Comfort Noise Generator (CNG) Algorithm User's Guide

SPIRIT CORP
DSP Software Source
www.spiritDSP.com/CST

Literature Number: SPRU633
March 2003

TEXAS INSTRUMENTS

Preface

Read This First

About This Manual

For the purposes of this Guide, the following abbreviations are used:

CNG	Comfort noise generator
LPC	Linear Predictive Coding (filter)
XDAIS	TMS320 DSP Algorithm Standard

CNG algorithm can be used in conjunction with SPIRIT Corp.'s Voice Activity Detector, which also supports generation of up to 10 LPC coefficients for noise shaping.

Related Documentation From Texas Instruments

Using the TMS320 DSP Algorithm Standard in a Static DSP System (SPRA577)

TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352)

TMS320 DSP Algorithm Standard API Reference (SPRU360)

Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers (SPRA579)

The TMS320 DSP Algorithm Standard (SPRA581)

Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface (SPRA716)

Related Documentation

Voice Activity Detector User's Guide, SPIRIT Corp., 2001

Trademarks

TMS320™ is a trademark of Texas Instruments.

SPIRIT CORP™ is a tradmark of Spirit Corp.

All other trademarks are the property of their respective owners.

Software Copyright

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/products/index.htm
DSP Solutions	http://www.ti.com/dsp
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm
Microcontroller Home Page	http://www.ti.com/sc/micro
Networking Home Page	http://www.ti.com/sc/docs/network/nbuhomex.htm
Military Memory Products Home Page	http://www.ti.com/sc/docs/military/product/memory/mem_1.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(972) 293-5050	Fax: (972) 293-5967	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
Microcontroller Hotline	(281) 274-2370	Fax: (281) 274-4203	Email: micro@ti.com
Microcontroller Modem BBS	(281) 274-3700 8-N-1		
DSP Hotline		Email: dsph@ti.com	
DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs			
Networking Hotline		Fax: (281) 274-4027 Email: TLANHOT@micro.ti.com	

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32
Email: epic@ti.com		
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	
Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/		

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ **Documentation**

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail:	Texas Instruments Incorporated Technical Documentation Services, MS 702 P.O. Box 1443 Houston, Texas 77251-1443	Email: dsph@ti.com Email: micro@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

For product price & availability questions, please contact your local Product Information Center, or see www.ti.com/sc/support http://www.ti.com/sc/support for details.

For additional CST technical support, see the TI CST Home Page (www.ti.com/telephonyclientside) or the TI Semiconductor KnowledgeBase Home Page (www.ti.com/sc/knowledgebase).

If you have any problems with the Client Side Telephony software, please, read first the list of Frequently Asked Questions at http://www.spiritDSP.com/CST.

You can also visit this web site to obtain the latest updates of CST software & documentation.

Contents

Figures

Tables

Notes, Cautions, and Warnings

Chapter 1

Introduction to Comfort Noise Generator (CNG) Algorithms

This chapter is a brief explanation of the Comfort Noise Generator (CNG) and its use with the TMS320C5400 Platform.

For the benefit of users who are not familiar with the TMS320 DSP Algorithm Standard (XDAIS), brief descriptions of typical XDAIS terms are provided.

Topic **Page**

1.1 Introduction

This document describes Comfort Noise Generator (CNG) developed by SPIRIT Corp. for TMS320C54xx platform.

SPIRIT CNG generates noise, distributed either uniformly or shaped according to the spectral envelope coefficients, which can be passed to CNG as parameters (up to 16 LPC coefficients).

It is recommended to use this object in conjunction with SPIRIT Corp.'s Voice Activity Detector, to provide information not only about silence periods, but also to relay noise spectral shape via LPC coefficients. For more information regarding the VAD, please refer to the *Voice Activity Detector (VAD) Algorithm User's Guide* (SPRU635).

The SPIRIT CNG software is a fully TMS320 DSP Algorithm Standard (XDAIS) compatible, reentrant code. CNG interface complies with the TMS320 DSP Algorithm Standard and can be used in multitasking environments.

The TMS320 DSP Algorithm Standard (XDAIS) provides the user with object interface simulating object-oriented principles and asserts a set of programming rules intended to facilitate integration of objects into a framework.

The following documents provide further information regarding the TMS320 DSP Algorithm Standard (XDAIS):

- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (SPRA577)
- *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352)
- *TMS320 DSP Algorithm Standard API Reference* (SPRU360)
- *Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers* (SPRA579)
- *The TMS320 DSP Algorithm Standard* (SPRA581)
- *Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716)

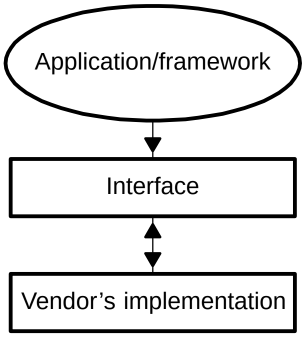
1.2 XDAIS Basics

This section instructs the user on how to develop applications/frameworks using the algorithms developed by vendors. It explains how to call modules through a fully eXpress DSP-compliant interface.

Figure 1-1 illustrates the three main layers required in an XDAIS system:

- ❑ Application/Framework layer
- ❑ Interface layer
- ❑ Vendor implementation. Refer to appendix A for a detailed illustration of the interface layer.

Figure 1-1. XDAIS System Layers



1.2.1 Application/Framework

Users should develop an application in accordance with their own design specifications. However, instance creation, deletion and memory management requires using a framework. It is recommended that the customer use the XDAIS framework provided by SPIRIT Corp. in ROM.

The framework in its most basic form is defined as a combination of a memory management service, input/output device drivers, and a scheduler. For a framework to support/handle XDAIS algorithms, it must provide the framework functions that XDAIS algorithm interfaces expect to be present. XDAIS framework functions, also known as the ALG Interface, are prefixed with “ALG_”. Below is a list of framework functions that are required:

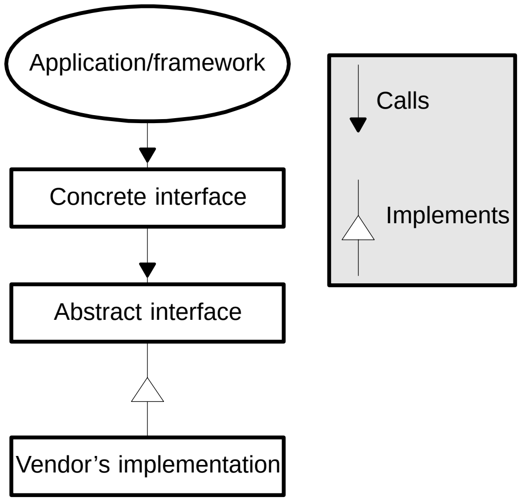
- ❑ `ALG_create` - for memory allocation/algorithm instance creation
- ❑ `ALG_delete` - for memory de-allocation/algorithm instance deletion
- ❑ `ALG_activate` - for algorithm instance activation

- ❑ `ALG_deactivate` - for algorithm instance de-activation
- ❑ `ALG_init` - for algorithm instance initialization
- ❑ `ALG_exit` - for algorithm instance exit operations
- ❑ `ALG_control` - for algorithm instance control operations

1.2.2 Interface

Figure 1-2 is a block diagram of the different XDAIS layers and how they interact with each other.

Figure 1-2. XDAIS Layers Interaction Diagram



1.2.2.1 Concrete Interface

A concrete interface is an interface between the algorithm module and the application/framework. This interface provides a generic (non-vendor specific) interface to the application. For example, the framework can call the function `MODULE_apply()` instead of `MODULE_VENDOR_apply()`. The following files make up this interface:

- ❑ Header file `MODULE.h` - Contains any required definitions/global variables for the interface.
- ❑ Source File `MODULE.c` - Contains the source code for the interface functions.

1.2.2.2 Abstract Interface

This interface, also known as the IALG Interface, defines the algorithm implementation. This interface is defined by the algorithm vendor but must comply with the XDAIS rules and guidelines. The following files make up this interface:

- ❏ Header file `iMODULE.h` - Contains table of implemented functions, also known as the IALG function table, and definition of the parameter structures and module objects.
- ❏ Source File `iMODULE.c` - Contains the default parameter structure for the algorithm.

1.2.2.3 Vendor Implementation

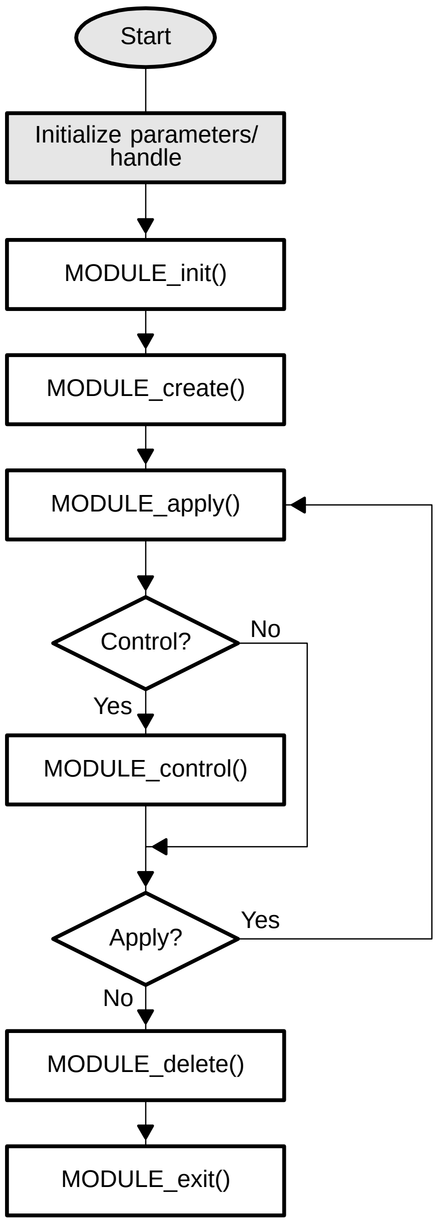
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is known as the function table. The function table allows the framework to invoke any of the algorithm functions through a single handle. The algorithm instance object definition is also done here. This instance object is a structure containing the function table (table of implemented functions) and pointers to instance buffers required by the algorithm.

1.2.3 Application Development

Figure 1-3 illustrates the steps used to develop an application. This flowchart illustrates the creation, use, and deletion of an algorithm. The handle to the instance object (and function table) is obtained through creation of an instance of the algorithm. It is a pointer to the instance object. Per XDAIS guidelines, software API allows direct access to the instance data buffers, but algorithms provided by SPIRIT prohibit access.

Detailed flow charts for each particular algorithm is provided by the vendor.

Figure 1-3. Module Instance Lifetime



The steps below describe the steps illustrated in Figure 1-3.

Step 1: Perform all non-XDAIS initializations and definitions. This may include creation of input and output data buffers by the framework, as well as device driver initialization.

Step 2: Define and initialize required parameters, status structures, and handle declarations.

Step 3: Invoke the `MODULE_init()` function to initialize the algorithm module. This function returns nothing. For most algorithms, this function does nothing.

Step 4: Invoke the `MODULE_create()` function, with the vendor's implementation ID for the algorithm, to create an instance of the algorithm. The `MODULE_create()` function returns a handle to the created instance. You may create as many instances as the framework can support.

Step 5: Invoke the `MODULE_apply()` function to process some data when the framework signals that processing is required. Using this function is not obligatory and vendor can supply the user with his own set of functions to obtain necessary processing.

Step 6: If required, the `MODULE_control()` function may be invoked to read or modify the algorithm status information. This function also is optional. Vendor can provide other methods for status reporting and control.

Step 7: When all processing is done, the `MODULE_delete()` function is invoked to delete the instance from the framework. All instance memory is freed up for the framework here.

Step 8: Invoke the `MODULE_exit()` function to remove the module from the framework. For most algorithms, this function does nothing.

The integration flow of specific algorithms can be quite different from the sequence described above due to several reasons:

❑ Specific algorithms can work with data frames of various lengths and formats. Applications can require more robust and effective methods for error handling and reporting.

❑ Instead of using `the MODULE_apply()` function, SPIRIT Corp. algorithms use extended interface for data processing, thereby encapsulating data buffering within XDAIS object. This provides the user with a more reliable method of data exchange.

Chapter 2

Comfort Noise Generator (CNG) Integration

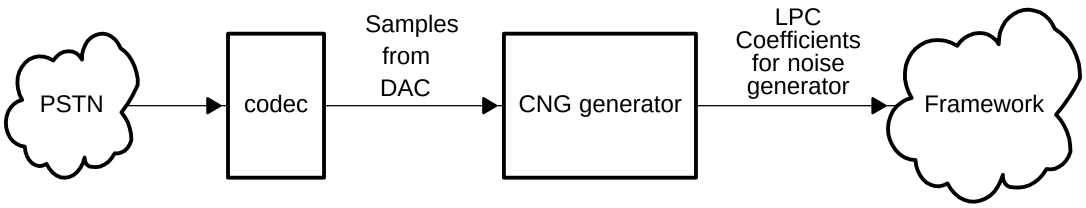
This chapter provides descriptions, diagrams, and examples explaining the integration of the Comfort Noise Generator with frameworks.

Topic **Page**

2.1 Overview

Figure 2-1 illustrates a typical CNG integration diagram.

Figure 2-1. CNG Integration Diagram



The CNG generator produces output signal according to values of LPC coefficients and Seed.

2.2 Integration Flow

In order to integrate the CNG generator into a framework, the user should:(Figure 2-2):

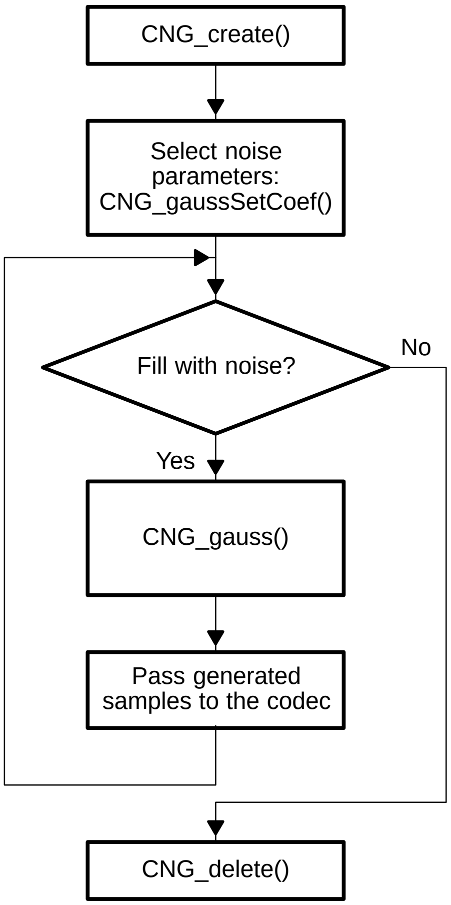
Step 1: Call `CNG_create()` to create the instance of a generator with specified parameters.

Step 2: Call `CNG_gaussSetCoef()` to choose required noise characteristics by setting LPC filter coefficients. This step can be skipped when white noise should be generated.

Step 3: Call `CNG_gauss()` to generate noise.

Step 4: Delete the generator by using `CNG_delete()`.

Figure 2-2. Typical CNG Integration Flow


CNG_create()
Select noise parameters: CNG_gaussSetCoef()
Fill with noise?
No
Yes
CNG_gauss()
Pass generated samples to the codec
CNG_delete()

2.3 Example of a Call Sequence

The example below demonstrates a typical call sequence for a CNG generator. Full sample code is placed in the file `Src\FlexExamples\StandaloneXDAS\CNG\main.c`.

```
XDAS_Void GenerateNoise(XDAS_Int16*pBuf, XDAS_Int16 BUFsize,
  XDAS_Int16*pLPC, XDAS_Int16 LPCsize, XDAS_Int16 magnitude)
{
  CNG_Handle CNGInst;
  /* creating CNG instance with default parameters */
  CNGInst = CNG_create(&CNG_SPCORP_ICNG, NULL);
  /* set filter coefficients */
  CNG_gaussSetCoef(CNGInst, pLPC, LPCsize, Magnitude);
  /* generate noise */
  CNG_gauss(CNGInst, pBuf, BUFsize);
  /* Deleting CNG instance */
  CNG_delete(CNGInst);
}
```

Chapter 3

Comfort Noise Generator (CNG) API Descriptions

This chapter provides the user with a clear understanding of Comfort Noise Generator (CNG) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).

Topic **Page**

3.1 Standard Interface Structures

In this section, parameter structures are described.

3.1.1 Instance Creation Parameters

Description Not used

3.1.2 Status Structure

Description This structure defines the status parameters for the algorithm. Generator status structure is used for control purposes. Status can be received by function `CNG_getStatus().`

Structure Definition

Table 3-1. CNG Generator Real-Time Status Parameters

```
typedef struct ICNG_Status {
```

Status Type	Status Name	Description
`Int`	`size`	ignored

```
 } ICNG_Status;
```

Type `ICNG_Status` defined in “`iCNG.h`”.

3.2 Standard Interface Functions

The CNG functions in this section are required when using the algorithm CNG.

`CNG_apply()` and `CNG_control()` are optional, but neither are supported by Spirit Corp.

Table 3-2 summarizes standard Interface functions of CNG API.

Table 3-2. CNG Standard Interface Functions

Functions	Description	See Page...
`CNG_init`	Calls the framework initialization function (Algorithm initialization)	3-3
`CNG_exit`	Calls the framework exit function (Algorithm deletion)	3-4
`CNG_create`	Calls the framework creation function (Instance creation)	3-4
`CNG_delete`	Calls the framework deletion function (Instance deletion)	3-5

3.2.1 Algorithm Initialization

CNG_init *Calls the framework initialization function to initialize an algorithm*

Description This function calls the framework initialization function, `ALG_init()`, to initialize the algorithm. For CNG generator, this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype `void CNG_init()`

Arguments none

Return Value none

3.2.2 Algorithm Deletion

CNG_exit *Calls the framework exit function to remove an algorithm*

Description

This function calls the framework exit function, `ALG_exit()`, to remove the algorithm. For CNG generator, this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype

```
void CNG_exit()
```

Arguments

none

Return Value

none

3.2.3 Instance Creation

CNG_create *Calls the framework creation function to create an algorithm*

Description

In order to create a new CNG generator object, `CNG_create` function should be called. This function calls the framework create function, `ALG_create()`, to create the instance object and perform memory allocation tasks. Global structure `CNG_SPCORP_ICNG` contains `CNG` virtual table supplied by SPIRIT Corp.

Function Prototype

```
CNG_Handle CNG_create
      (const ICNG_Fxns *fxns,
       const CNG_Params *prms);
```

Arguments

`ICNG_Fxns *`	Pointer to vendor's functions (Implementation ID). Use reference to `CNG_SPCORP_ICNG` virtual table supplied by SPIRIT Corp.
`CNG_Params *`	Pointer to Parameter Structure. Use `NULL` pointer to load default parameters.

Return Value

`CNG_Handle`	Defined in file "`CNG.h`". This is a pointer to the created instance.

3.2.4 Instance Deletion

CNG_delete *Calls a framework delete function to delete an instance object*

Description This function calls the framework delete function, `ALG_delete()`, to delete the instance object and perform memory de-allocation tasks.

Function Prototype `void CNG_delete (CNG_Handle handle)`

Arguments `CNG_Handle` Instance’s handle obtained from `CNG_create()`

Return Value none

3.3 Vendor-Specific Interface Functions

In this section, functions in the SPIRIT's algorithm implementation and interface (extended IALG methods) are described.

Table 3-3 summarizes SPIRIT's API functions of CNG generator.

The whole interface is placed in header files `iCNG.h`, `CNG.h`, `CNG_spcorp.h`.

Table 3-3. Generator-Specific Interface Functions

Functions	Description	See Page...
`CNG_gaussInit`	Initialize CNG with specified parameters.	3-6
`CNG_gauss`	Noise generation	3-7
`CNG_gaussSetCoef`	Setting LPC coefficients	3-7

3.3.1 CNG Initialization

CNG_gaussInit *Reinitializes a CNG instance and sets it to its initial state*

Description

Call this function to reinitialize CNG instance and set it into the initial state. This function can be called in any time you need.

Function Prototype

```
XDAS_Void CNG_gaussInit
      (CNG_Handle handle,
       XDAS_Int16 seed)
```

Arguments

handle	Pointer to a CNG instance
seed	Random number generator seed value. Any value can be accepted.

Return Value none

Restrictions none

3.3.2 Noise Generation

CNG_gauss *Generates noise samples and stores them in a buffer*

Description Generates a number of noise samples and stores it in buffer.

Function Prototype

```
XDAS_Void CNG_gauss
      (ICNG_Handle handle,
 XDAS_Int16 *pOut,
 XDAS_Int16 size)
```

Arguments

handle	Pointer to CNG instance
*p0ut	Pointer to buffer to fill with noise
size	Size of buffer

Return Value none

Restrictions Maximal length of buffer is 32767.

3.3.3 Setting LPC Coefficients

CNG_gaussSet-Coef *Sets the actual LPC coefficients stored in an internal buffer*

Description Invoke this function for setting actual LPC coefficients. Coefficients are stored in the internal buffer, so host can change or remove them immediately after this function returns.

Function Prototype

```
XDAS_Void CNG_gaussSetCoef
      (CNG_Handle handle,
       const XDAS_Int16 *pLPCCoef,
       XDAS_Int16 size,
       XDAS_Int16 magnitude)
```

Arguments

handle	Pointer to CNG instance
*pLP CCoef	Pointer to array with LPC coefficients
size	Size of buffer
magnitude	Output magnitude (0..32765)

Return Value Returns 1 on success and 0 if specified number of LPC coefficients exceeds maximal allowed value (16).

Restrictions Number of LPC coefficents should not exceed 16.

Appendix A

Test Environment

Note: Test Environment Location

This chapter describes test environment for the CNG object.

For TMS320C54CST device, test environment for standalone CNG object is located in the Software Development Kit (SDK) in `Src\FlexExamples\StandaloneXDAS\CNG`.

Topic **Page**

A.1 Description of Directory Tree

The SDK package includes the test project “test.pjt” and corresponding reference test vectors. The user is free to modify this code as needed, without submissions to SPIRIT Corp.

Table A-1. Test Files for CNG

File	Description
main.c	Test file
FileC5x.c	File input/output functions
..\ROM\CSTRom.s54	ROM entry address
Test.cmd	Linker command file
Vectors\output.pcm	Reference output test vectors

A.1.1 Test Vectors Format

All test vectors are raw PCM files with following parameters:

- Bits per sample - 16, Mono
- Word format - Intel PCM (LSB goes first)
- Encoding - uniform
- Level - -2 dBF

A.1.2 Test Project

To build and run a project, the following steps must be performed:

Step 1: Open the project: `Project\Open`

Step 2: Build all necessary files: `Project\Rebuild All`

Step 3: Initialize the DSP: `Debug\Reset CPU`

Step 4: Load the output-file: `File\Load program`

Step 5: Run the executable: `Debug\Run`

Once the program finishes testing, the file *Output.pcm* will be written in the current directory. Compare this file with the reference vector contained in the directory *Vectors*.

Note: Test Duration

Since the standard file I/O for EVM is very slow, testing may take several minutes. Test duration does not indicate the real algorithm's throughput.

Index