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
About This Manual

The following abbreviations are used in this document:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDAIS</td>
<td>TMS320 DSP Algorithm Standard</td>
</tr>
<tr>
<td>EC</td>
<td>Echo Canceller</td>
</tr>
<tr>
<td>DT</td>
<td>Double talk detector</td>
</tr>
<tr>
<td>NLP</td>
<td>Nonlinear processor</td>
</tr>
<tr>
<td>STA</td>
<td>Short time averaging</td>
</tr>
<tr>
<td>LTA</td>
<td>Long time averaging</td>
</tr>
</tbody>
</table>

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

**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](http://www.ti.com)
  - DSP Solutions: [http://www.ti.com/dsp](http://www.ti.com/dsp)
  - 320 Hotline On-line: [http://www.ti.com/sc/docs/dsp320/support.htm](http://www.ti.com/sc/docs/dsp320/support.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)
  - DSP Hotline: +03-3457-0972 or (INTL) 813-3457-0972, Fax: +03-3457-1259 or (INTL) 813-3457-1259
  - DSP BBS via Nifty-Serve: Type “go TIASP”
If You Need Assistance

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

1 **Introduction to Echo Canceller (EC) Algorithms** ............................................. 1-1
   *This chapter is a brief explanation of Echo Canceller (EC) and its use with the TMS320C5400 platform.*
   1.1 Introduction ........................................................................................................... 1-2
   1.2 XDAIS Basics ....................................................................................................... 1-3
       1.2.1 Application/Framework ............................................................................... 1-3
       1.2.2 Interface ...................................................................................................... 1-4
       1.2.3 Application Development ......................................................................... 1-5
   1.3 Limitations ............................................................................................................ 1-8

2 **Echo Canceller Integration** .................................................................................. 2-1
   *This chapter provides input signal requirements, descriptions, diagrams, and examples explaining the integration of the Echo Canceller with frameworks.*
   2.1 Overview ............................................................................................................ 2-2
   2.2 Input Signals Requirements ............................................................................... 2-3
   2.3 Integration Flow .................................................................................................. 2-4
   2.4 Integration Example ............................................................................................. 2-5

3 **Echo Canceller (EC) API Descriptions** ............................................................. 3-1
   *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).*
   3.1 Standard Interface Structures ............................................................................. 3-2
       3.1.1 Parameters Structure .................................................................................. 3-2
       3.1.2 Status Structure ......................................................................................... 3-4
   3.2 Standard Interface Functions ............................................................................ 3-5
       3.2.1 Instance Creation ......................................................................................... 3-5
       3.2.2 Instance Deletion ........................................................................................ 3-6
       3.2.3 Algorithm Initialization ............................................................................... 3-6
       3.2.4 Algorithm Deletion ...................................................................................... 3-6
   3.3 Vendor-Specific Interface Functions .................................................................. 3-7
       3.3.1 Status Reporting ......................................................................................... 3-7
       3.3.2 Process Samples .......................................................................................... 3-8
       3.3.3 State Changing ........................................................................................... 3-8

A **Test Environment** ............................................................................................... A-1
   A.1 Description of Directory Tree ............................................................................ A-2
Contents

A.1.1 Test Vectors Format .................................................... A-2
A.1.2 Test Project ............................................................. A-3
Figures

1-1 XDAIS System Layers ................................................................. 1-3
1-2 XDAIS Layers Interaction Diagram ............................................. 1-4
1-3 Module Instance Lifetime ........................................................... 1-6
2-1 Echo Canceller Diagram .............................................................. 2-2

Tables

1-1 Limitations for Echo Canceller .................................................... 1-8
3-1 Standard Interface Structures Summary ....................................... 3-2
3-2 Echo Canceller Run-Time Creation Parameters ............................ 3-2
3-3 IEC_Tail Enumerator Definition ................................................... 3-3
3-4 Bit Definition in ecState Word ..................................................... 3-3
3-5 EC Status Structure ................................................................. 3-4
3-6 Echo Canceller Standard Interface Functions .............................. 3-5
3-7 Echo Canceller Vendor-Specific Interface Functions ...................... 3-7
3-8 IEC_SetState Enumerator .......................................................... 3-9
A-1 Test Files for EC ................................................................. A-2

Notes, Cautions, and Warnings

Echo Signal Delay ................................................................. 2-3
Test Environment Location .......................................................... A-1
Test Duration ................................................................. A-3
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.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Introduction</td>
<td>1-2</td>
</tr>
<tr>
<td>1.2 XDAIS Basics</td>
<td>1-3</td>
</tr>
<tr>
<td>1.3 Limitations</td>
<td>1-8</td>
</tr>
</tbody>
</table>
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](image)

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
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.
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

<table>
<thead>
<tr>
<th>Feature</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported ITU recommendation</td>
<td>G.168 and G.165</td>
</tr>
<tr>
<td>Tone Disabler</td>
<td>Not inside Echo Canceller object. Can be added as an external object.</td>
</tr>
<tr>
<td>Comfort noise generator</td>
<td>Not inside Echo Canceller object. Can be added as an external object.</td>
</tr>
<tr>
<td>Maximum echo path delay</td>
<td>16, 32, and 64 ms</td>
</tr>
<tr>
<td>Input/output signal range</td>
<td>PCM range (14-bit)</td>
</tr>
</tbody>
</table>
This chapter provides input signal requirements, descriptions, diagrams, and examples explaining the integration of the Echo Canceller (EC) with frameworks.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Overview</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2 Input Signals Requirements</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3 Integration Flow</td>
<td>2-4</td>
</tr>
<tr>
<td>2.4 Integration Example</td>
<td>2-5</td>
</tr>
</tbody>
</table>
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:

```c
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 filter) */
    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).

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Standard Interface Structures</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2 Standard Interface Functions</td>
<td>3-5</td>
</tr>
<tr>
<td>3.3 Vendor-Specific Interface Functions</td>
<td>3-7</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Located in Table...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo Canceller Run-Time Creation</td>
<td>Table 3-2</td>
</tr>
<tr>
<td>IEC_Tail Enumerator Definition</td>
<td>Table 3-3</td>
</tr>
<tr>
<td>Bit Definition in ecState Word</td>
<td>Table 3-4</td>
</tr>
<tr>
<td>EC Status Structure</td>
<td>Table 3-5</td>
</tr>
</tbody>
</table>

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{

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

} 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEC_DTDETECTED</td>
<td>Double talk is detected</td>
</tr>
<tr>
<td>IEC_ECCONVERGED</td>
<td>Echo canceller is converged now</td>
</tr>
<tr>
<td>IEC_NLPENABLE</td>
<td>Nlp is enabled</td>
</tr>
<tr>
<td>IEC_DTENABLE</td>
<td>Dt is enabled</td>
</tr>
<tr>
<td>IEC_UPDATEENABLE</td>
<td>Adaptation is enabled</td>
</tr>
<tr>
<td>IEC_FEACTIVE</td>
<td>Reference(Far) end is active</td>
</tr>
<tr>
<td>IEC_NEACTIVE</td>
<td>Near end is active</td>
</tr>
<tr>
<td>IEC_BIGERR</td>
<td>Double talk with low level is detected</td>
</tr>
</tbody>
</table>

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 {

<table>
<thead>
<tr>
<th>Status Type</th>
<th>Status Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>ecState</td>
<td>EC state</td>
</tr>
<tr>
<td>Int</td>
<td>feSTA</td>
<td>Far end level for short averaging</td>
</tr>
<tr>
<td>Int</td>
<td>neSTA</td>
<td>Near end level for short averaging</td>
</tr>
<tr>
<td>Int</td>
<td>errSTA</td>
<td>Residual echo level for short averaging</td>
</tr>
<tr>
<td>Int</td>
<td>feLTA</td>
<td>Far end level for long averaging</td>
</tr>
<tr>
<td>Int</td>
<td>neLTA</td>
<td>Near end level for long averaging</td>
</tr>
<tr>
<td>Int</td>
<td>errLTA</td>
<td>Residual echo level for long averaging</td>
</tr>
<tr>
<td>Int *</td>
<td>pTaps</td>
<td>Pointer to Taps array of adaptive subtractor</td>
</tr>
</tbody>
</table>

} 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>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>See Page...</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_create</td>
<td>Instance creation</td>
<td>3-5</td>
</tr>
<tr>
<td>EC_delete</td>
<td>Instance deletion</td>
<td>3-6</td>
</tr>
<tr>
<td>EC_init</td>
<td>Algorithm initialization</td>
<td>3-6</td>
</tr>
<tr>
<td>EC_exit</td>
<td>Algorithm deletion</td>
<td>3-6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>See Page...</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC_process ()</td>
<td>Process samples</td>
<td>3-7</td>
</tr>
<tr>
<td>EC_getStatus()</td>
<td>Status reporting</td>
<td>3-8</td>
</tr>
<tr>
<td>EC_changeState()</td>
<td>Status changing</td>
<td>3-8</td>
</tr>
</tbody>
</table>

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
EC_process

3.3.2 Process Samples

**EC_process** *Processes samples from far and near end*

**Description**
This routine is invoked each \( \frac{\text{size}}{8} \) ms for processing \( \text{size} \) 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 \( \text{ne} \) also is used as a pointer to output near end speech samples without echo. Thus, after the call to \( \text{EC_process()} \) 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**
\[
\text{XDAS_Void EC_process(EC_Handle handle, Int } \ast \text{fe,Int } \ast \text{ne,Int size)}
\]

**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 \( \text{EC_changeState()} \) is used for changing of current Echo Canceller state.

**Function Prototype**
\[
\text{XDAS_Void EC_changeState(}
\text{EC_Handle handle,}
\text{Int state,}
\text{IEC_SetState setResetFlag)}
\]

**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 \( \text{state} \)

Table 3-8 determines enumerator \( \text{IEC_SetState} \).
Table 3-8. IEC_SetState Enumerator

typedef enum IEC_SetState {
    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_SetState;

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 SourceFlexExamples\StandaloneXDAS\G.168.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Description of Directory Tree</td>
<td>A-2</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Test file</td>
</tr>
<tr>
<td>FileC5x.c</td>
<td>File input/output functions</td>
</tr>
<tr>
<td>..\ROM\CSTRom.s54</td>
<td>ROM entry address</td>
</tr>
<tr>
<td>Test.cmd</td>
<td>Linker command file</td>
</tr>
<tr>
<td>Vectors\output.pcm</td>
<td>Reference output test vectors</td>
</tr>
</tbody>
</table>

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 with 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

A
ALG, interface 1-3
ALG_activate 1-3
ALG_control 1-4
ALG_create 1-3
ALG_deactivate 1-4
ALG_delete 1-3
ALG_exit 1-4
ALG_init 1-4
Algorithm Deletion 3-6
Algorithm Initialization 3-6
Application Development steps to creating an application 1-7
Application/Framework 1-3

D
Directory Tree A-2
Distortions. See Input Signals Requirements

E
EC_apply() 3-5
EC_changeState 3-8
EC_control() 3-5
EC_create 3-5
EC_delete 3-6
EC_exit 3-6
EC_getStatus 3-7
EC_init 3-6
EC_process 3-8
Echo Canceller, limitations 1-8
Environment, for testing A-2

F
Framework 1-3
Functions standard 3-5 vendor-specific 3-7

H
Header file for abstract interfaces 1-5 for concrete interfaces 1-4

I
IALG 1-5
IEC_PARAMS 3-3
IEC_Params 3-2
IEC_SetState Enumerator 3-9
IEC_Status 3-4
IEC_Tail 3-3
Input, signal requirements 2-3
Input Signals Requirements 3-5
Instance Creation 3-5
Instance Deletion 3-6
Integration example of 2-5 overview 2-2 steps to integrating a EC generator into a framework 2-4
Interface 1-4 abstract 1-5 concrete 1-4 vendor implementation 1-5

L
Limitations, Echo Canceller 1-8
Index

M
Module Instance Lifetime. See Application Development

P
parameters
Bit Definition in ecState Word 3-3
Echo Canceller Run-Time creation 3-2
IEC_Tail Enumerator Definition 3-3
Process Samples 3-8

S
Signals, requirements 2-3
Source file
for abstract interfaces 1-5
for concrete interfaces 1-4
State Changing 3-8
Status Reporting 3-7

T
Test files A-2
format A-2
project A-3
Test Environment A-2

X
XDAIS
Application Development 1-5
Application/Framework basics 1-3
Interface 1-4
related documentation 1-2
System Layers, illustration of 1-3