (d) not in any form export, re-export, resell, ship or divert or cause to be exported, re-exported, resold, shipped, or diverted, directly or indirectly, the Software or a direct product thereof to any country which the United States government or any agency thereof at the time of export or re-export requires an export license or other government approval without first obtaining such license or approval.

THE SOFTWARE IS PROVIDED ON AN "AS IS" BASIS AND WITHOUT WARRANTY OF ANY KIND INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL MOTOROLA BE LIABLE FOR

ANY LIABILITY OR DAMAGES OF ANY KIND INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT OR INCIDENTAL OR CONSEQUENTIAL OR PUNITIVE DAMAGES OR LOST PROFITS OR LOSS OF USE ARISING FROM USE OF THE SOFTWARE OR THE PRODUCT REGARDLESS OF THE FORM OF ACTION OR THEORY OF LIABILITY (INCLUDING WITHOUT LIMITATION, ACTION IN CONTRACT, NEGLIGENCE, OR PRODUCT LIABILITY) EVEN IF MOTOROLA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THIS DISCLAIMER OF WARRANTY EXTENDS TO LICENSEE OR USERS OF PRODUCTS AND IS IN LIEU OF ALL WARRANTIES WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE.

Motorola does not represent or warrant that the Software is free of infringement of any third party patents, copyrights, trade secrets, or other intellectual property rights or that Motorola has the right to grant the licenses contained herein. Motorola does not represent or warrant that the Software is free of defect, or that it meets any particular requirements or need of the Licensee, or that it conforms to any documentation, or that it meets any standards.

Motorola shall not be responsible to maintain the Software, provide upgrades to the Software, or provide any field service of the Software. Motorola reserves the right to make changes to the Software without further notice to Licensee.

The Software is not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Software could create a situation where personal injury or death may occur. Should Licensee purchase or use the Software for any such unintended or unauthorized application, Licensee shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the Software.

The term of this Agreement is for as long as Licensee uses the Software for its intended purpose and is not in default of any provisions of this Agreement. Motorola may terminate this Agreement if Licensee is in default of any of the terms and conditions of this Agreement.

This Agreement shall be governed by and construed in accordance with the laws of the State of Arizona and can only be modified in a writing signed by both parties. Licensee agrees to jurisdiction and venue in the State of Arizona.

By using, modifying, installing, compiling, or copying the Software, Licensee acknowledges that this Agreement has been read and understood and agrees to be bound by its terms and conditions. Licensee agrees that this Agreement is the complete and exclusive statement of the agreement between Licensee and Motorola and supersedes any earlier proposal or prior arrangement, whether oral or written, and any other communications relative to the subject matter of this Agreement.

Index

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

JAPAN: Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

Technical Information Center: 1–800–521–6274

HOME PAGE: http://www.motorola.com/semiconductors/

MOTOROLA