Echo Canceller (EC) Algorithm User's Guide

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

Literature Number: SPRU634
March 2003

Texas Instruments

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright © 2003, Texas Instruments Incorporated

Preface

Read This First

About This Manual

The following abbreviations are used in this document:

XDAIS	TMS320 DSP Algorithm Standard
EC	Echo Canceller
DT	Double talk detector
NLP	Nonlinear processor
STA	Short time averaging
LTA	Long time averaging

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

ITU-T Recommendation G.165 (03/93). General characteristics of international telephone connections and international telephone circuits. Echo Cancellers.

ITU-T Recommendation G.168 (04/97). Digital network echo cancellers

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

CST Software Copyright © 2003, SPIRIT Technologies, Inc.

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 Echo Canceller (EC) Algorithms

This chapter is a brief explanation of the Echo Canceller (EC) 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 implementation of Electric Echo Canceller developed by SPIRIT Corp. for TMS320C54xx platform, PCM-oriented version, and intended for integration into digital networks, embedded equipment, etc.

This Echo Canceller is used for cancellation of echo created by telephone hybrid, and conforms to G.165 and G.168 ITU recommendations. User can set the value of maximum echo path equal to 16, 32, or 64 ms. Input/output signal range should be within standard PCM samples range.

The SPIRIT EC software is a fully TMS320 DSP Algorithm Standard (XDAIS) compatible, reentrant code. The EC 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)

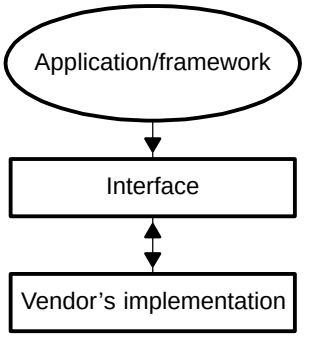
1.2 XDAIS Basics

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:

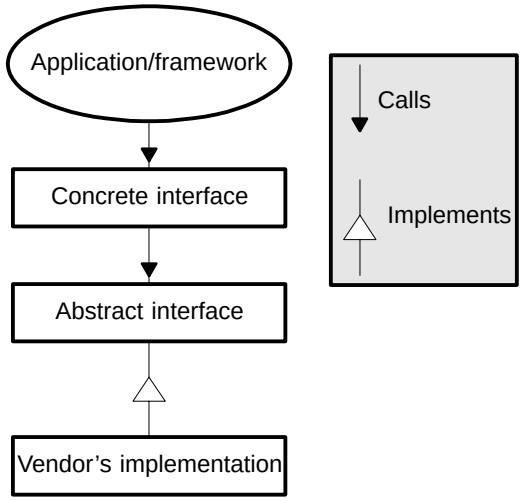
- `ALG_create` - for memory allocation/algorithm instance creation
- `ALG_delete` - for memory de-allocation/algorithm instance deletion
- `ALG_activate` - for algorithm instance activation

- 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

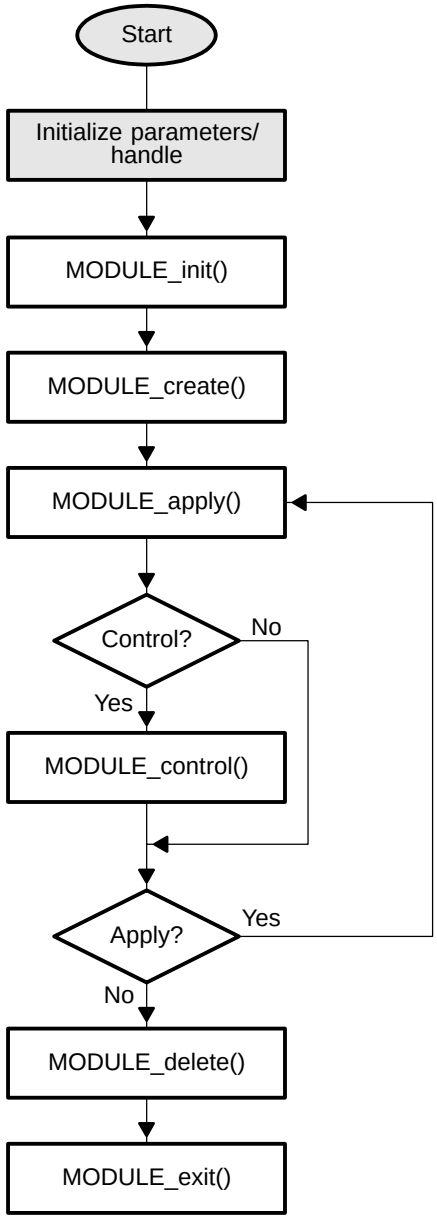
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is 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.

1.3 Limitations

Table 1-1 lists the limitations for this version of SPIRIT Corp. Echo Canceller.

Table 1-1. Limitations for Echo Canceller

Feature	Limitation
Supported ITU recommendation	G.168 and G.165
Tone Disabler	Not inside Echo Canceller object. Can be added as an external object.
Comfort noise generator	Not inside Echo Canceller object. Can be added as an external object.
Maximum echo path delay	16, 32, and 64 ms
Input/output signal range	PCM range (14-bit)

Chapter 2

Echo Canceller Integration

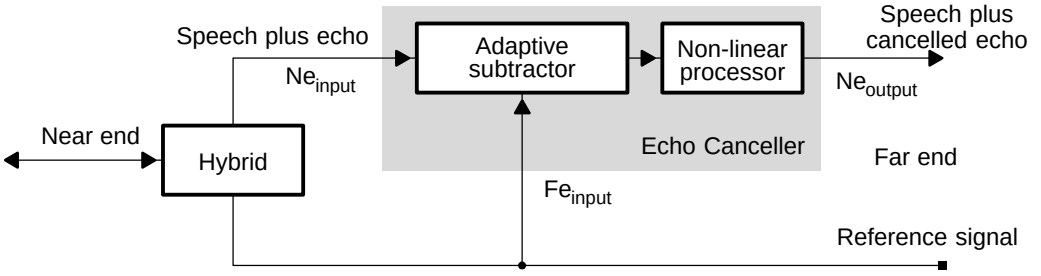
This chapter provides input signal requirements, descriptions, diagrams, and examples explaining the integratino of the Echo Canceller (EC) with frameworks.

Topic **Page**

2.1 Overview

Figure 2-1 shows Echo Canceller flowchart.

Figure 2-1. Echo Canceller Diagram



Echo Canceller is used for cancellation of echoes created by a telephone hybrid. In **Fe** denotes far end speech (reference signal), **Ne input** denotes echo with near end speech, **Ne output** denotes near end speech without echo.

2.2 Input Signals Requirements

As mentioned in section 2.1, the purpose of Echo Canceller is to cancel echoes caused by hybrid connections in telephone networks. This can be achieved if the Echo Canceller's input signals conform to specific requirements. The User must follow them, otherwise, strong Echo Canceller performance degradation may be experienced.

The requirements are as follows:

- ❑ Echo signal (Ne) must be delayed relatively to reference signal (Fe).

Note: Echo Signal Delay

If it is only possible to obtain an echo signal with additional fixed delays, compensate by delaying reference signals of the same value before performing echo cancellations.

- ❑ Maximum echo delay must not exceed currently selected EC maximum echo path delay defined by `IEC_Tail`.
- ❑ Input and output samples should be linear PCM samples with absolute values less than 8159 (this is the maximum value for linear samples after μ-law expansion)
- ❑ The average amplitude of a reference signal (Fe) must not increase by more than 5000.
- ❑ The average amplitude of an echo signal must not exceed half the average amplitude of a reference signal (Fe).
- ❑ A Reference and echo signal must not have any DC offset.
- ❑ The Echo canceller is sensitive to non-linear distortions (clipping, etc.) in input signals. The presence of more distortions result in further EC performance degradation.
- ❑ Echo canceller is designed to cancel an echo caused by a hybrid connection in telephone networks that changes slowly, therefore it would not be able to track rapid large continuous changes in an echo path.

2.3 Integration Flow

For integration of Echo Canceller into user Framework it is necessary to perform the following steps:

Step 1: Create an Echo Canceller object.

To accomplish this, it is necessary to invoke the `EC_Handle EC_create(const IEC_Fxns *fxns, const EC_Params *prms)` routine.

The first parameter of this function is a pointer to table of EC virtual methods `&EC_SPCORP_IEC`, and the second parameter is a pointer to structure of EC parameters (this structure is described below).

This routine returns a pointer to the created EC object.

Step 2: Run the Echo Canceller.

To accomplish this, it is necessary to periodically invoke the `EC_process(EC_Handle handle, Int *fe, Int *ne, Int size)` routine, each `size/8` ms.

The first parameter of this function is a pointer to EC object, the second parameter is a pointer to reference (Far end) speech samples, the third parameter is a pointer to echo with near end speech samples, the fourth parameter is the number of samples to be processed.

The third parameter `ne` is also used as a pointer to output near end speech samples without echo.

Thus, after the call to `EC_process()` routine, the near end speech input samples with echo are replaced by near speech samples without echo.

Step 3: At the end of operation, delete Echo Canceller object.

To accomplish this, it is necessary to invoke the `Void EC_delete(EC_Handle handle)` routine.

The first parameter of this function is a pointer to the object to be deleted.

2.4 Integration Example

The following code example demonstrates Echo Canceller integration:

```
extern void OpenFiles(void);
extern void CloseFiles(void);
/* Create a pointer to EC object */
EC_Handle Ecn;
void main(void)
{
  int16 fe[BUFFER_LENGTH];
  int16 ne[BUFFER_LENGTH];
  /* Open files ref, echo, res with reference, echo and cleared speech */
  OpenFiles();
  /*Create EC object for maximum echo path delay 32 ms */
  /* (255 Taps of adaptive subtracting flter) */
  Ecn=EC_create(&EC_SPCORP_IEC, &IEC_PARAMS);
  while(1)
  {
    if(fread_(&fe[0],sizeof(int16),BUFFER_LENGTH,ref)!=BUFFER_LENGTH)
      break;
    if(fread_(&ne[0],sizeof(int16),BUFFER_LENGTH,echo)!=BUFFER_LENGTH)
      break;
    /* Process BUFFER_LENGTH samples */
    EC_process(Ecn,&fe[0],&ne[0],BUFFER_LENGTH);

    /*Save result */
    fwrite_(&ne[0],sizeof(int16),BUFFER_LENGTH,res);
  }
  /* Delete Ec object */
  EC_delete(Ecn);
  /* Close all files */
  CloseFiles();
}
```

Chapter 3

Echo Canceller (EC) API Descriptions

This chapter provides the user with a clear understanding of Echo Canceller (EC) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).

Topic **Page**

3.1 Standard Interface Structures

The section describes parameters, status structures and standard methods for the Echo Canceller.

Table 3-1 lists the type and location of the Standard Interface structures.

Table 3-1. Standard Interface Structures Summary

Parameters	Located in Table...
Echo Canceller Run-Time Creation	Table 3-2
IEC_Tail Enumerator Definition	Table 3-3
Bit Definition in `ecState` Word	Table 3-4
EC Status Structure	Table 3-5

3.1.1 Parameters Structure

Description This structure is used for creation of Echo Canceller object. A default parameter structure is defined in `“iec.c”` file

Structure Definition

Table 3-2. Echo Canceller Run-Time Creation Parameters

```
typedef struct IEC_Params{
```

Parameter Type	Parameter Name	Description	Value
`Int`	`ecState`	Default echo canceller state	Described in
`IEC_Tail`	`amountOfTaps`	This enumeration (see) determines amount of taps of adaptive subtractor (maximum echo path delay)	Described in
`Int`	`nlpMin`	This value determines level of nonlinear suppression	From 8159 to 0
`Int`	`nlpShift`	This value determines sluggishness of nonlinear processor	From 8 to 15

```
} IEC_Params
```

Table 3-3. IEC_Tail Enumerator Definition

```
typedef enum {
```

Value	Description
IEC_TAIL127	Maximum echo delay path equals to 16 ms (127 Taps)
IEC_TAIL255	Maximum echo delay path equals to 32 ms (255 Taps)
IEC_TAIL511	Maximum echo delay path equals to 64 ms (511 Taps)

```
} IEC_Tail
```

Table 3-4. Bit Definition in `ecState` Word

Value	Description
IEC_DTDETECTED	Double talk is detected
IEC_ECCONVERGED	Echo canceller is converged now
IEC_NLPENABLE	Nlp is enabled
IEC_DTENABLE	Dt is enabled
IEC_UPDATEENABLE	Adaptation is enabled
IEC_FEACTIVE	Reference(Far) end is active
IEC_NEACTIVE	Near end is active
IEC_BIGERR	Double talk with low level is detected

Type IEC_PARAMS is defined in “iec.h” file.

3.1.2 Status Structure

Description This structure is used for Echo Canceller state control. Call `EC_getStatus()` function to obtain this structure.

Structure Definition

Table 3-5. EC Status Structure

```
typedef struct IEC_Status {
```

Status Type	Status Name	Description
`Int`	`ecState`	EC state
`Int`	`feSTA`	Far end level for short averaging
`Int`	`neSTA`	Near end level for short averaging
`Int`	`errSTA`	Residual echo level for short averaging
`Int`	`feLTA`	Far end level for long averaging
`Int`	`neLTA`	Near end level for long averaging
`Int`	`errLTA`	Residual echo level for long averaging
`Int *`	`pTaps`	Pointer to Taps array of adaptive subtractor

```
} IEC_Status;
```

Type `IEC_Status` is defined in "`iec.h`" file.

3.2 Standard Interface Functions

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

EC_`apply()` and `EC_control()` are optional, but neither are supported by Spirit Corp.

Table 3-6 summarizes the standard Interface functions of the Echo Canceller API.

Table 3-6. Echo Canceller Standard Interface Functions

Functions	Description	See Page...
`EC_create`	Instance creation	3-5
`EC_delete`	Instance deletion	3-6
`EC_init`	Algorithm initialization	3-6
`EC_exit`	Algorithm deletion	3-6

3.2.1 Instance Creation

EC_create *Calls the framework create function to create an instance object*

Description In order to create a new Echo Canceller object, `EC_create`() function should be called. This function calls the framework create function, `ALG_create()`, to create the instance object and perform memory allocation tasks. Also the function `ALG_create()` performs Echo Canceller initialization in accordance with the parameter structure. Global structure `EC_SPCORP_IEC` contains echo canceller virtual table supplied by SPIRIT Corp.

Function Prototype

```
EC_Handle EC_create(const IEC_Fxns *fxns, const EC_Params *prms)
```

Arguments

`IEC_Fxns *fxn`	Pointer to virtual method table of Echo Canceller. Use reference to `EC_SPCORP_IEC` virtual table supplied by SPIRIT Corp.
`EC_Params *prms`	Pointer to the parameter structure. Fields of this structure are described above. Use `NULL` pointer to load default parameters.

Return Value This routine returns pointer to Echo Canceller object.

3.2.2 Instance Deletion

EC_delete *Calls the 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 EC_delete(EC_Handle handle)`

Arguments `EC_Handle handle` Pointer to Echo Canceller object.

Return Value none

3.2.3 Algorithm Initialization

EC_init *Calls the framework initialization function to initialize the algorithm EC*

Description This function calls the framework initialization function, `ALG_init()`, to initialize the algorithm `EC`. Usually, 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 EC_init0()`

Arguments none

Return Value none

3.2.4 Algorithm Deletion

EC_exit *Calls the framework exit function ALG_exit()*

Description This function calls the framework exit function, `ALG_exit()`, to remove the algorithm `EC`.

Usually, 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 EC_exit()`

Arguments none

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 3-7 summarizes SPIRIT's API functions of Echo canceller.

The whole interface is located in header files `iec.h`, `ec.h`, `ec_spcorp.h`.

Table 3-7. Echo Canceller Vendor-Specific Interface Functions

Functions	Description	See Page...
`EC_process ()`	Process samples	3-7
`EC_getStatus()`	Status reporting	3-8
`EC_changeState()`	Status changing	3-8

3.3.1 Status Reporting

EC_getStatus *Gets the status structure from an Echo Canceller object*

Description The routine `EC_getStatus()` is used to get status structure from Echo Canceller object. Fields of this structure are described above in Table 3-5.

Function Prototype `XDAS_Void EC_getStatus(EC_Handle handle, EC_Status *status)`

Arguments

`EC_Handle handle` Pointer to Echo Canceller object
`EC_Status *status` Pointer to Echo Canceller status structure

Return Value This routine fills the fields of Echo Canceller status structure

Restrictions none

3.3.2 Process Samples

EC_process *Processes samples from far and near end*

Description

This routine is invoked each `size/8` ms for processing `size` samples from far and near end. The first parameter of this function is a pointer to EC object, the second parameter is a pointer to reference (Far end) speech samples, the third parameter is a pointer to echo with near end speech samples, the fourth parameter is number of samples to be processed. The third parameter `ne` also is used as a pointer to output near end speech samples without echo. Thus, after the call to `EC_process()` routine, the near end speech input samples with echo are replaced by near speech samples without echo. The sampling frequency for input and output samples is 8000 Hz.

Function Prototype

```
XDAS_Void  EC_process(EC_Handle  handle,  Int  *fe,Int
*ne,Int size)
```

Arguments

`EC_Handle handle`	Pointer to Echo Canceller object
`Int *fe`	Pointer to reference (Far end) speech samples
`Int *ne`	Pointer to echo with near end speech samples
`Int size`	Amount of samples to be processed

Return Value

This routine returns near end speech samples without echo. These samples replace input near end speech samples with echo.

3.3.3 State Changing

EC_changeState *Changes current Echo Canceller state*

Description

The routine `EC_changeState()` is used for changing of current Echo Canceller state.

Function Prototype

```
XDAS_Void  EC_changeState(
  EC_Handle handle,
  Int state,
  IEC_SetState setResetFlag
)
```

Arguments

`EC_Handle handle`	Pointer to Echo Canceller object
`Int state`	Current state of Echo Canceller. Bits of EC state word are described above
`IEC_SetState setResetFlag`	This flag tells whether bits are cleared or set in EC state word in accordance with bits determined in second parameter `state`

Table 3-8 determines enumerator `IEC_SetState`.

Table 3-8. IEC_SetState Enumerator

```
typedef enum IEC_SetState{
```

Value	Description
IEC_SETBITSINSTATEWORD	Set bits equal to 1 in the parameter state, in current Echo Canceller state word.
IEC_CLEARBITSINSTATEWORD	Clear bits equal to 1 in parameter state, in current Echo Canceller state word.

```
} IEC_SetState0
```

Return Value none

Restrictions none

Appendix A

Test Environment

Note: Test Environment Location

This chapter describes test environment for the EC object.

For TMS320C54CST device, test environment for standalone EC object is located in the Software Development Kit (SDK) in `Src\FlexExamples\StandaloneXDAS\G.168`.

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 EC

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:

- Test vector `ref_sp.bin` includes far end (reference) speech samples
- Test vector `echo_sp.bin` includes near end speech samples wioth far end echo.
- Test vector `res.bin` includes near end speech samples.

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

M

P

S

T

X