

TMS320 DSP Algorithm Standard Demonstration Application

Literature Number: SPRU361E
September 2002



Printed on Recycled Paper

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 that 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

Read This First

About This Manual

This document is a companion to the *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) and contains all of the APIs that are defined by the *TMS320 DSP Algorithm Interoperability Standard* (also known as XDAIS) specification.

The TMS320 DSP algorithm standard is part of TI's eXpressDSP technology initiative. Algorithms that comply with the standard are tested and awarded an "expressDSP-compliant" mark upon successful completion of the test.

Intended Audience

This document assumes that you are fluent in the C language, have a good working knowledge of digital signal processing and the requirements of DSP applications, and have had some exposure to the principles and practices of object-oriented programming. This document describes the interfaces between algorithms and the applications that utilize these algorithms. System integrators will see how to incorporate multiple algorithms from separate developers into a complete system. Algorithm writers will be able to determine the methods that must be implemented by eXpressDSP-compliant algorithms and how each method works in a system.

How to Use This Manual

This document contains the following chapters:

- ❑ **Chapter 1 – Using the Demonstration Application**, illustrates the benefits of standardizing components according to the TMS320 DSP Algorithm Standard Rules and Guidelines.
- ❑ **Chapter 2 – Description of the Target Application**, describes the target application that runs under BIOS.
- ❑ **Chapter 3 - TMS320 DSP Algorithm Components and Performance Characterization**, explains how to link TMS320 DSP algorithm components and provides performance characterization for each algorithm.
- ❑ **Appendix A – TMS320 DSP Demo Software Descriptions**, provides demo software descriptions and APIs.

Each function reference page includes the name of the function, number and type of all parameters and return values of the function, a brief description of the function, and all preconditions and postconditions associated with the function. Preconditions are conditions that must be satisfied prior to calling the function. Postconditions are all conditions that the function insures are true when the function returns.

Preconditions must be satisfied by the client while postconditions are ensured by the implementation. Application or framework developers must satisfy the preconditions, whereas developers who implement the interfaces must satisfy the postconditions.

Additional Documents and Resources

The following documents contain supplementary information necessary to adhere to the TMS320 DSP Algorithm Standard specification:

- 1) *TMS320 DSP Algorithm Standard Rules and Guidelines (literature number SPRU352)*
- 2) *TMS320 DSP Algorithm API Reference (SPRU360)*
- 3) *TMS320 DSP Algorithm Standard Developer's Guide (SPRU424)*
- 4) *DSP/BIOS User's Guide (SPRU423)*

In addition to the previously listed documents, complete sources to modules and examples described in this document are included in *the TMS320 DSP Developer's Kit*. This developer's kit includes additional examples and tools to assist in both the development of TMS320 DSP Algorithm Standard algorithms and the integration of these algorithms into applications.

Text Conventions

The following conventions are used in this specification:

- Text inside back-quotes ("") represents pseudo-code
- Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced font`.

Contents

1	Using the Demonstration Application	1-1
	<i>Illustrates the benefits of standardizing components according to the TMS320 DSP Algorithm Standard Rules and Guidelines.</i>	
1.1	About the Demo Application	1-2
1.2	Requirements for the Demo	1-3
1.2.1	Software	1-3
1.2.2	Hardware	1-3
1.3	Opening the Demo Application	1-4
1.4	Running the Demo Application	1-6
1.5	Linking with the Alternate Algorithms	1-11
1.6	Linking with Other Algorithms	1-13
1.7	More About the Demo Application	1-14
2	Description of the Target Application	2-1
	<i>Describes the target application that runs under BIOS.</i>	
2.1	Threads and Pipes in the Demo Application	2-2
2.2	Analysis in the Demo Application	2-4
2.3	Host Plug-In in the Demo Application	2-5
3	TMS320 DSP Algorithm Components and Performance Characterization	3-1
	<i>Explains how to link TMS320 DSP algorithm components and provides performance characterization for each algorithm.</i>	
3.1	Using TMS320 DSP Standard Algorithms	3-2
3.1.1	Linking TMS320 DSP Standard Algorithm Components	3-2
3.1.2	XDAIS Demonstration Algorithm Performance Characterization	3-3
3.2	ITU g723dec_itu.l62	3-4
3.3	Texas Instruments g723dec_ti.l62	3-6
3.4	ITU g723enc_itu.l62	3-8
3.5	Texas Instruments g723enc_ti.l62	3-10
3.6	Texas Instruments lec_ti.l62	3-12
3.7	Texas Instruments g726dec_ti.l54f	3-14
3.8	Texas Instruments g726enc_ti.l54f	3-16
3.9	PUB g726enc_pub.l54f	3-18
3.10	PUB g726dec_pub.l54f	3-20
3.11	ADT lec_adt.l54f	3-22
A	TMS320 DSP Demo Software Descriptions	A-1
	<i>Provides demo software descriptions and APIs.</i>	

Figures

A-1	CPTD Module Input Sample Format	A-5
A-2	Example of an Output Where a Dial Tone was Detected	A-5
A-3	DTMF Module Input Sample Format	A-12
A-4	Example of a DTMF Output Event Stream	A-12
A-5	G711DEC Module Input Sample Format	A-18
A-6	G711ENC Module Input Sample Format	A-24
A-7	G726DEC Input Sample Format	A-43
A-8	G726ENC Module Output Sample Format	A-49
A-9	Line Echo Canceller	A-80

Tables

2-1	Processing Threads	2-3
2-2	Pipe Dimensions and Associated Data Notification	2-3
2-3	STS Objects Required by Target Application	2-4
3-1	ITU g723dec_itu.l62 Algorithm	3-4
3-2	Texas Instruments g723dec_ti.l62 Algorithm	3-6
3-3	ITU g723enc_itu.l62 Algorithm	3-8
3-4	Texas Instruments g723enc_ti.l62 Algorithm	3-10
3-5	Texas Instruments lec_ti.l62 Algorithm	3-12
3-6	Texas Instruments g726dec_ti.l54f Algorithm	3-14
3-7	Texas Instruments g726enc_ti.l54f Algorithm	3-16
3-8	PUB g726enc_pub.l54f Algorithm	3-18
3-9	PUB g726dec_pub.l54f Algorithm	3-20
3-10	ADT lec_adt.l54f Algorithm	3-22

Using the Demonstration Application

The TMS320 DSP Algorithm Standard demo application illustrates the benefits of standardizing components according to the *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352). This standard, proposed by Texas Instruments, allows DSP algorithms that originate from different sources to share a common interface.

This document is divided into three sections. Chapter one explains how to run and use the demonstration application. Chapter two discusses the general operation of the aggregate system including pipes, threads, I/O, and notification functions. Chapter three discusses standard interface topics.

Topic	Page
1.1 About the Demo Application	1-2
1.2 Requirements for the Demo	1-3
1.3 Opening the Demo Application	1-4
1.4 Running the Demo Application	1-6
1.5 Linking with the Alternate Algorithms	1-11
1.6 Linking with Other Applications	1-13
1.7 More About the Demo Application	1-14

1.1 About the Demo Application

The demo application includes a target DSP application that uses eXpressDSP-compliant algorithms and a Code Composer Studio™ plug-in that provides a host user interface to the target application.

This application shows how eXpressDSP-compliant algorithms can be readily interchanged, how DSP applications can take advantage of DSP/BIOS for threading control, instrumentation and real-time analysis; how Code Composer Studio can be extended to provide application-specific functionality; and, how RTDX can be used for host/target communication.

The demo application takes input audio from a source such as a CD player and generates output in the form of audio. The Code Composer Studio plug-in allows the user to apply various algorithms to the audio stream. The demo application can be relinked with eXpressDSP-compliant variations of the algorithms.

1.2 Requirements for the Demo

This section describes how to configure your computer with the correct software and hardware in order to use the TMS320 DSP Algorithm Standard demo application.

1.2.1 Software

In order to use the demo application, you must have installed the following software on your PC:

- Code Composer Studio (this includes the TMS320 DSP Algorithm Standard Developer's Kit)

1.2.2 Hardware

In order to use the demo application, you must have the following hardware connected to your computer:

- Any of the following DSP development boards:
 - TI C6201 EVM
 - TI C6711 DSK (or C6211 DSK)
 - TI C5402 DSK
- An audio source connected to the LINE IN port of the EVM or the MIC IN port on the DSK. Use a 1/8" miniplug to 1/8" miniplug audio cable (Radio Shack Cat No. 42-2387).
- An amplified speaker connected to the LINE OUT port of the EVM or DSK.

1.3 Opening the Demo Application

To open the demonstration application, follow these steps:

Note: C6000 or C5000 Target Configuration

The following instructions use the word `target` in the generic sense. To use the demonstration application with the C6000, substitute C6000 wherever `target` appears. To use the demonstration application with the C5000, substitute C5000 wherever `target` appears.

- 1) From the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 2 (*target*)→Hardware Resets→*your board*. You should see a message in an MS-DOS window that the reset was successful.
- 2) From the Windows Start menu, choose Programs→Texas Instruments→Code Composer Studio 2 (*target*)→Code Composer Studio. (Or, double-click the Code Composer Studio icon on your desktop.)
- 3) Choose File→Load Program. Load the demo.out file which is found in `c:\ti\examples\<targetboard>\xdais\demo\src` in the folder.
- 4) Choose File→Workspace→Load Workspace. The demo includes workspace definition files. Load the workspace file for your screen resolution. For example, if your screen resolution is 1024 x 768, load the `1024x768_target.wks` file.

Loading the workspace definition files resizes your Code Composer Studio window and opens the demo plug-in and a number of DSP/BIOS tools.

If you are unable to open the workspace due to configuration conflict, first check the name of the processor using the Code Composer Studio Setup application. If the name is not CPU_1, rename it CPU_1. Then save your settings and restart Code Composer Studio. For instructions, refer to the Code Composer Studio Online Help. If you are still unable to open the workspace, you may manually create a new workspace by performing the following steps.

- Open the project.
Choose Project→Open. Open the demo.pjt project. If you installed Code Composer Studio in `c:\ti`, this project file is in `c:\ti\examples\<targetboard>\xdais\demo\src`.
- Load the target.
Choose File→Load Program. Load the demo.out file, which is found in `c:\ti\examples\<targetboard>\xdais\demo\src`.

- Open DSP/BIOS real-time analysis tools:
Choose DSP/BIOS→RTA control panel.
(If some boxes are not checked, right-click and choose Enable All.)
Choose DSP/BIOS→CPU Load Graph.
Choose DSP/BIOS→Execution Graph.
Choose DSP/BIOS→Statistics View.
- Open the demo plug-in:
Choose Tools→XDAIS→Demo.


Save this workspace with a name of your choosing and in a location you will remember, as you will want to reuse it. Do this by choosing File→Work-space→SaveWorkspace.


- 5) Choose Project→Open. Open the demo.pjt project. If you installed Code Composer Studio in c:\ti, this project file is in c:\ti\examples*<targetboard>*\xdais\demo\src.

1.4 Running the Demo Application


To run the demonstration application, follow these steps:

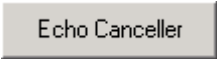
1) Start the audio source. For example if you are using your PC's CD player as an audio source, insert an audio CD and use the CD player application to play the disc. (If your CD finishes playing while you are using this application, restart the CD).

2) Click the  Launch button on the demo plug-in's control panel. This loads the program onto the target.

3) Click the  Loopback button. This button runs the program, which takes input from the audio source and generates output in the form of audio. The vocoder and echo cancel button, which have been grayed out until now, are enabled. At this point, no signal processing algorithms are applied to the signal.

4) You can now experiment with the controls in the following ways:

■ Click the  Vocoder button to toggle the vocoder encoding and decoding on and off.

■ Click the  Echo Cancel button to toggle the echo cancellation on and off.

■ Right-click the Vocoder button and select Options from the pop-up menu. Set options for the algorithm and click OK. The run-time configurable options for the G.723 algorithm used in the C6000 version of the demo are as follows:



- Encoder bit rate. May be 5300 bps or 6300 bps.
- Enable/Disable High Pass Filter. If checked, the input to the encoder is high-pass filtered.
- Enable/Disable VAD/CNG. If checked, voice activity detection (VAD) is enabled during encoding.
- Enable/Disable Postfilter. If checked, the decoder's output is post-filtered.

The run-time configurable option for the G.726 algorithm used in the C5000 version of the demo is:

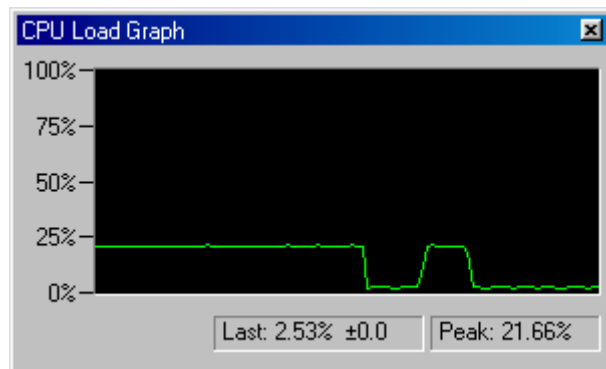
- Frame Length. The same length is used by both the decoder and encoder.

- Right-click the Echo Cancel button and select Options from the pop-up menu. Set options for the algorithm and click OK. The only option available is as follows:
 - Enable/Disable Non-Linear Process. If checked, the non-linear processor is enabled.

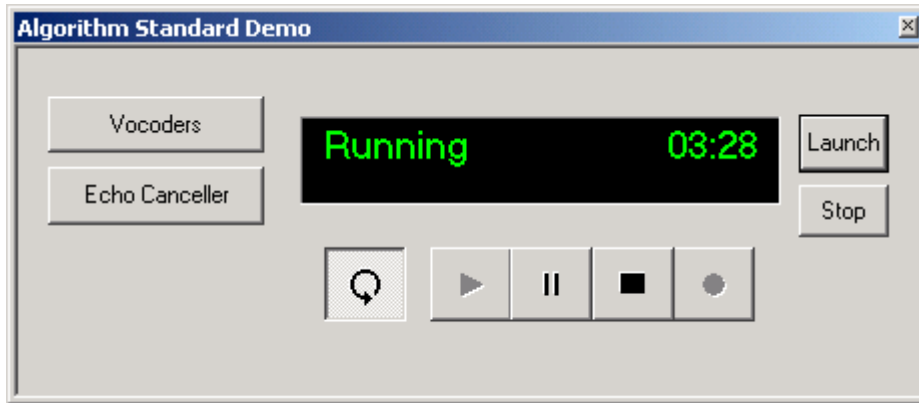
Note: Record and Play

The  Record and  Play buttons are not active in this version of the demo.

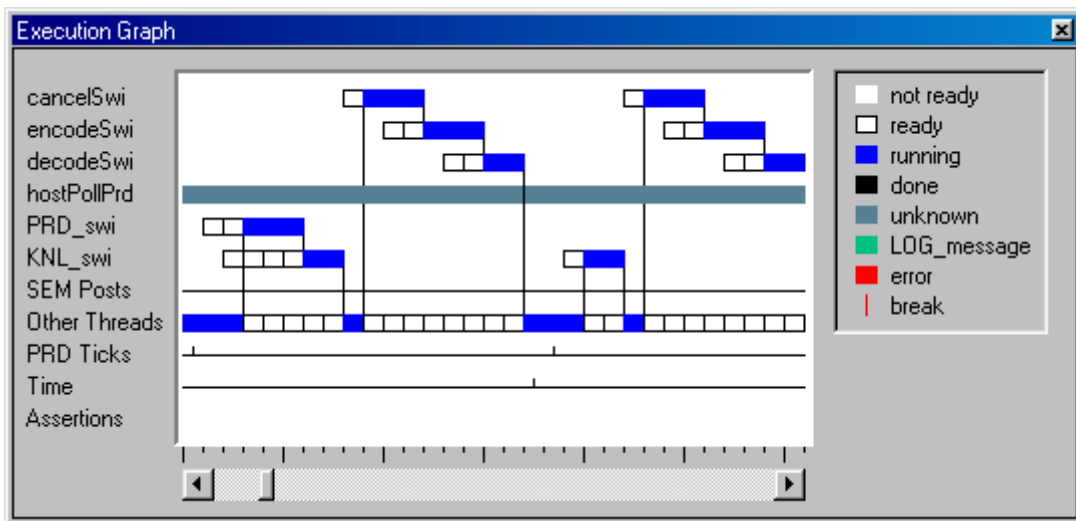
- 5) As you experiment, notice the changes to the following:
- **Audio output quality.** The audio quality varies depending on whether the vocoder algorithm is in use and the option settings for this algorithm. Differences are most apparent with music that has a wide dynamic range since vocoders are intended for speech processing. Since there is no echo to cancel when the input comes from a CD player, you should hear no differences when toggling the echo-cancel algorithm on and off.
 - **CPU Load Graph.** Notice the peak load value. Right-click on the CPU Load Graph and choose Clear Peak from the pop-up menu to see the peak value with the current settings.



- **Status display.** This area shows the vendor and name of the algorithms in use. The plug-in extracts this information from the algorithm. It can do this because the algorithms follow the naming conventions specified by the standard.



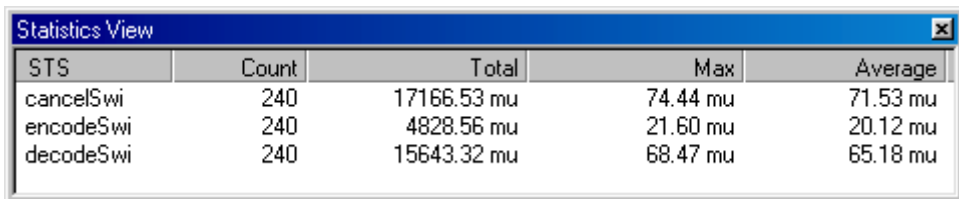
- **Execution Graph.** Right-click on the Execution Graph and choose Refresh Window from the pop-up menu to update the graph.



- **Thread statistics.** These numbers show the time spent during various software interrupt functions. For example, encodeSWI is the software interrupt (SWI) that does the audio encoding. DSP/BIOS SWI manager automatically gathers real-time statistical data for each SWI

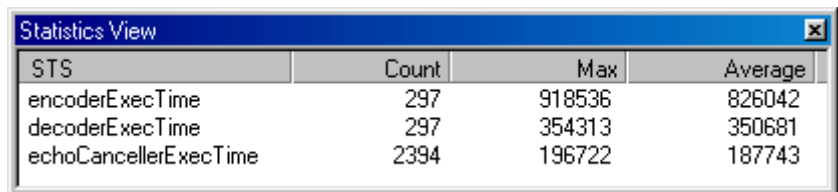
thread, such as the number of times the thread has run, the average and maximum time elapsed from when the thread gets posted to when it completes. These statistics are then streamed to the host and displayed in the Statistics View.

Right-click on the Statistics View and choose Clear from the pop-up menu to reset the values to 0. This action allows you to see the maximum and average values for your current algorithm settings. (You see that some instructions are performed even if you have toggled off the algorithms. These SWI functions also copy the signal from the source to the output, so some instruction cycles are used even if the signal is not being encoded and decoded.)



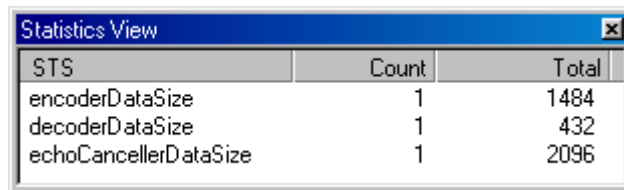
STS	Count	Total	Max	Average
cancelSwi	240	17166.53 mu	74.44 mu	71.53 mu
encodeSwi	240	4828.56 mu	21.60 mu	20.12 mu
decodeSwi	240	15643.32 mu	68.47 mu	65.18 mu

- Time to completion statistics.** In addition to the automatically gathered thread execution statistics, the demo application collects and streams application-defined statistics for reporting the time to completion information for the encoding, decoding, and echo-cancellation algorithms. For example, the `encoderExecTime` shows average and maximum number of cycles that the encoder spent while processing each 30-millisecond frame and the number of times the encoder was called. Please note that these algorithms are detuned for demo purposes.



STS	Count	Max	Average
encoderExecTime	297	918536	826042
decoderExecTime	297	354313	350681
echoCancellerExecTime	2394	196722	187743

- **Algorithm Heap Data sizes.** Notice the data sizes used by the algorithm instances do not change during program execution. The DSP algorithm standard requires each algorithm component to implement the standard IALG interface. The demo application uses the standard IALG interface during instance creation and to obtain each algorithm's instance memory requirements. The demo collects this as a statistic object as shown in the following Statistics View window. Note that, there are additional data and code memory requirements for each algorithm used in the demo. This information is available as part of each algorithm's performance characterization data sheet. You may also experiment with command line tools such as sectti to inspect and validate code sizes for each library code section.



STS	Count	Total
encoderDataSize	1	1484
decoderDataSize	1	432
echoCancellerDataSize	1	2096

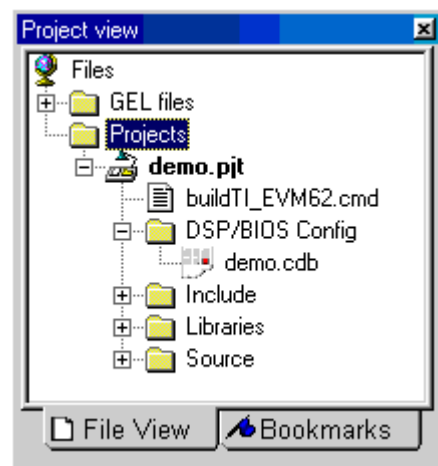
1.5 Linking with the Alternate Algorithms

This application can be relinked with different eXpressDSP-compliant versions of the DSP algorithms. Interchangeable algorithms are one of the major benefits of using standard-compliant algorithms.

For C6000 platforms, an ITU G.723 algorithm is provided as an alternative in the demo application. For C54x platforms, a PUB G.726 algorithm is provided as an alternative.

To link with other algorithms, follow these steps:

- 1) Click the Stop button to halt the target processor.
- 2) In the Project View, expand the Project folder and the demo.pjt project so that you can see the buildTI_target.cmd file.



- 3) Choose Project→Add Files to Project.
- 4) Change the Files of Type to Linker Command File (*.cmd).
- 5) Select the buildITU_target.cmd (C6000) or buildPUB_target.cmd (C54x) file and click Open. This linker command file links the application with the alternate version of the vocoder algorithm instead of the Texas Instruments version used in the previous section. When prompted to replace the file, select Yes.
- 6) Choose Project→Build. Code Composer Studio relinks the application.
- 7) After the link is complete, click the Launch button in the Demo window.

- 8) Experiment with the controls and watch changes in the various windows as you did in the previous section. Notice differences between the algorithms in the following areas:
- **Audio output quality.** You may be able to hear differences in audio quality between the two algorithms.
 - **Status display.** When the vocoder is enabled, this area shows the algorithm vendor.
 - **CPU Load Graph.** The load may be higher with the alternate algorithms.
 - **Execution Graph.** Assertions are more common with the alternate algorithms. An assertion usually indicates that a real-time deadline was missed.
 - **Maximum and Average statistics.** In general, more instruction cycles may be consumed with the alternate algorithms.
 - **Time to completion statistics.** In general, more time is consumed by the alternate algorithms.
 - **Instance Heap Data sizes.** The code and data sizes are different with the two sets of algorithms.

1.6 Linking with Other Algorithms

The vocoder and echo-canceller algorithms used in this demo are compliant with the TMS320 DSP Algorithm Standard (XDAIS). This demo application uses only the standard algorithm interface and module Application Program Interfaces (APIs) described in Appendix A.

This standardization makes it possible to use other eXpressDSP-compliant algorithms solely by editing the linker command file and relinking the project. There is no need to change or recompile the application code.

To link the demo application with a different vendor's component, for example, the G.723 Encoder component from Texas Instruments, edit the linker command file as shown in the following excerpts. These excerpts are taken from the buildITU_target.cmd linker command file.

- ❑ This line sets the generic module interface, IG723ENC, to the vendor-specific module interface.

```
Change this: _G723ENC_IG723ENC= _G723ENC_ITU_IG723ENC;
To this:     _G723ENC_IG723ENC= _G723ENC_TI_IG723ENC;
```

- ❑ This excerpt includes the vendor specific library or object file that implements the algorithm.

```
Change this: .vocoder_code:
              {
                ...
                ..\extern\lib\g723_itu.l62 (.text)
                ...} > SBSRAM PAGE 0
```

```
To this:     .vocoder_code:
              {
                ...
                ..\extern\lib\g723_ti.l62 (.text)
                ...} > SBSRAM PAGE 0
```

If you are using a C54x platform, compare the buildPUB_target.cmd and buildTI_target.cmd files to see how similar changes can be made to these files.

Please refer to the application note, *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (literature number SPRA577), for details on how to use eXpressDSP-compliant algorithms.

1.7 More About the Demo Application

The TMS320 DSP Algorithm Standard demonstration application shows how easy it is to integrate standard compliant algorithms from multiple vendors and to switch algorithm components in and out of the application without recompiling code. The demo brings together all four integrated components of the eXpressDSP Real-Time Software Technology:

- Code Composer Studio IDE
- DSP/BIOS real-time software foundation
- The TMS320 DSP Algorithm Standard
- TI Third Party Software Developers Network

The target application processes an incoming audio stream in real-time by performing encode and decode functions, and G165 compliant line-echo cancellation to produce an output audio stream. The encode and decode functions are eXpressDSP-compliant (G723.1 for the C6000 version and G726 for the C5000 version).

The same application source supports both the C5000 and C6000 target development boards. Build-time configuration for either target is accomplished by simply compiling and linking the source code with board-specific codec drivers and DSP/BIOS system and algorithm libraries. Code Composer Studio provides the graphical interface for configuring all aspects of a DSP/BIOS application. The drivers support real-time audio, while processing threads implement the algorithmic and control functionality of the target application.

Description of the Target Application

The TMS320 DSP Algorithm Standard demonstration application shows how easy it is to integrate standard compliant algorithms from multiple vendors and to switch algorithm components in and out of the application without recompiling code. The demo brings together all four integrated components of the TMS320 DSP Algorithm Standard Real-Time Software Technology:

- Texas Instruments DSK or EVM card
- Code Composer Studio IDE
- DSP/BIOS real-time software foundation
- the TMS320 DSP Algorithm Standard
- third-party algorithms which support the C6000 and C5000 DSP targets within the application framework

The target application processes an incoming audio stream in real-time by performing encode and decode functions, and G165-compliant line-echo cancellation to produce an output audio stream. The encode and decode functions are standard-compliant (G723.1 for the C6000 version and G726 for the C5000 version).

The same application source supports both the C5000 and C6000 target development boards. Build-time configuration for either target is accomplished by simply compiling and linking the source code with board-specific codec drivers and DSP/BIOS system and algorithm libraries. Code Composer Studio provides the graphical interface for configuring all aspects of a DSP/BIOS application. The drivers support real-time audio, while processing threads implement the algorithmic and control functionality of the target application.

Topic	Page
2.1 Threads and Pipes in the Demo Application	2-2
2.2 Analysis in the Demo Application	2-4
2.3 Host Plug-In in the Demo Application	2-5

2.1 Threads and Pipes in the Demo Application

DSP/BIOS SWI execution threads, PIP pipes and real-time analysis objects form the framework of the demo application. Fixed-size buffered frames of streamed audio data flow through the functions of the application. The function blocks are independent software threads, or subsystems. The data paths that connect these subsystems and tie the target application to the “outside world” are implemented with DSP/BIOS data pipes.

Each processing block or function corresponds to a DSP/BIOS SWI execution thread. An execution thread gets scheduled to run when all of its input pipes have a new frame of data available and all of its output pipes are writable (i.e., their readers have consumed the earlier data). A thread runs to completion once scheduled, and consumes a frame of data from each one of its input pipes, producing new data frames on its output pipes. A control, or process, function uses its algorithm component to produce the output data frame.

DSP/BIOS provides data notification capabilities to synchronize the transfer of data. Whenever a writer puts a frame of data on a pipe, the pipe’s notify-reader function is implicitly called to inform that pipe’s reader that a full frame of data is ready to read. All pipes in the demo use data notification to synchronize their processing components by clearing a bit in the reader software interrupt (SWI) thread’s mailbox. When all required bits in a thread’s mailbox are cleared, the DSP/BIOS run-time schedules the thread for execution. This execution occurs when all currently running equal, or higher priority, threads complete their executions. Likewise, when a reader thread gets a frame out of its input pipe, a notifyWriter function notifies the pipe’s writer mailbox that the pipe is available.

Board-specific codec drivers provide the synchronization heart-beat for the flow of data throughout the demo application processing modules. External hardware interrupts drive the codec’s buffered serial port. The demo application’s initialization sequence primes the farEnd pipe by putting an initial empty frame, allowing the canceller thread to start operating in synchrony with the arrival of new frames. In a cascading fashion, all pipes and processing blocks are chained back to back to operate in this frame-synchronous operation mode.

Table 2-1 lists the processing threads.

Table 2-1. Processing Threads

Thread	Description	Function
cancelSwi	Runs G.165 (LEC) on a frame of PCM data. Processes nearEnd input from Line In and echo from farEnd.	cancelFxn
encodeSwi	Runs Encoder on a frame of PCM data.	encoderFxn
decodeSwi	Runs Decoder on a frame of compressed bits.	decoderFxn
HostPollPrd	Periodically checks plug-in updated shared data structure and updates target algorithm settings	HostPollFxn

Table 2-2 lists the pipes, their dimensions, and associated data notification functions.

Table 2-2. Pipe Dimensions and Associated Data Notification

Pipe	Framesize	# Frames	Notify Reader	Notify Writer
lineIn	240	2	cancelSwi	DSS_rxPrime
lineOut	240	2	DSS_txPrime	decodeSwi
encoderIn	240	1	encodeSwi	cancelSwi
decoderIn	240	1	decodeSwi	encodeSwi
farEnd	240	1	cancelSwi	decodeSwi

You may view additional properties of these and other DSP/BIOS objects by opening the demo.cdb file using the Code Composer Studio.

2.2 Analysis in the Demo Application

Several DSP/BIOS statistics (STS) objects are defined at build-time. STS objects allow DSP/BIOS Statistics to monitor basic statistics, reporting real-time statistical information to the host application.

The target application makes use of six user-defined STS objects. Table 2-3 lists these objects.

Table 2-3. STS Objects Required by Target Application

STS Objects	Description
encoderExecTime	Encoder time-to-completion cycle count (to process a 30ms frame)
decoderExecTime	Decoder time-to-completion cycle count (to process a 30ms frame)
echoCancellerExecTime	EchoCanceller time-to-completion cycle count (for a 30ms frame)
encoderDataSize	Instance data size for each encoder algorithm instance object.
decoderDataSize	Instance data size for each decoder algorithm instance object.
echoCancellerDataSize	Instance data size for each EchoCanceller algorithm instance object.

Additionally, a LOG object, trace, is used to communicate application debugging and status information to the host application.

2.3 Host Plug-In in the Demo Application

An independently scheduled periodic thread managed by the DSP/BIOS PRD module `hostPollPrd`, periodically runs and updates the global application state. The plug-in asynchronously signals the current user interface state to the target application. The plug-in uses standard Code Composer Studio APIs to encode and write current algorithm settings to a shared data-structure on the target application. The settings can reflect whether an algorithm is enabled or disabled, as well as each enabled algorithm's run-time configurable status parameters.

The `hostPollPrd` thread updates the application's global state by decoding the state information in the designated section of the target DSP memory. For example, when the user enables the Echo Cancel button on the plug-in window, the `hostPollPrd` thread decodes this state change information and sets the global `cancelEnable` flag. The frame-synchronous `cancelSwi` thread always checks the flag to see if it needs to perform the echo-cancellation operation using its `lineIn` and `farEnd` input frames. If the flag is not set, the `cancelSwi` thread instead copies its `lineIn` frames to its output pipe, ensuring a continuous flow of data to the next processing thread's input pipe. Real-time control operations are handled in a similar fashion.

TMS320 DSP Algorithm Components and Performance Characterization

This chapter explains how to link TMS320 DSP algorithm components and provides performance characterization for each algorithm.

Topic	Page
3.1 Using TMS320 DSP Standard Algorithms	3-2
3.2 ITU g723dec_itu.l62	3-4
3.3 Texas Instruments g723dec_ti.l62	3-6
3.4 ITU g723enc_itu.l62	3-8
3.5 Texas Instruments g723enc_ti.l62	3-10
3.6 Texas Instruments lec_ti.l62	3-12
3.7 Texas Instruments g726dec_ti.l54f	3-14
3.8 Texas Instruments g726enc_ti.l54f	3-16
3.9 PUB g726enc_pub.l54f	3-18
3.10 PUB g726dec_pub.l54f	3-21
3.11 ADT lec_adt.l54f	3-24

3.1 Using TMS320 DSP Standard Algorithms

Please refer to the application note, *Using the eXpressDSP Algorithm Standard in a Static DSP System* (literature number SPRA577), for details on how to use eXpressDSP-compliant-algorithms.

3.1.1 Linking TMS320 DSP Standard Algorithm Components

The vocoder and echo-canceller algorithms used in this demo are compliant with the TMS320 DSP Algorithm Standard (also known as XDAIS). This demo application uses only the standard algorithm interface and module Application Program Interfaces (APIs) published in Appendix A of this document. This standardization makes it possible to use or interchange other compliant algorithm components solely through the use of linker technology. There is no need to change or recompile the application code.

The following excerpts are taken from the linker command file, buildITU.cmd:

- 1) This line sets the generic module interface, IG723ENC, to the vendor specific module interface.

```
G723ENC_IG723ENC= G723ENC_ITU_IG723ENC;
```

- 2) This excerpt includes the vendor specific library or object file that implements the algorithm.

```
.vocoder_code:  
{  
    ...  
    ..\extern\lib\g723_itu.l62 (.text)  
    ...} > SBSRAM PAGE 0
```

To link the demo application with a different vendor's component, for example, the G723 Encoder component from Texas Instruments, replace the linker command file excerpts above with the following lines:

```
_G723ENC_IG723ENC= _G723ENC_TI_IG723ENC;
```

and

```
..\extern\lib\g723_ti.162 (.text)
```

3.1.2 XDAIS Demonstration Algorithm Performance Characterization

Sections 3.2 through 3.11 provide algorithm performance characterization.

3.2 ITU g723dec_itu.l62

Table 3-1. ITU g723dec_itu.l62 Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G723DEC	ITU	62	none	none	03.29. 00	g723_itu.l62

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	432	0	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size	Align (bytes)
g723_ituobj(.far)	19,168	0	R/W	No	g723_ituobj(.text)	78,912	0
					g723_ituobj(.cinit)	19,180	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	0	.text	78,208
.far	19,168	.cinit	19,180

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	3600	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
decode()	430,000

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
decode()	30,000	430,000	no periodic execution	no periodic execution

(g) ROMable

Yes	No
x	

Note:

Module memory requirements for the ITU encoder include the module memory of the ITU decoder since the same library implements both module interfaces.

3.3 Texas Instruments g723dec_ti.l62

Table 3-2. Texas Instruments g723dec_ti.l62 Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G723DEC	TI	62	none	none	03.29.00	g723_ti.l62

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	432	0	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g723_tiobj(.far)	19,174	0	R/W	No	g723_tiobj(.text)	80,160	0
					g723_tiobj(.cinit)	19,180	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	44	.text	80,160
.far	19,174	.cinit	19,180

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	3424	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
decode()	420,000

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
decode()	30,000	420,000	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

Note:

Module memory requirements for the TI encoder include the module memory of the TI decoder since the same library implements both module interfaces.

3.4 ITU g723enc_itu.l62

Table 3-3. ITU g723enc_itu.l62 Algorithm

Module Name	Architecture	Variant	Version	Date	Library Name
G723ENC_ITU	62	none	none	03.29.00	g723_itu.l62

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	1484	0	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g723_ituobj(.far)	19,168	0	R/W	No	g723_ituobj(.text)	78,208	0
					g723_ituobj(.cinit)	19,180	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	0	.text	78,208
.far	19,168	.cinit	19,180

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	3600	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
encode()	1,636,000

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
encode()	30,000	1,636,000	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

Note:

Module memory requirements for the ITU encoder include the module memory of the ITU decoder since the same library implements both module interfaces.

3.5 Texas Instruments g723enc_ti.l62

Table 3-4. Texas Instruments g723enc_ti.l62 Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G723ENC	TI	62	none	none	03.29.00	g723_ti.l62

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	1,484	0	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g723_tiobj(.far)	19,174	0	R/W	No	g723_tiobj(.text)	80,160	0
g723_tiobj(.bss)	44	0	R/W	No	g723_tiobj(.cinit)	19,180	

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	44	.text	80,160
.far	19,174	.cinit	19,180

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	3,424	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
encode()	428,000

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
encode()	30,000	428,000	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

Note:

Module memory requirements for the TI encoder include the module memory of the TI decoder since the same library implements both module interfaces.

3.6 Texas Instruments lec_ti.l62

Table 3-5. Texas Instruments lec_ti.l62 Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
LEC	TI	62	none	none	04.05.00	lec_ti62.l62

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	2096	512	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
lec_ti.obj(.bss)	176	0	R/W	No	g165a.obj(.text)	2432	0
			R	No	lec_ti.obj(.text)	2464	0
					lec_ti.obj(.cinit)	220	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	176	.text	4896
		.cinit	220

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	256	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
echoCancel()	345,000

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
echoCancel()	30,000	345,000	No periodic execution	No periodic execution
feedData()	30,000	4,000	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

3.7 Texas Instruments g726dec_ti.I54f

Table 3-6. Texas Instruments g726dec_ti.I54f Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G726DECI	TI	54f	none	none	04.03.00	g726_ti.I54f

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	4	0	External
1	Persist	196	256	SARAM

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g726dec_ti_math.obj (.data)	1698	0	R	No	g726dec_ti.obj (.text:init)	6	0
g726dec_ti_vtab.obj (.bss)	44	0	R	No	g726dec_ti.obj (.text:exit)	6	0
					g726dec_ti_alg.obj (.text:algAlloc)	94	0
					g726dec_ti_alg.obj (.text:algFree)	60	0
					g726dec_ti_alg.obj (.text:algInit)	64	0
					g726dec_ti_alg.obj (.text:algMoved)	34	0
					g726dec_ti_ig726dec.obj (.text)	134	0
					g726dec_ti_math.obj (.text)	1710	0
					g726dec_ti_vtab.obj (.cinit)	48	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	44	.text	2108
.data	1698	.cinit	48

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	98	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
algInit ()	49

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
decode()	125	733	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

3.8 Texas Instruments g726enc_ti.I54f

Table 3-7. Texas Instruments g726enc_ti.I54f Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G726ENC	TI	54f	none	none	03.28.00	g726_ti.I54f

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	4	0	External
1	Persist	196	256	SARAM

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g726enc_ti_math.obj (.data)	849	0	R	No	g726enc_ti.obj (.text:init)	6	0
g726enc_ti_vtab.obj (.bss)	44	0	R	No	g726enc_ti.obj (.text:exit)	6	0
					g726enc_ti_ialg.obj (.text:algAlloc)	94	0
					g726enc_ti_ialg.obj (.text:algFree)	60	0
					g726enc_ti_ialg.obj (.text:algInit)	54	0
					g726enc_ti_ialg.obj (.text:algMoved)	34	0
					g726enc_ti_ig726enc. obj (.text)	134	0
					g726enc_ti_math.obj (.text)	1712	0
					g726enc_ti_vtab.obj (.cinit)	48	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	44	.text	2096
.data	1698	.cinit	48

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	98	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
algInit()	49

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
encode()	125	674	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

3.9 PUB g726enc_pub.I54f

Table 3-8. PUB g726enc_pub.I54f Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G726ENC	PUB	54f	none	none	03.28.00	g726_pub.I54f

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	76	0	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g726coder.o54f (.bss)	978	0	R/W	No	g726coder.o54f (.text)	5408	0
vtab_e.o54f (.bss)	80	0	R	No	g726enc_pub.o54f (.text)	12	0
vtabif_e.o54f (.bss)	16	0	R	No	vtabif_e.o54f (.text)	610	0
					vtabif_e.o54f (.text:algActivate)	12	0
					vtabif_e.o54f (.text:algAlloc)	102	0
					vtabif_e.o54f (.text:algControl)	26	0
					vtabif_e.o54f (.text:algDeactivate)	12	0
					vtabif_e.o54f (.text:algDelete)	90	0
					vtabif_e.o54f (.text:algMoved)	12	0
					vtabif_e.o54f (.text:algNumAlloc)	48	0
					g726coder.o54f (.cinit)	1008	0
					vtab_e.o54f (.cinit)	88	0
					vtabif_e.o54f (.cinit)	20	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	537	.text	2882
.data	0	.cinit	558

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	52	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
Encode()	0

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
encode()	125	1054	No periodic execution	No periodic execution

(g) ROMable

Yes	No
	x

3.10 PUB g726dec_pub.I54f

Table 3-9. PUB g726dec_pub.I54f Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
G726DEC	PUB	54f	none	none	03.31.00	g726_pub.I54f

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	38	0	External

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
g726coder.o54f (.bss)	978	0	R/W	No	g726coder.o54f (.text)	5408	0
vtab_d.o54f (.bss)	80	0	R	No	g726dec_pub.o54f (.text)	612	0
vtabif_d.o54f (.bss)	16	0	R	No	vtabif_d.o54f (.text)	610	0
					vtabif_d.o54f (.text:algActivate)	12	0
					vtabif_d.o54f (.text:algAlloc)	102	0
					vtabif_d.o54f (.text:algControl)	26	0
					vtabif_d.o54f (.text:algDeactivate)	12	0
					vtabif_d.o54f (.text:algDelete)	90	0
					vtabif_d.o54f (.text:algMoved)	12	0
					vtabif_d.o54f (.text:algNumAlloc)	48	0
					g726coder.o54f (.cinit)	1008	0
					vtab_d.o54f (.cinit)	88	0
					vtabif_d.o54f (.cinit)	20	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	537	.text	2882
.data	0	.cinit	558

(d) *Stack Space*

Condition	Size (bytes)	Align (bytes)
Worst Case	58	0

(e) *Interrupt Latency*

Operation	Worst Case (Instruction Cycles)
decode()	0

(f) *Execution Time*

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
decode()	125	1095	No periodic execution	No periodic execution

(g) *ROMable*

Yes	No
	x

3.11 ADT lec_adt.I54f

Table 3-10. ADT lec_adt.I54f Algorithm

Module Name	Vendor Name	Architecture	Variant	Version	Date	Library Name
LEC	ADT	54f	none	none	04.03.00	lec_adt.I54f

(a) Instance Memory

memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	550	0	External
1	Persist	64	0	DARAM0
2	Persist	96	128	DARAM0
3	Persist	$(T/16) + 6$	0	SARAM
4	Persist	32	0	SARAM

Note: The unit for size and alignment is (8-bit) byte.

(b) Module Memory

Data					Program		
File(.section)	Size (bytes)	Align (bytes)	Read/Write	Scratch	File(.section)	Size (bytes)	Align (bytes)
echo_ini.obj(.const)	138	0	R	No	echo.obj(.text)	413	0
lec_adt_vtab.obj (.bss)	48	0	R	No	echo_ini.obj (.text:algInit)	257	0
lec_adt.obj(.const)	14	0	R	No	ecwrap.obj(.text)	26	0
					lec_adt.obj(.text)	254	0
					lec_adt.obj (.text:algAlloc)	264	0
					lec_adt.obj (.text:algFree)	30	0
					lec_adt.obj (.text:algInit)	182	0
					lec_adt.obj (.text:init)	6	0
					lec_adt.obj (.text:exit)	6	0
					mem.obj (.text:algAlloc)	28	0
					lec_adt_vtab.obj (.cinit)	52	0

(c) Module Memory Total

Data		Program	
Section	Size (bytes)	Section	Size (bytes)
.bss	24	.text	1065
.const	76		
.cinit	26		

(d) Stack Space

Condition	Size (bytes)	Align (bytes)
Worst Case	80	0

(e) Interrupt Latency

Operation	Worst Case (Instruction Cycles)
echoCancel()	T + 3

(f) Execution Time

Operation	Period (microsec)	Worst-case Cycles/Period	Worst-case Cycles/Period ₀	Worst-case Cycles/Period _N
echoCancel()	30,000	87409	No periodic execution	No periodic execution
feedData()	30,000	304	No periodic execution	No periodic execution

(g) ROMable

Yes	No
x	

Notes:

T refers to the number of echo canceller filter taps, i.e the tailLen.

Note the limitation of the LEC module in the demonstration code. The frameLen parameter is fixed at 240 samples, bulkDelay is set to 0 and tailLen is fixed at 32 samples.

TMS320 DSP Demo Software Descriptions

The TMS320 DSP Algorithm Standard™ (also known as XDAIS) is comprised of a set of rules and guidelines that pertain to general programming practices, instruction set architecture (ISA) specific rules, and application programmer interfaces (APIs) for abstracting system resource calls from the algorithm. This document is an adjunct to the standard.

While XDAIS does not require that an algorithm use a specific set of APIs, it is practical that the algorithm use an API that is specific to the algorithm module. Therefore, to increase acceptance of the standard and to provide examples of how such APIs could be developed, TI is distributing the APIs for various telephony algorithms in this document.

By using common algorithm APIs, you can “switch out” one of the algorithms and “link-in” another version of the same algorithm without having to recompile, (although a relink will still be necessary). This can only be done when object code compatibility is ensured, and this can only be achieved if the second algorithm also uses the same specific API. TI will distribute one or more demonstration software applications that use these algorithms and the APIs described here. In this way, TI third party vendors and customers will be able to substitute their algorithm’s object code providing they adhere to the eXpressDSP specification and the APIs described in this document.

Each interface defined in this document is presented in a common format. The interface documentation is organized in this manner:

- overall capabilities of each interface
- programming interface for each function
- description of the interface

as a series of reference pages that describes the programming interface for each function. Also included are reference pages that describe the overall capabilities of each interface and appears before the functions defined by the interface.

Each function reference page includes the function syntax, type of all parameters and return values of the function, and all preconditions (conditions that must be satisfied before calling the function) and postconditions (conditions that the function insures are true when the function returns) associated with the function.

The preconditions are conditions that must be satisfied by the client, while postconditions are those things that are guaranteed by the implementation. Thus, application or framework developers must satisfy the preconditions whereas developers that who implement the interfaces must satisfy the postconditions.

It is important to notice that the interfaces do not specify parameters such as signal frequencies, pulse width, power levels, etc. of the data. These parameters are regarded as implementation-specific and do not belong to the interface.

Please refer to the *TMS320 DSP Algorithm Standard Rules and Guidelines* (literature number SPRU352) and the *eXpressDSP Algorithm Standard (XDAIS) API Reference* (literature number SPRU360), for more information on XDAIS.

CPTD**Call Progress Tone Detection Abstract Interface****Includes**

```
#include <XDAS.h>
#include <ialg.h>
#include <icptd.h>
```

Interface**Types and Constants**

```
/*
 * ===== ICPTD_Tone =====
 * Definition of the possible tones for a CPTD instance object.
 */
typedef XDAS_Int16 ICPTD_Tone;
#define ICPTD_DIALTONE ((ICPTD_Tone)-1) /* Dial tone */
#define ICPTD_RINGINGTONE ((ICPTD_Tone)-2) /* Ringing tone */
#define ICPTD_BUSYTONe ((ICPTD_Tone)-3) /* Busy/Congestion tone */
#define ICPTD_SITTONe ((ICPTD_Tone)-4) /* Special Information tone */
#define ICPTD_WARNINGTONE ((ICPTD_Tone)-5) /* Warning tone (recording) */
#define ICPTD_PAYPHONETONE ((ICPTD_Tone)-6) /* Payphone recognition tone */
#define ICPTD_CALLWTONE ((ICPTD_Tone)-7) /* Call waiting tone */
#define ICPTD_CALLERWTONE ((ICPTD_Tone)-8) /* Caller waiting tone */
/*
 * ===== ICPTD_Obj =====
 * This structure must be the first field of all CPTD instance objects.
 */
typedef struct ICPTD_Obj {
    struct ICPTD_Fxns *fxns;
} ICPTD_Obj;
/*
 * ===== ICPTD_Cmd =====
 * Control commands for a CPTD instance object.
 */
typedef enum ICPTD_Cmd {
    ICPTD_GETSTATUS,
    ICPTD_SETSTATUS
} ICPTD_Cmd;
```

Creation Parameters

```
/*
 * ===== ICPTD_Params =====
 * This structure defines the creation parameters for a CPTD instance object.
 */
typedef struct ICPTD_Params {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Frame length in number of samples */
} ICPTD_Params;
```

Status Parameters

```
/*
 * ===== ICPTD_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct ICPTD_Status {
    Int          size;          /* Size of this structure */
    ICPTD_Tone   lastTone;     /* Last detected Tone (Read-Only) */
} ICPTD_Status;
```

Functions

```
/*
 * ===== ICPTD_Fxns =====
 * This structure defines all of the operations on CPTD objects.
 */
typedef struct ICPTD_Fxns {
    IALG_Fxns   ialg; /* ICPTD extends IALG */
    XDAS_Bool   (*control)(ICPTD_Handle handle, ICPTD_Cmd cmd, ICPTD_Status
                          *status);
    XDAS_Int8   (*detect)(ICPTD_Handle handle, XDAS_Int16 *in, ICPTD_Tone *out);
} ICPTD_Fxns;
```

Default Creation Parameters

```

/*
 * ===== ICPTD_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a CPTD object.
 */
const ICPTD_Params ICPTD_PARAMS = {
    sizeof(ICPTD_PARAMS), /* Size of this structure */
    80,                    /* 80 samples per frame, 10msec of data */
};

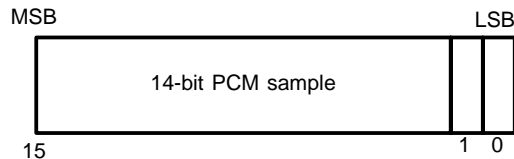
```

Description

The CPTD module is used to detect tones generated by the central office to provide the status of a phone call.

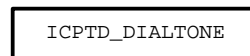
The input to the CPTD module is 14-bit-linear little endian PCM samples that are shifted towards the MSB. The output is a sequence of detected tones. See Figure A-1 for the format of the input sample.

Figure A-1. CPTD Module Input Sample Format



See Figure A-2 for an example of where a CPTD module detects a dial tone. The `detect()` function should return the number of tones in the output stream.

Figure A-2. Example of an Output Where a Dial Tone was Detected



An implementation of the CPTD module is required to support all the `ICPTD_Tone` tones listed in the interface.

Comments

Creation Parameters

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>frameLen</code>	Number of samples in the frame passed to <code>detect()</code> for processing.

Default Creation Parameters

The default creation parameters specify 80 samples for each input frame.

Status Parameters

<code>size</code>	Size of the status parameter structure. If a vendor extends the status parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>lastTone</code>	<code>if (ICPTD_Cmd == ICPTD_GETSTATUS)</code> Most recently detected tone. <code>if (ICPTD_Cmd == ICPTD_SETSTATUS)</code> This is a read-only parameter. Value is ignored.

Name	control - Runtime control and status function
Syntax	<code>val = handle->fxns->control(handle, cmd, status);</code>
Parameters	<pre> ICPTD_Handle handle; /* ICPTD object handle */ ICPTD_Cmd cmd; /* control command */ ICPTD_Status status; /* Pointer to status structure */ </pre>
Return Value	<code>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid CPTD instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>ICPTD_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>ICPTD_Cmd</code>.
Postconditions	<pre> if (cmd == ICPTD_GETSTATUS) if (val == XDAS_TRUE) </pre> <p>The status structure pointed to by <code>status</code> was successfully updated and reflects the instance current state.</p> <pre> if (val == XDAS_FALSE) </pre> <p>The update of the status structure pointed to by <code>status</code>, for some reason, failed.</p> <pre> if (cmd == ICPTD_SETSTATUS) if (val == XDAS_TRUE) </pre> <p>The write parameters in the status structure pointed to by <code>status</code> were successfully copied into the instance object.</p> <pre> if (val == XDAS_FALSE) </pre> <p>The update of the instance status write parameters, for some reason, failed.</p>
Comments	None
See Also	None

Name	detection of CPTD tones
Syntax	<pre>numTones = handle->fxns->detect(handle, in, out);</pre>
Parameters	<pre>ICPTD_Handle handle; /* ICPTD object handle */ XDAS_Int16 *in; /* Pointer to input buffer */ ICPTD_Tone *out; /* Pointer to output buffer */</pre>
Return Value	<pre>XDAS_Int8 numTones; /* returns number of tones detected */</pre>
Preconditions	<ul style="list-style-type: none"><input type="checkbox"/> <code>handle</code> is pointing to a valid CPTD instance object.<input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>.<input type="checkbox"/> <code>in</code> is pointing to a 14-bit linear PCM little endian <code>frameLen</code> number of samples as described in Figure A-1.
Postconditions	<pre>if (numTones < 0)</pre> <p>The detection of CPTD tones in the <code>in</code> buffer for some reason failed.</p> <pre>if (numTones >= 0)</pre> <p>The <code>out</code> buffer contains <code>numTones</code> number of tones.</p> <p>The contents of the <code>in</code> buffer should be the same as when entering this function.</p>
Comments	The function should work properly for all possible configurations of the <code>frameLen</code> creation parameter.
See Also	None

DTMF*Dual Tone Multi-frequency Detector Abstract Interface***Includes**

```
#include <XDAS.h>
#include <ialg.h>
#include <idtmf.h>
```

Interface**Types and Constants**

```
/*
 * ===== Events =====
 * Definition of the possible events for a DTMF instance object.
 */
#define IDTMF_EARLY          (-1)      /* Early detection of a digit */
#define IDTMF_FALSESEARLY   (-2)      /* Early detection was false */
#define IDTMF_LEADEDGE      (-3)      /* Leading edge */
#define IDTMF_TRAILEDGE     (-4)      /* Trailing edge */
#define IDTMF_DIGIT         (-5)      /* Digit detected */
/*
 * ===== Digits =====
 * Definition of the DTMF digits.
 */
#define IDTMF_0              (0)       /* Digit --0-- detected */
#define IDTMF_1              (1)       /* Digit --1-- detected */
#define IDTMF_2              (2)       /* Digit --2-- detected */
#define IDTMF_3              (3)       /* Digit --3-- detected */
#define IDTMF_4              (4)       /* Digit --4-- detected */
#define IDTMF_5              (5)       /* Digit --5-- detected */
#define IDTMF_6              (6)       /* Digit --6-- detected */
#define IDTMF_7              (7)       /* Digit --7-- detected */
#define IDTMF_8              (8)       /* Digit --8-- detected */
#define IDTMF_9              (9)       /* Digit --9-- detected */
#define IDTMF_A              (10)      /* Digit --A-- detected */
#define IDTMF_B              (11)      /* Digit --B-- detected */
#define IDTMF_C              (12)      /* Digit --C-- detected */
#define IDTMF_D              (13)      /* Digit --D-- detected */
```

```
#define IDTMF_STAR          (14)      /* Digit --*-- detected */
#define IDTMF_POND         (15)      /* Digit --#-- detected */
/*
 * ===== IDTMF_Obj =====
 * This structure must be the first field of all DTMF instance objects.
 */
typedef struct IDTMF_Obj {
    struct IDTMF_Fxns *fxns;
} IDTMF_Obj;
/*
 * ===== IDTMF_Handle =====
 * This handle is used to reference all DTMF instance objects.
 */
typedef struct IDTMF_Obj *IDTMF_Handle;
/*
 * ===== IDTMF_Cmd =====
 * Control commands for a DTMF instance object.
 */
typedef enum IDTMF_Cmd {
    IDTMF_GETSTATUS,
    IDTMF_SETSTATUS
} IDTMF_Cmd;
```

Creation Parameters

```
/*
 * ===== IDTMF_Params =====
 * This structure defines the creation parameters for a DTMF instance object.
 */
typedef struct IDTMF_Params {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Frame length in number of samples */
} IDTMF_Params;
```

Status Parameters

```

/*
 * ===== IDTMF_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IDTMF_Status {
    Int          size;          /* Size of this structure */
    XDAS_Int16   lastDigit;     /* Last detected digit (Read-Only) */
    XDAS_Int16   lastEvent;     /* Last event (Read-Only) */
} IDTMF_Status;

```

Functions

```

/*
 * ===== IDTMF_Fxns =====
 * This structure defines all of the operations on DTMF objects.
 */
typedef struct IDTMF_Fxns {
    IALG_Fxns ialg;           /* IDTMF extends IALG */
    XDAS_Bool (*control)(IDTMF_Handle handle, IDTMF_Cmd cmd, IDTMF_Status
                        *status);
    XDAS_Int8 (*detect)(IDTMF_Handle handle, XDAS_Int16 *in, XDAS_Int16 *out);
} IDTMF_Fxns;

```

Default Creation Parameters

```

/*
 * ===== IDTMF_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a DTMF object.
 */
const IDTMF_Params IDTMF_PARAMS = {
    sizeof(IDTMF_PARAMS), /* Size of this structure */
    80,                   /* 80 samples per frame, 10msec of data */
};

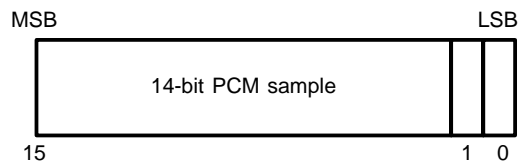
```

Description

The DTMF detection module is used in telephony applications such as financial interactions, business directories, and other menu-driven operations that respond to digits entered from a telephone.

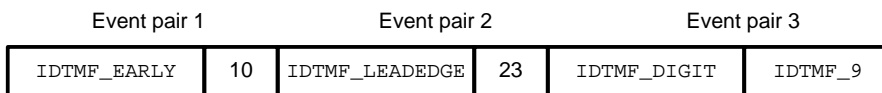
The input to the DTMF detection module is 14-bit linear little endian PCM samples shifted towards the MSB, as seen in Figure A-3. The output is a sequence of detected events. Events always occur as a pair of `XDAS_Int16` values; the first value is negative and labels the event that was detected. The second value specifies at which sample in the frame the event occurred, except the `IDTMF_DIGIT` event, which is followed by the actual digit.

Figure A-3. DTMF Module Input Sample Format



See Figure A-4 for an example of how a DTMF module detects 3 events. The last event detects digit 9. The `detect()` function should return 3 (the number of events in the output stream).

Figure A-4. Example of a DTMF Output Event Stream



An implementation of the DTMF module is required to support the `IDTMF_DIGIT` event. The other events are optional.

Comments

Creation Parameters

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>frameLen</code>	Number of samples in the frame passed to <code>detect()</code> for processing.

Default Creation Parameters

The default creation parameters specify 80 samples in each input frame.

Status Parameters

<code>size</code>	Size of the status parameter structure. If a vendor extends the status parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>lastDigit</code>	<code>if (IDTMF_Cmd == IDTMF_GETSTATUS)</code> Most recently detected digit. <code>if (IDTMF_Cmd == IDTMF_SETSTATUS)</code> This is a read-only parameter. Value is ignored.
<code>lastEvent</code>	<code>if (IDTMF_Cmd == IDTMF_GETSTATUS)</code> Most recently detected event. <code>if (IDTMF_Cmd == IDTMF_SETSTATUS)</code> This is a read-only parameter. Value is ignored.

Name**control - Runtime control and status function****Syntax**

```
val = handle->fxns->control(handle, cmd, status);
```

Parameters

```
IDTMF_Handle  handle; /* IDTMF object handle */
IDTMF_Cmd     cmd;    /* control command */
IDTMF_Status  status; /* Pointer to status structure */
```

Return Value

```
XDAS_BOOL  val; /* XDAS_TRUE if call was successful */
```

Preconditions

- `handle` is pointing to a valid DTMF instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `status` is pointing to a valid `IDTMF_Status` structure.
- `cmd` is a valid `IDTMF_Cmd`.

Postconditions

```
if (cmd == IDTMF_GETSTATUS)
```

```
if (val == XDAS_TRUE)
```

The status structure pointed to by `status` was successfully updated, reflecting the instance current state.

```
if (val == XDAS_FALSE)
```

The update of the status structure pointed to by `status`, for some reason, failed.

```
if (cmd == IDTMF_SETSTATUS)
```

```
if (val == XDAS_TRUE)
```

The write parameters in the status structure pointed to by `status` were successfully copied into the instance object.

```
if (val == XDAS_FALSE)
```

The update of the instance status write parameters, for some reason, failed.

Comments

None

See Also

None

Name	detect - Detection of DTMF digits
Syntax	<pre>numEvents = handle->fxns->detect(handle, in, out);</pre>
Parameters	<pre>IDTMF_Handle handle; /* IDTMF object handle */ XDAS_Int16 *in; /* Pointer to input buffer */ XDAS_Int16 *out; /* Pointer to output buffer */</pre>
Return Value	<pre>XDAS_Int8 numEvents; /* returns number of event detected */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid DTMF instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>in</code> is pointing to 14-bit linear little endian PCM <code>frameLen</code> number of samples as described in Figure A-3.
Postconditions	<pre>if (numEvents < 0)</pre> <p>The detection of DTMF events, for some reason, failed.</p> <pre>if (numEvents >= 0)</pre> <p>The <code>out</code> buffer contains <code>numEvents</code> number of events. The size of the <code>out</code> buffer is <code>2*sizeof(XDAS_Int16)</code>.</p> <p>The contents of the <code>in</code> buffer should be the same as when entering this function.</p>
Comments	The function should work properly for all possible configurations of <code>frameLen</code> .
See Also	None

```

Includes           #include <XDAS.h>
                    #include <ialg.h>
                    #include <ig711.h>

```

Interface***Types and Constants***

```

/*
 * ===== IG711DEC_Obj =====
 * This structure must be the first field of all G711DEC instance objects.
 */
typedef struct IG711DEC_Obj {
    struct IG711DEC_Fxns *fxns;
} IG711DEC_Obj;
/*
 * ===== IG711DEC_Handle =====
 * This handle is used to reference all G711DEC instance objects.
 */
typedef struct IG711DEC_Obj *IG711DEC_Handle;

```

Creation Parameters

```

/*
 * ===== IG711DEC_Params =====
 * This structure defines the creation parameters for all G711DEC instance.
 * objects.
 */
typedef struct IG711DEC_Params {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Number of samples in a input frame */
    IG711_Mode   mode;         /* Format of the encoded input samples */
} IG711DEC_Params;

```

Status Parameters

```

/*
 * ===== IG711DEC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG711DEC_Status {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Num samples in the in frame (Read/Write) */
    IG711_Mode   mode;        /* Format of encoded in samples (Read-Only) */
} IG711DEC_Status;

```

Functions

```

/*
 * ===== IG711DEC_Fxns =====
 * This structure defines all of the operations on G711DEC objects.
 */
typedef struct IG711DEC_Fxns {
    IALG_Fxns    ialg;        /* IG711DEC extends IALG */
    XDAS_Bool    (*control)(IG711DEC_Handle handle, IG711_Cmd cmd,
                           IG711DEC_Status *status);
    XDAS_Int16   (*decode)(IG711DEC_Handle handle, XDAS_Int8 *in, XDAS_Int16
                           *out);
} IG711DEC_Fxns;

```

Default Creation Parameters

```

/*
 * ===== IG711DEC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G711DEC object.
 */
const IG711DEC_Params IG711DEC_PARAMS = {
    sizeof(IG711DEC_PARAMS), /* Size of this structure */
    80,                      /* 80 samples per frame, 10msec of data */
    IG711_ALAW,             /* input data format is A-law */
};

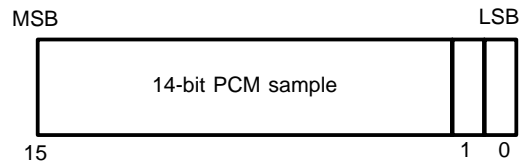
```

Description

The G711DEC module is used in digital telecommunication systems to decode ITU G.711 voice encoded samples to uniform PCM samples.

The input to the G711DEC module is 8-bit A-law or μ -law encoded samples. Each 8-bit sample is shifted towards the LSB for the C54x and consumes a 16-bit word. In other words, two input samples are not packed into a word. The output is 14-bit linear little endian PCM samples shifted towards the MSB. See Figure A-5 for the format of the output sample.

Figure A-5. G711DEC Module Input Sample Format



The input samples are decoded according to the ITU A-law or μ -law standards. A-law is the standard in North America and Japan, while μ -law is the standard in Europe.

Comments**Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>frameLen</code>	Number of samples in the frame passed to <code>detect()</code> for processing.
<code>mode</code>	IG711_ALAW – A-law decoding. IG711_ULAW - μ -law decoding.

Default Creation Parameters

The default creation parameters specify 80 samples in each frame and A-law decoding.

Status Parameters

size	Size of the status parameter structure. If a vendor extends the status parameter structure, <code>size</code> should reflect the size of the extended structure.
frameLen	<p>if (IG711_Cmd == IG711_GETSTATUS) Number of samples passed to <code>decode()</code> for processing.</p> <p>if (IG711_Cmd == IG711_SETSTATUS) Change the number of samples to be passed to <code>decode()</code> for processing to <code>frameLen</code>.</p>
mode	<p>if (IG711_Cmd == IG711_GETSTATUS) IG711_ALAW – Decoding law is A-law. IG711_ULAW – Decoding law is μ-law.</p> <p>if (IG711_Cmd == IG711_SETSTATUS) This is a read-only parameter. Value is ignored.</p>

Name**control - Runtime control and status function****Syntax**

```
val = handle->fxns->control(handle, cmd, status);
```

Parameters

```
IG711DEC_Handle handle;    /* IG711DEC object handle */
IG711_Cmd          cmd;     /* control command */
IG711DEC_Status status;    /* Pointer to status structure */
```

Return Value

```
XDAS_BOOL    val;          /* XDAS_TRUE if call was successful */
```

Preconditions

- `handle` is pointing to a valid G711DEC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `status` is pointing to a valid `IG711DEC_Status` structure.
- `cmd` is a valid `IG711_Cmd`.

Postconditions

```
if (cmd == IG711_GETSTATUS)
```

```
if (val == XDAS_TRUE)
```

The status structure pointed to by `status` was successfully updated reflecting the instance current state.

```
if (val == XDAS_FALSE)
```

The update of the status structure pointed to by `status`, for some reason, failed.

```
if (cmd == IG711_SETSTATUS)
```

```
if (val == XDAS_TRUE)
```

The write parameters in the status structure pointed to by `status` was successfully copied into the instance object.

```
if (val == XDAS_FALSE)
```

The update of the instance status write parameters, for some reason, failed.

Comments

None

See Also

None

Name	decode - Decoder function
Syntax	<code>numSamples = handle->fxns->decode(handle, in, out);</code>
Parameters	<pre>IG711DEC_Handle handle; /* IG711DEC object handle */ XDAS_Int8 *in; /* Pointer to input buffer */ XDAS_Int16 *out; /* Pointer to output buffer */</pre>
Return Value	<pre>XDAS_Int16 numSamples; /* returns number of 14-bit samples in output buffer */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G711DEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>in</code> is pointing to <code>frameLen</code> number of type <code>mode</code> samples.
Postconditions	<pre>if (numSamples < 0)</pre> <p>The decoding of the input samples, for some reason, failed.</p> <pre>if (numSamples >= 0)</pre> <p>The <code>out</code> buffer contains <code>numSamples</code> 14-bit little endian linear PCM samples as described in Figure A-5.</p> <p>The contents of the <code>in</code> buffer do not need to be the same as when entering this function.</p>
Comments	The function should work properly for all possible configurations of <code>frameLen</code> .
See Also	None


```

Includes           #include <XDAS.h>
                    #include <ialg.h>
                    #include <ig711.h>

```

Interface***Types and Constants***

```

/*
 * ===== IG711ENC_Obj =====
 * This structure must be the first field of all G711ENC instance objects.
 */
typedef struct IG711ENC_Obj {
    struct IG711ENC_Fxns *fxns;
} IG711_Obj;
/*
 * ===== IG711ENC_Handle =====
 * This handle is used to reference all G711ENC instance objects.
 */
typedef struct IG711ENC_Obj *IG711ENC_Handle;

```

Creation Parameters

```

/*
 * ===== IG711ENC_Params =====
 * This structure defines the creation parameters for all G711ENC instance
 * objects.
 */
typedef struct IG711ENC_Params {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;      /* Number of samples in a input frame */
    IG711_Mode   mode;         /* Format of the encoded output samples */
} IG711ENC_Params;

```

Status Parameters

```

/*
 * ===== IG711ENC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG711ENC_Status {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Num samples in the in frame (Read/Write) */
    IG711_Mode  mode;         /* Format of encoded out samples (Read-Only) */
} IG711ENC_Status;

```

Functions

```

/*
 * ===== IG711ENC_Fxns =====
 * This structure defines all of the operations on G711ENC objects.
 */
typedef struct IG711ENC_Fxns {
    IALG_Fxns    ialg;        /* IG711ENC extends IALG */
    XDAS_Bool    (*control)(IG711ENC_Handle handle, IG711_Cmd cmd, IG711ENC_Status
                           *status);
    XDAS_Int16   (*encode)(IG711ENC_Handle handle, XDAS_Int16 *in, XDAS_Int8
                           *out);
} IG711ENC_Fxns;

```

Default Creation Parameters

```

/*
 * ===== IG711ENC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G711ENC object.
 */
const IG711ENC_Params IG711ENC_PARAMS = {
    sizeof(IG711ENC_PARAMS), /* Size of this structure */
    80,                      /* 80 samples per frame, 10msec of data */
    IG711_ALAW,              /* Encoded data is A-law format */
};

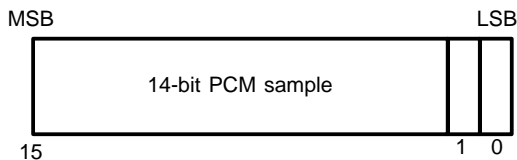
```

Description

The G711ENC module is used in digital telecommunication systems to encode 14-bit linear PCM samples to 8-bit samples according to the ITU G.711 recommendation.

The input to the G711ENC module is 14-bit linear little endian PCM samples shifted towards the MSB. See Figure A-6.

Figure A-6. G711ENC Module Input Sample Format



The output is 8-bit samples encoded according to the A-law or μ -law standards. Each 8-bit sample is shifted towards the LSB for the C54x and consumes 16 bits. In other words, two input samples are not packed into a word. A-law is the standard in North America and Japan, while μ -law is the standard in Europe. Every 14-bit linear input sample is mapped logarithmically to an 8-bit value.

Comments**Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>frameLen</code>	Number of samples in the frame passed to <code>encode()</code> for processing.
<code>mode</code>	IG711_ALAW – A-law decoding. IG711_ULAW - μ -law decoding.

Default Creation Parameters

The default creation parameters specify 80 samples in each input frame and A-law encoding.

Status Parameters

size	Size of the status parameter structure. If a vendor extends the status parameter structure, <code>size</code> should reflect the size of the extended structure.
frameLen	<p>if (IG711_Cmd == IG711_GETSTATUS) Returns the number of samples passed to <code>encode()</code> for processing.</p> <p>if (IG711_Cmd == IG711_SETSTATUS) Set the number of samples to be to passed to <code>encode()</code> for processing to <code>frameLen</code>.</p>
mode	<p>if (IG711_Cmd == IG711_GETSTATUS) IG711_ALAW – Encoding law is A-law. IG711_ULAW – Encoding law is μ-law.</p> <p>if (IG711_Cmd == IG711_SETSTATUS) This is a read-only parameter. Value is ignored.</p>

Name	control - Runtime control and status function
Syntax	<pre>val = handle->fxns->control(handle, cmd, status);</pre>
Parameters	<pre>IG711ENC_Handle handle; /* IG711ENC object handle */ IG711_Cmd cmd; /* control command */ IG711ENC_Status status; /* Pointer to status structure */</pre>
Return Value	<pre>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</pre>
Preconditions	<ul style="list-style-type: none"><input type="checkbox"/> <code>handle</code> is pointing to a valid G711ENC instance object.<input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>.<input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG711ENC_Status</code> structure.<input type="checkbox"/> <code>cmd</code> is a valid <code>IG711_Cmd</code>.
Postconditions	<pre>if (cmd == IG711_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated and reflects the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG711_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was successfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name	encode - Encoder function
Syntax	<code>numSamples = handle->fxns->encode(handle, in, out);</code>
Parameters	<pre>IG711ENC_Handle handle; /* IG711ENC object handle */ XDAS_Int16 *in; /* Pointer to input buffer */ XDAS_Int8 *out; /* Pointer to output buffer */</pre>
Return Value	<pre>XDAS_Int16 numSamples; /* returns number of samples in output buffer */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G711ENC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>in</code> is pointing to 14-bit linear little endian PCM <code>frameLen</code> number of samples as described in Figure A-6.
Postconditions	<pre>if (numSamples < 0)</pre> <p>The encoding of the input samples, for some reason, failed.</p> <pre>if (numSamples >= 0)</pre> <p>The <code>out</code> buffer contains <code>numSamples</code> encoded type <code>mode</code> samples.</p> <p>The contents of the <code>in</code> buffer should be the same as when entering this function.</p>
Comments	The function should work properly for all possible configurations of <code>frameLen</code> .
See Also	None

```

Includes           #include <XDAS.h>
                    #include <ialg.h>
                    #include <ig723.h>

```

Interface***Types and Constants***

```

/*
 * ===== IG723DEC_Obj =====
 * This structure must be the first field of all G723DEC instance objects.
 */
typedef struct IG723DEC_Obj {
    struct IG723DEC_Fxns *fxns;
} IG723DEC_Obj;
/*
 * ===== IG723DEC_Handle =====
 * This handle is used to reference all G723DEC instance objects.
 */
typedef struct IG723DEC_Obj *IG723DEC_Handle;

```

Creation Parameters

```

/*
 * ===== IG723DEC_Params =====
 * This structure defines the creation parameters for all G723DEC instance
 * objects.
 */
typedef struct IG723DEC_Params {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A (Silence Insertion Descriptor frames) */
    XDAS_Bool    pfoEnable;     /* Post Filter enable */
} IG723DEC_Params;

```

Status Parameters

```

/*
 * ===== IG723DEC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG723DEC_Status {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A (SID frames) (Read-Only) */
    XDAS_Bool    badFrame;     /* Bad frame indicator (CRC) (Read/Write) */
    XDAS_Bool    pfoEnable;    /* Post Filter enable */
} IG723DEC_Status;

```

Functions

```

/*
 * ===== IG723DEC_Fxns =====
 * This structure defines all of the operations on G723DEC objects
 */
typedef struct IG723DEC_Fxns {
    IALG_Fxns ialg;           /* IG723DEC extends IALG */
    XDAS_Bool (*control)(IG723DEC_Handle handle, IG723_Cmd cmd, IG723DEC_Status
                        *status);
    XDAS_Int8 (*decode)(IG723DEC_Handle handle, XDAS_Int8 *in, XDAS_Int16 *out);
} IG723DEC_Fxns;

```

Default Creation Parameters

```

/*
 * ===== IG723DEC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G723DEC object.
 */
const IG723DEC_Params IG723DEC_PARAMS = {
    sizeof(IG723DEC_PARAMS), /* Size of this structure */
    XDAS_TRUE,               /* Annex A implementation */
    XDAS_TRUE,               /* Post Filter turned on */
};

```


Description

The G723DEC module is used in multimedia telecommunications to decode ITU G.723.1 dual-rate vocoder bit streams. The interface also supports the silence insertion descriptor (SID) packets scheme described in the ITU G.723.1 Annex A specifications.

The input to the G723DEC module is an ITU G.723.1 specified bit stream. Based on the coding information in the bit stream, the decoder reads 24 8-bit code words for the 6.3 kbps rate or 20 8-bit code words for the 5.3 kbps to produce an output frame of 240 16-bit linear little endian PCM samples. For Annex A, the number of 8-bit code words in the bit stream can also be 4 or 8 (SID packets).

Comments**Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>annexA</code>	<code>XDAS_TRUE</code> – Annex A implementation (SID packets). <code>XDAS_FALSE</code> – Standard implementation.
<code>pfoEnable</code>	<code>XDAS_TRUE</code> – Turn on post filter. <code>XDAS_FALSE</code> – Turn off post filter.

Default Creation Parameters

The default creation parameters specify an annex A implementation with the post filter turned on.

Status Parameters

<code>size</code>	Size of the status parameter structure. If a vendor extends the status parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>annexA</code>	<code>if (IG723_Cmd == IG723_GETSTATUS)</code> <code>XDAS_TRUE</code> – Annex A implementation <code>XDAS_FALSE</code> – Standard implementation <code>if (IG723_Cmd == IG723_SETSTATUS)</code> This is a read-only parameter. Value is ignored.

```
badFrame    if ( IG723_Cmd == IG723_GETSTATUS )
              XDAS_TRUE – Bad frame indicator is turned on for
              the next frame.
              XDAS_FALSE – Bad frame indicator is turned off for
              the next frame.
              if ( IG723_Cmd == IG723_SETSTATUS )
                XDAS_TRUE – Set bad frame indicator for next in-
                coming frame. NOTE: The algorithm is responsible
                for setting badFrame = XDAS_FALSE after proc-
                essing the bad frame.
pfoEnable    if ( IG723_Cmd == IG723_GETSTATUS )
              XDAS_TRUE – Post filter is turned on.
              XDAS_FALSE – Post filter is turned off.
              if ( IG723_Cmd == IG723_SETSTATUS )
                XDAS_TRUE – Turn on high post filter. If post filter
                was already on, this set operation will have no ef-
                fect.
                XDAS_FALSE – Turn off high pass filter. If post pass
                filter was already off, this set operation will have no
                effect.
```

Name	control - Runtime control and status function
Syntax	<pre>val = handle->fxns->control(handle, cmd, status);</pre>
Parameters	<pre>IG723DEC_Handle handle; /* IG723DEC object handle */ IG723_Cmd cmd; /* control command */ IG723DEC_Status status; /* Pointer to status structure */</pre>
Return Value	<pre>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> points to a valid G723DEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> points to a valid <code>IG723DEC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG723_Cmd</code>.
Postconditions	<pre>if (cmd == IG723_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated reflecting the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG723_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> were successfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	After a call to <code>control()</code> with <code>badFrame=XDAS_TRUE</code> , the algorithm is responsible for setting <code>badFrame=XDAS_FALSE</code> after processing the next input frame.
See Also	None

Name	decode - Decoder function
Syntax	<code>numCodeWords = handle->fxns->decode(handle, in, out);</code>
Parameters	<pre>IG723DEC_Handle handle; /* IG726DEC object handle */ XDAS_Int8 *in; /* Pointer to input buffer */ XDAS_Int16 *out; /* Pointer to output buffer */</pre>
Return Value	<code>XDAS_Int8 numCodeWords; /* returns number of 8-bit code words in input buffer */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G723DEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>in</code> is pointing to a valid ITU G.723.1 input bit stream of data.
Postconditions	<pre>if (numCodeWords < 0)</pre> <p>The decoding of the input samples, for some reason, failed.</p> <pre>if (numCodeWords >= 0)</pre> <p>The <code>in</code> buffer contained <code>numCodeWords</code> 8-bit code words.</p> <p>The contents of the <code>in</code> buffer do not need to be the same as when entering this function.</p>
Comments	None
See Also	None

```

Includes           #include <xdas.h>
                    #include <ialg.h>
                    #include <ig723.h>

```

Interface***Types and Constants***

```

/*
 * ===== IG723ENC_Obj =====
 * This structure must be the first field of all G723ENC instance objects.
 */
typedef struct IG723ENC_Obj {
    struct IG723ENC_Fxns *fxns;
} IG723ENC_Obj;

/*
 * ===== IG723ENC_Handle =====
 * This handle is used to reference all G723ENC instance objects.
 */
typedef struct IG723ENC_Obj *IG723ENC_Handle;

```

Creation Parameters

```

/*
 * ===== IG723ENC_Params =====
 * This structure defines the creation parameters for all G723ENC instance
 * objects.
 */
typedef struct IG723ENC_Params {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A (silence compression scheme) */
    XDAS_Bool    hpfEnable;    /* High Pass Filter enable */
    IG723_Rate   rate;         /* Working Rate */
    XDAS_Bool    vadEnable;    /* Voice Activity Detector enable */
                } IG723ENC_Params;

```

Status Parameters

```

/*
 * ===== IG723ENC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG723ENC_Status {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A (Read-Only) */
    XDAS_Bool    hpfEnable;     /* High Pass Filter on/off (Read/Write) */
    IG723_Rate   rate;         /* Working Rate (Read/Write) */
    XDAS_Bool    vadEnable;     /* Voice Activity Detector on/off (Read/Write) */
} IG723ENC_Status;

```

Functions

```

/*
 * ===== IG723ENC_Fxns =====
 * This structure defines all of the operations on G723ENC objects.
 */
typedef struct IG723ENC_Fxns {
    IALG_Fxns    ialg;         /* IG723ENC extends IALG */
    XDAS_Bool    (*control)(IG723ENC_Handle handle, IG723_Cmd cmd,
                           IG723ENC_Status *status);
    XDAS_Int8    (*encode)(IG723ENC_Handle handle, XDAS_Int16 *in, XDAS_Int8
                           *out);
} IG723ENC_Fxns;

```

Default Creation Parameters

```

/*
 * ===== IG723ENC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G723ENC object.
 */
const IG723ENC_Params IG723ENC_PARAMS = {
    sizeof(IG723ENC_PARAMS),      /* Size of this structure */
    XDAS_TRUE,                    /* Annex A implementation */
    XDAS_TRUE,                    /* High Pass Filter is turned on */
    IG723_5300BPS,               /* Coding rate is 5.3kbps */
    XDAS_TRUE,                    /* VAD turned on (if Annex A) */
};

```

Description

The G723ENC module is used in multimedia telecommunication systems to encode 16-bit linear PCM speech according to the ITU G.723.1 recommendation.

The interface also supports the silence compression scheme described in the ITU G.723.1 Annex A specifications.

The input to the G723ENC module is a frame of 240 samples of 16-bit linear little endian PCM samples. The output for each frame is 24-bit code words for the 6.3 kbps rate or 20-bit code words for the 5.3 kbps rate. For Annex A, if the voice activity detector (VAD) (silence compression scheme) is enabled, the encoder will insert SID packets for input frames with no voice activity.

Comments

Creation Parameters

size	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
annexA	XDAS_TRUE – Annex A (silence compression scheme) implementation. XDAS_FALSE – Standard implementation.
hpfEnable	XDAS_TRUE – Turn on high pass filter. XDAS_FALSE – Turn off high pass filter.

rate IG723_5300BPS – 5.3kbps output rate.
 IG723_6300BPS – 6.3kbps output rate.

vadEnable if (annexA == XDAS_TRUE)
 XDAS_TRUE – Turn on voice activity detector.
 XDAS_FALSE – Turn off voice activity detector.
 if (annexA == XDAS_FALSE)
 Value is ignored.

NOTE: if (annexA == XDAS_TRUE) and the implementation does not support Annex A, the `algInit()` function of the algorithm should fail.

Default Creation Parameters

The default creation parameters specifies an Annex A implementation, with the high pass filter and the voice activity detector to be turned on, and the coding rate to 5.3 kbps.

Status Parameters

size Size of the status parameter structure. If a vendor extends the status parameter structure, `size` should reflect the size of the extended structure.

annexA if (IG723_Cmd == IG723_GETSTATUS)
 XDAS_TRUE – Annex A implementation
 XDAS_FALSE – Standard implementation.
 if (IG723_Cmd == IG723_SETSTATUS)
 This is a read-only parameter. Value is ignored.

hpfEnable if (IG723_Cmd == IG723_GETSTATUS)
 XDAS_TRUE – High pass filter is turned on.
 XDAS_FALSE – High pass filter is turned off.
 if (IG723_Cmd == IG723_SETSTATUS)
 XDAS_TRUE – Turn on high pass filter. If high pass filter was already on, this set operation will have no effect.
 XDAS_FALSE – Turn off high pass filter. If high pass filter was already off, this set operation will have no effect.


```
rate      if (IG723_Cmd == IG723_GETSTATUS)
           IG723_53000BPS – Current encoding rate is 5.3
           kbps.
           IG723_63000BPS – Current encoding rate is 6.3
           kbps.
           if (IG723_Cmd == IG723_SETSTATUS)
           IG723_53000BPS – Set encoding rate to 5.3 kbps.
           IG723_63000BPS – Set encoding rate to 6.3 kbps.
vadEnable if (IG723_Cmd == IG723_GETSTATUS)
           if (annexA == XDAS_TRUE)
           XDAS_TRUE – Voice activity detector is turned
           on.
           XDAS_FALSE – Voice activity detector is turned
           off.
           if (annexA == XDAS_FALSE)
           No VAD support. Value is ignored.
           if (IG723_Cmd == IG723_SETSTATUS)
           if (annexA == XDAS_TRUE)
           XDAS_TRUE – Turn on voice activity detector.
           XDAS_FALSE – Turn off voice activity detector.
           if (annexA == XDAS_FALSE)
           No VAD support. Value is ignored.
```

Name	control - Runtime control and status function
Syntax	<pre>val = handle->fxns->control(handle, cmd, status);</pre>
Parameters	<pre>IG723ENC_Handle handle; /* IG723ENC object handle */ IG723_Cmd cmd; /* control command */ IG723ENC_Status status; /* Pointer to status structure */</pre>
Return Value	<pre>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G723ENC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG723ENC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG723_Cmd</code>.
Postconditions	<pre>if (cmd == IG723_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated reflecting the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG723_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was suc- cessfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name	encode - Encoder function
Syntax	<pre>numCodeWords = handle->fxns->encode(handle, in, out);</pre>
Parameters	<pre>IG723ENC_Handle handle; /* IG723ENC object handle */ XDAS_Int16 *in; /* Pointer to input buffer */ XDAS_Int8 *out; /* Pointer to output buffer */</pre>
Return Value	<pre>XDAS_Int16 numCodeWords; /* returns number of 8-bit code words in out buffer */</pre>
Preconditions	<ul style="list-style-type: none"><input type="checkbox"/> <code>handle</code> is pointing to a valid G723ENC instance object.<input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>.<input type="checkbox"/> <code>in</code> is pointing to 240 16-bit linear little endian PCM samples.
Postconditions	<pre>if (numCodeWords < 0)</pre> <p>The encoding of the input samples, for some reason, failed.</p> <pre>if (numCodeWords >= 0)</pre> <p>The <code>out</code> buffer contains <code>numCodeWords</code> bytes.</p> <p>The contents of the <code>in</code> buffer should be the same as when entering this function.</p>
Comments	None
See Also	None

G726DEC**G.726 Decoder Abstract Interface****Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ig726.h>
```

Interface**Types and Constants**

```
/*
 * ===== IG726DEC_Obj =====
 * This structure must be the first field of all G726DEC instance objects.
 */
typedef struct IG726DEC_Obj {
    struct IG726DEC_Fxns *fxns;
} IG726DEC_Obj;
/*
 * ===== IG726DEC_Handle =====
 * This handle is used to reference all G726DEC instance objects.
 */
typedef struct IG726DEC_Obj *IG726DEC_Handle;
```

Creation Parameters

```
/*
 * ===== IG726DEC_Params =====
 * This structure defines the creation parameters for all G726DEC instance
 * objects.
 */
typedef struct IG726DEC_Params {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Length of input buffer */
    IG726_Mode   mode;         /* Format of the output buffer */
    IG726_Rate   rate;         /* Working rate */
} IG726DEC_Params;
```

Status Parameters

```
/*
 * ===== IG726DEC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG726DEC_Status {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Length of input buffer (Read/Write) */
    IG726_Mode   mode;         /* Format of the decoded buffer (Read-Only) */
    IG726_Rate   rate;         /* Working rate (Read-Write) */
} IG726DEC_Status;
```

Functions

```
/*
 * ===== IG726DEC_Fxns =====
 * This structure defines all of the operations on G726DEC objects.
 */
typedef struct IG726DEC_Fxns {
    IALG_Fxns    ialg;         /* IG726DEC extends IALG */
    XDAS_Bool    (*control)(IG726DEC_Handle handle, IG726_Cmd cmd,
                            IG726DEC_Status *status);
    XDAS_Int16   (*decode)(IG726DEC_Handle handle, XDAS_Int8 *out, XDAS_Int16
                            *in);
} IG726DEC_Fxns;
```

Default Creation Parameters

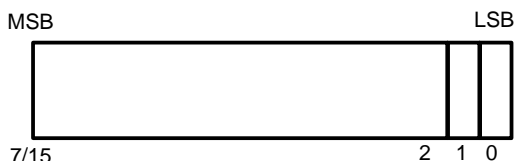
```
/*
 * ===== IG726DEC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G726DEC object.
 */
const IG726DEC_Params IG726DEC_PARAMS = {
    sizeof(IG726DEC_PARAMS), /* Size of this structure */
    1,                       /* Sample by sample processing */
    IG726_ALAW,              /* Out buffer is A-law */
    IG726_16KBPS,           /* Working rate is 16kbps */
};
```

Description

The G726DEC module is used to decode ITU G.726 Adaptive Differential Pulse Code Modulated (ADPCM) voice samples. It decodes a 40, 32, 24, or a 16 kbits channel to 64 kbps.

The input to the G726DEC module is a stream of samples (each sample consumes 8-bits for C6x and 16-bit for C54x) containing 2, 3, 4, or 5 bits of information depending on the encoding rate. The bits of information are shifted towards the LSB, i.e., no packing of the input bits. The output of the decoder are 16-bit PCM linear little endian samples. See Figure A-7 for an example with 16 kbps encoding rate and the 2-bit information shifted towards the LSB.

Figure A-7. G726DEC Input Sample Format

**Comments****Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, size should reflect the size of the extended structure.
<code>frameLen</code>	Number of samples passed to <code>decode()</code> for processing.
<code>mode</code>	IG726_ALAW - A-law encoded output data. IG726_ULAW - μ -law encoded output data. IG726_LINEAR - Linear 16-bit little endian encoded output data.
<code>rate</code>	IG726_16KBPS - 16 kbps input rate. IG726_24KBPS - 24 kbps input rate. IG726_32KBPS - 32 kbps input rate. IG726_40KBPS - 40 kbps input rate.

Default Creation Parameters

The default creation parameters are one sample input frame encoded at 16 kbps, and the output data to be A-law.

Status Parameters

size	Size of the status parameter structure. If a vendor extends the status parameter structure, size should reflect the size of the extended structure.
frameLen	<p>if (IG726_Cmd == IG726_GETSTATUS) Returns the number of samples in the input frame.</p> <p>if (IG726_Cmd == IG726_SETSTATUS) Set the number of samples to be decoded in an input frame to frameLen.</p>
mode	<p>if (IG726_Cmd == IG726_GETSTATUS) IG726_ALAW – Output data is A-law. IG726_ULAW – Output data is μ-law. IG726_LINEAR – Output data is 16-bit linear PCM.</p> <p>if (IG726_Cmd == IG726_SETSTATUS) This is a read-only parameter. Value is ignored.</p>
rate	<p>if (IG726_Cmd == IG726_GETSTATUS) IG726_16KBPS – 16 kbps input data rate. IG726_24KBPS – 24 kbps input data rate. IG726_32KBPS – 32 kbps input data rate. IG726_40KBPS – 40 kbps input data rate.</p> <p>if (IG726_Cmd == IG726_SETSTATUS) This is a read-only parameter. Value is ignored.</p>

Name	control - Runtime control and status function
Syntax	<code>val = handle->fxns->control(handle, cmd, status);</code>
Parameters	<pre>IG726DEC_Handle handle; /* IG726DEC object handle */ IG726_Cmd cmd; /* control command */ IG726DEC_Status status; /* Pointer to status structure */</pre>
Return Value	<code>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> points to a valid G726DEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> points to a valid <code>IG726DEC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG726_Cmd</code>.
Postconditions	<pre>if (cmd == IG726_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated reflecting the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG726_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was suc- cessfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name**decode - Decoder function****Syntax**

```
numSamples = handle->fxns->decode(handle, in, out)
```

Parameters

```
IG726DEC_Handle  handle;          /* IG726DEC object handle */
XDAS_Int8        *in;             /* Pointer to input buffer */
XDAS_Int16       *out;            /* Pointer to output buffer */
```

Return Value

```
XDAS_Int16 numSamples; /* returns number of samples
                        in output buffer */
```

Preconditions

- `handle` is pointing to a valid G726DEC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `in` is pointing to `frameLen` number of encoded samples.

Postconditions

```
if (numSamples < 0)
```

The decoding of the `in` buffer, for some reason, failed.

```
if (numSamples >= 0)
```

The `out` buffer contains `numSamples` decoded samples.

The contents of the `in` buffer do not need to be the same as when entering this function.

Comments

The function should work properly for all possible configurations `frameLen`.

See Also

None

G726ENC**G.726 Encoder Abstract Interface****Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ig726.h>
```

Interface**Types and Constants**

```
/*
 * ===== IG726ENC_Obj =====
 * This structure must be the first field of all G726ENC instance objects.
 */
typedef struct IG726ENC_Obj {
    struct IG726ENC_Fxns *fxns;
} IG726ENC_Obj;
/*
 * ===== IG726ENC_Handle =====
 * This handle is used to reference all G726ENC instance objects.
 */
typedef struct IG726ENC_Obj *IG726ENC_Handle;
```

Creation Parameters

```
/*
 * ===== IG726ENC_Params =====
 * This structure defines the creation parameters for all G726ENC objects
 */
typedef struct IG726ENC_Params {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Length of output buffer */
    IG726_Mode   mode;         /* Format of the encoded buffer */
    IG726_Rate   rate;         /* Working rate */
} IG726ENC_Params;
```

Status Parameters

```
/*
 * ===== IG726ENC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG726ENC_Status {
    Int          size;          /* Size of this structure */
    XDAS_UInt16  frameLen;     /* Length of input buffer (Read/Write) */
    IG726_Mode   mode;         /* Format of the decoded buffer (Read-Only) */
    IG726_Rate   rate;         /* Working rate (Read/Write) */
} IG726ENC_Status;
```

Functions

```
/*
 * ===== IG726ENC_Fxns =====
 * This structure defines all of the operations on G726ENC objects.
 */
typedef struct IG726ENC_Fxns {
    IALG_Fxns    ialg;        /* IG726ENC extends IALG */
    XDAS_Bool    (*control)(IG726ENC_Handle handle, IG726_Cmd cmd,
                            IG726ENC_Status *status);

    XDAS_Int16   (*encode)(IG726ENC_Handle handle, XDAS_Int16 *in, XDAS_Int8
*out);
} IG726ENC_Fxns;
```

Default Creation Parameters

```
/*
 * ===== IG726ENC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G726ENC object.
 */
const IG726ENC_Params IG726ENC_PARAMS = {
    sizeof(IG726ENC_PARAMS),    /* Size of this structure */
    1,                          /* Sample by sample processing */
    IG726_ALAW,                 /* Input buffer format is A-law */
    IG726_16KBPS,               /* Working rate is 16kbps */
};
```

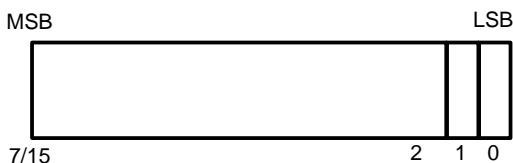
Description

The G726ENC module encodes voice samples according to the ITU G.726 Adaptive Differential Pulse Code Modulation (ADPCM) specification.

The input to the G726ENC module is either 8 bits A-law or μ -law samples, or linear little endian 16-bit PCM samples. Each 8-bit sample is shifted towards the LSB for the C54x and consumes 16 bits. In other words, two input samples are not packed into a word.

The output of the encoder is a sample (8-bits for C6x and 16-bits for C54x) containing either 2-bits, 3-bits, 4-bits, or 5-bits of information depending on the configured coding rate. The 2, 3, 4, or 5 bits of information is shifted towards the LSB (i.e., no packing). See Figure A-8 for an example where the encoding rate is 16 kbps.

Figure A-8. G726ENC Module Output Sample Format

**Comments****Creation Parameters**

size	Size of the creation parameter structure. If a vendor extends the creation parameter structure, size should reflect the size of the extended structure.
frameLen	Number of samples in the frame passed to <code>encode()</code> for processing.
mode	IG726_ALAW - A-law encoded input data. IG726_ULAW - μ -law encoded input data. IG726_LINEAR - Linear 16-bit little endian encoded input data.
rate	IG726_16KBPS - 16 kbps output rate. IG726_24KBPS - 24 kbps output rate. IG726_32KBPS - 32 kbps output rate. IG726_40KBPS - 40 kbps output rate.

Default Creation Parameters

The default creation parameters specifies a one sample input frame of A-law encoded data, and an output rate of at 16 kbps.

Status Parameters

size	Size of the status parameter structure. If a vendor extends the status parameter structure, size should reflect the size of the extended structure.
frameLen	<p>if (IG726_Cmd == IG726_GETSTATUS) Returns the number of samples in the input frame.</p> <p>if (IG726_Cmd == IG726_SETSTATUS) Set the number of samples to be encoded in an input frame to frameLen.</p>
mode	<p>if (IG726_Cmd == IG726_GETSTATUS) IG726_ALAW – Input data is A-law. IG726_ULAW – Input data is μ-law. IG726_LINEAR – Input data is 16-bit linear PCM.</p> <p>if (IG726_Cmd == IG726_SETSTATUS) This is a read-only parameter. Value is ignored.</p>
rate	<p>if (IG726_Cmd == IG726_GETSTATUS) IG726_16KBPS – 16 kbps output rate. IG726_24KBPS – 24 kbps output rate. IG726_32KBPS – 32 kbps output rate. IG726_40KBPS – 40 kbps output rate.</p> <p>if (IG726_Cmd == IG726_SETSTATUS) This is a read-only parameter. Value is ignored.</p>

Name	control - Runtime control and status function
Syntax	<pre>val = handle->fxns->control(handle, cmd, status);</pre>
Parameters	<pre>IG726ENC_Handle handle; /* IG726ENC object handle */ IG726_Cmd cmd; /* control command */ IG726ENC_Status status; /* Pointer to status structure */</pre>
Return Value	<pre>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G76ENC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG726ENC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG726_Cmd</code>.
Postconditions	<pre>if (cmd == IG726_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated reflecting the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG726_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was successfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name**encode - Encoder function****Syntax**

```
numSamples = handle->fxns->encode(handle, in, out);
```

Parameters

```
IG726ENC_Handle handle;    /* IG726ENC object handle */
XDAS_Int16      *in;       /* Pointer to input buffer */
XDAS_Int8       *out;      /* Pointer to output buffer */
```

Return Value

```
XDAS_Int16 numSamples;     /* returns number of samples
                             in encoded buffer */
```

Preconditions

- `handle` is pointing to a valid G726ENC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `in` is pointing to `frameLen` number of type `mode` samples.

Postconditions

```
if (numSamples < 0)
```

The encoding of the input samples for some reason failed.

```
if (numSamples >= 0)
```

The out buffer contains `numSamples` encoded samples.

The contents of the `in` buffer should be the same as when entering this function.

Comments

None

See Also

None

G728DEC**G.728 Decoder Abstract Interface****Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ig728.h>
```

Interface**Types and Constants**

```
/*
 * ===== IG728DEC_Handle =====
 * This handle is used to reference all G728DEC instance objects
 */
typedef struct IG728DEC_Obj *IG728DEC_Handle;
/*
 * ===== IG728DEC_Obj =====
 * This structure must be the first field of all G728DEC instance objects
 */
typedef struct IG728DEC_Obj {
    struct IG728DEC_Fxns *fxns;
} IG728DEC_Obj;
```

Creation Parameters

```
/*
 * ===== IG728DEC_Params =====
 * This structure defines the creation parameters for all G728DEC objects
 */
typedef struct IG728DEC_Params {
    Int          size;          /* Size of this structure */
    IG728_Mode   mode;         /* Format of the out buffer */
    XDAS_Bool    pfoEnable;    /* Enable/Disable postfilter */
    XDAS_Int8    syncPeriod;   /* Positive value measured in codewords, */
} IG728DEC_Params;          /* zero disables inband sync */
```


Status Parameters

```
/*
 * ===== IG728DEC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG728DEC_Status {
    Int          size;          /* Size of this structure */
    IG728_Mode   mode;         /* Format of the out buffer (Read/Write) */
    XDAS_Bool    pfoEnable;    /* Enable/Disable postfilter (Read/Write) */
    XDAS_Int8    syncPeriod;   /* Positive value measured in codewords, */
} IG728DEC_Status;          /* zero disables inband sync (Read/Write) */
```

Functions

```
/*
 * ===== IG728DEC_Fxns =====
 * This structure defines all of the operations on G728DEC objects
 */
typedef struct IG728DEC_Fxns {
    IALG_Fxns    ialg;        /* IG728DEC extends IALG */
    XDAS_Bool    *(control)(IG728DEC_Handle handle, IG728_Cmd cmd,
                            IG728DEC_Status *status);

    XDAS_Int8    (*decode)(IG728DEC_Handle handle, XDAS_Int8 *in, XDAS_Int16
*out);
} IG728DEC_Fxns;
```

Default Creation Parameters

```
/*
 * ===== IG728DEC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G728DEC object.
 */
const IG728DEC_Params IG728DEC_PARAMS = {
    sizeof(IG728DEC_PARAMS), /* Size of this structure */
    IG728_ALAW,             /* Out buffer is A-law */
    XDAS_TRUE,              /* Post Filter is tuned on */
    0,                      /* Inband synch is disabled */
};
```

Description The G728DEC module is used in applications such as video telecommunications to decode ITU G.728 voice encoded frames.

The input to the G728DEC module is a 10-bit code word, and the output is a frame of 5 A-law, μ -law or 16-bit linear PCM little endian samples.

Comments

Creation Parameters

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>mode</code>	IG728_ALAW - A-law output data. IG728_ULAW - μ -law output data. IG728_LINEAR - Linear 16-bit little endian output data.
<code>pfoEnable</code>	XDAS_TRUE – Turn on post filter. XDAS_FALSE – Turn off post filter.
<code>syncPeriod</code>	0 – Disable inband Synch num – specify number of codewords for inband synchronization.

Default Creation Parameters

The default creation parameters specify an decoder with the post filter tuned on, inband synchronization turned off, and A-law output data.

Status Parameters

<code>size</code>	Size of the status parameter structure. If a vendor extends the status parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>mode</code>	if (IG728_Cmd == IG728_GETSTATUS) IG728_ALAW – Output data format is A-law. IG728_ULAW – Output data format is μ -law. IG728_LINEAR – Output data format is 16-bit linear PCM little endian. if (IG728_Cmd == IG728_SETSTATUS) This is a read-only parameter. Value is ignored.

```
pfoEnable  if (IG728_Cmd == IG728_GETSTATUS)
            XDAS_TRUE – Post filter is turned on.
            XDAS_FALSE – Post filter is turned off.
            if (IG728_Cmd == IG728_SETSTATUS)
                XDAS_TRUE – Turn on post filter.
                XDAS_FALSE – Turn off post filter.
syncPeriod if (IG728_Cmd == IG728_GETSTATUS)
            Return number of inband synchronization periods.
            if (IG728_Cmd == IG728_SETSTATUS)
                Set number of inband synchronization periods to
                syncPeriod.
```

Name	control - Runtime control and status function
Syntax	<code>val = handle->fxns->control(handle, cmd, status);</code>
Parameters	<pre>IG728DEC_Handle handle; /* IG728DEC object handle */ IG728_Cmd cmd; /* control command */ IG728DEC_Status status; /* Pointer to status structure */</pre>
Return Value	<code>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G728DEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG728DEC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG728_Cmd</code>.
Postconditions	<pre>if (cmd == IG728_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated reflecting the current state of the instance. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG728_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was suc- cessfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name**decode - Decoder function****Syntax**

```
numSamples = handle->fxns->decode(handle, in, out);
```

Parameters

```
IG728DEC_Handle  handle; /* IG728DEC object handle */
XDAS_Int8        *in;    /* Pointer to input buffer */
XDAS_Int16       *out;   /* Pointer to output buffer */
```

Return Value

```
XDAS_Int16  numSamples; /* returns number of samples in
                        output buffer */
```

Preconditions

- `handle` is pointing to a valid G728DEC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `in` is pointing to a 10-bit G728 ITU code word.

Postconditions

```
if (numSamples < 0)
```

The decoding of the input samples, for some reason, failed.

```
if (numSamples >= 0)
```

The `out` buffer contains `numSamples` samples of type `mode`.

The contents of the `in` buffer do not need to be the same as when entering this function.

Comments

None

See Also

None

G728ENC**G.728 Encoder Abstract Interface****Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ig728.h>
```

Interface**Types and Constants**

```
/*
 * ===== IG728ENC_Handle =====
 * This handle is used to reference all G728ENC instance objects
 */
typedef struct IG728ENC_Obj *IG728ENC_Handle;
/*
 * ===== IG728ENC_Obj =====
 * This structure must be the first field of all G728ENC instance objects
 */
typedef struct IG728ENC_Obj {
    struct IG728ENC_Fxns *fxns;
} IG728ENC_Obj;
```

Creation Parameters

```
/*
 * ===== IG728ENC_Params =====
 * This structure defines the creation parameters for all G728ENC objects
 */
typedef struct IG728ENC_Params {
    Int          size;          /* Size of this structure */
    IG728_Mode   mode;         /* Format of the in buffer */
    XDAS_Bool    pwfEnable;    /* Enable/Disable perceptual weighting filter */
    XDAS_Int8    syncPeriod;   /* Positive value measured in codewords, */
} IG728ENC_Params;          /* zero disables inband sync */
```

Status Parameters

```
/*
 * ===== IG728ENC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG728ENC_Status {
    Int          size;          /* Size of this structure */
    IG728_Mode   mode;         /* Format of the in buffer (Read/Write) */
    XDAS_Bool    pwfEnable;    /* Enable/Disable pw filter (Read/Write) */
    XDAS_Int8    syncPeriod;   /* Positive value measured in codewords, */
} IG728ENC_Status;          /* zero disables inband sync (Read/Write) */
```

Functions

```
/*
 * ===== IG728ENC_Fxns =====
 * This structure defines all of the operations on G728ENC objects
 */
typedef struct IG728ENC_Fxns {
    IALG_Fxns ialg;           /* IG728ENC extends IALG */
    XDAS_Bool (*control)(IG728ENC_Handle handle, IG728_Cmd cmd,
                        IG728ENC_Status *status);
    XDAS_Int8 (*encode)(IG728ENC_Handle handle, XDAS_Int16 *in, XDAS_Int8 *out);
} IG728ENC_Fxns;
```

Default Creation Parameters

```
/*
 * ===== IG728ENC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G728ENC object.
 */
const IG728ENC_Params IG728ENC_PARAMS = {
    sizeof(IG728ENC_PARAMS), /* Size of this structure */
    IG728_ULAW,              /* Input buffer format is u-law */
    XDAS_TRUE,               /* Perceptual weighting filter in turned on */
    0,                       /* Inband sync disabled */
};
```

Description

The G728ENC module is used in applications such as video telecommunications to encode audio samples at 16kHz according to the ITU G.728 recommendation.

The input to the G728ENC module is a frame of 5 A-law, μ -law or 16-bit linear little endian samples of audio sampled at 16kHz. The output of the encoder is a 10-bit codeword for each input frame

Comments**Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>mode</code>	IG728_ALAW - A-law encoded input data. IG728_ULAW - μ -law encoded input data. IG728_LINEAR - Linear 16-bit little endian encoded input data.
<code>pwfEnable</code>	XDAS_TRUE – Turn on perceptual weighting filter. XDAS_FALSE – Turn off perceptual weighting filter.
<code>syncPeriod</code>	0 – Disable inband Synch num – specify number of codewords for inband synchronization.

Default Creation Parameters

The default creation parameters specify an encoder working on A-law input data with the perceptual weighting filter turned on and inband synchronization turned off.

Status Parameters

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>mode</code>	<code>if (IG728_Cmd == IG728_GETSTATUS)</code> <code>IG728_ALAW</code> – Input data format is A-law. <code>IG728_ULAW</code> – Input data format is μ -law. <code>IG728_LINEAR</code> – Input data format is 16-bit PCM linear little endian. <code>if (IG728_Cmd == IG728_SETSTATUS)</code> This is a read-only parameter. Value is ignored.
<code>pwfEnable</code>	<code>if (IG728_Cmd == IG728_GETSTATUS)</code> <code>XDAS_TRUE</code> – Perceptual weighting filter is turned on. <code>XDAS_FALSE</code> – Perceptual weighting filter is turned off. <code>if (IG728_Cmd == IG728_SETSTATUS)</code> <code>XDAS_TRUE</code> – Turn on perceptual weighting filter. <code>XDAS_FALSE</code> – Turn off perceptual weighting filter.
<code>syncPeriod</code>	<code>if (IG728_Cmd == IG728_GETSTATUS)</code> Return number of inband synchronization periods. <code>if (IG728_Cmd == IG728_SETSTATUS)</code> Set number of inband synchronization periods to <code>syncPeriod</code> .

Name	control - Runtime control and status function
Syntax	<code>val = handle->fxns->control(handle, cmd, status);</code>
Parameters	<pre>IG728ENC_Handle handle; /* IG728ENC object handle */ IG728_Cmd cmd; /* control command */ IG728ENC_Status status; /* Pointer to status structure */</pre>
Return Value	<code>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G728 instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG728ENC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG728_Cmd</code>.
Postconditions	<pre>if (cmd == IG728_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated and reflects the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG728_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was successfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name **encode - Encoder function**

Syntax `numBytes = handle->fxns->encode(handle, in, out);`

Parameters

```
IG728ENC_Handle  handle; /* IG728ENC object handle */
XDAS_Int16      *in;     /* Pointer to input buffer */
XDAS_Int8       *out;    /* Pointer to output buffer */
```

Return Value `XDAS_Int16 numBytes; /* returns number of bytes in encoded buffer */`

Preconditions

- `handle` is pointing to a valid G728ENC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `in` is pointing to 5 type `mode` samples sampled at 16kHz.

Postconditions

```
if (numBytes < 0)
```

The encoding of the input samples, for some reason, failed.

```
if (numBytes >= 0)
```

The `out` buffer contains `numBytes` bytes encoded samples.

The contents of the `in` buffer should be the same as when entering this function.

Comments None

See Also None

G729DEC**G.729 Decoder Abstract Interface****Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ig729.h>
```

Interface**Types and Constants**

```
/*
 * ===== IG729DEC_Obj =====
 * This structure must be the first field of all G729DEC instance objects.
 */
typedef struct IG729DEC_Obj {
    struct IG729DEC_Fxns *fxns;
} IG729DEC_Obj;
/*
 * ===== IG729DEC_Handle =====
 * This handle is used to reference all G729DEC instance objects.
 */
typedef struct IG729DEC_Obj *IG729DEC_Handle;
```

Creation Parameters

```
/*
 * ===== IG729DEC_Params =====
 * This structure defines the parameters necessary to create an
 * instance of an G729 decoder object.
 */
typedef struct IG729DEC_Params {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A (reduced complexity decoding) */
    XDAS_Bool    annexB;       /* Annex B (Silence Insertion Descriptors frames) */
    XDAS_Bool    pfoEnable;    /* Post Filter enable */
} IG729DEC_Params;
```

Status Parameters

```
/*
 * ===== IG729DEC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG729DEC_Status {
    Int            size;          /* Size of this structure */
    XDAS_Bool     annexA;       /* Annex A (reduced complexity) (Read-Only) */
    XDAS_Bool     annexB;       /* Annex B (SID frames) (Read-Only) */
    XDAS_Bool     pfoEnable;     /* Post Filter on/off (Read/Write) */
} IG729DEC_Status;
```

Functions

```
/*
 * ===== IG729DEC_Fxns =====
 * This structure defines all of the operations on G729DEC objects.
 */
typedef struct IG729DEC_Fxns {
    IALG_Fxns ialg;
    XDAS_Bool (*control)(IG729DEC_Handle handle, IG729_Cmd cmd, IG729DEC_Status
                        *status);
    XDAS_Int8 (*decode)(IG729DEC_Handle handle, XDAS_Int8 *in, XDAS_Int16 *out,
                       XDAS_UInt8 packetSize);
} IG729DEC_Fxns;
```

Default Creation Parameters

```
/*
 * ===== IG729DEC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G729DEC object.
 */
const IG729DEC_Params IG729DEC_PARAMS = {
    sizeof(IG729DEC_PARAMS), /* Size of this structure */
    XDAS_TRUE,               /* Annex A implementation */
    XDAS_TRUE,               /* Annex B implementation */
    XDAS_TRUE,               /* Post Filter is tuned on */
};
```

Description

The G729DEC module is used in, for example, videoconferencing, multimedia and voice-email to decode ITU G.729 input frames. The interface also supports the reduced complexity version described in Annex A, and the silence compression scheme described in the Annex B.

The input to the G729DEC module is an ITU G.729-specified frame of 10 8-bit code words. The output is a frame of 80 samples of 16-bit little endian linear PCM data.

Annex A describes a reduced complexity version of the G.729 standard. They are, however, fully interoperable, so an annex A encoded packet can be decoded by a standard decoder, and vice versa.

Comments**Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>annexA</code>	<code>XDAS_TRUE</code> – Annex A (reduced complexity) implementation. <code>XDAS_FALSE</code> – Standard implementation.
<code>annexB</code>	<code>XDAS_TRUE</code> – Annex B (silence compression scheme) implementation. <code>XDAS_FALSE</code> – Standard implementation.
<code>pfoEnable</code>	<code>XDAS_TRUE</code> – Turn on post filter. <code>XDAS_FALSE</code> – Turn off post filter.

Default Creation Parameters

The default creation parameters specifies an annex A and annex B implementation with the post filter turned on.

Status Parameters

size	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
annexA	<pre>if (IG729_Cmd == IG729_GETSTATUS) XDAS_TRUE – Annex A implementation XDAS_FALSE – Standard implementation. if (IG729_Cmd == IG729_SETSTATUS) This is a read-only parameter; value is ignored.</pre>
annexB	<pre>if (IG729_Cmd == IG729_GETSTATUS) XDAS_TRUE – Annex B implementation XDAS_FALSE – Standard implementation. if (IG729_Cmd == IG729_SETSTATUS) This is a read-only parameter, value is ignored.</pre>
pfoEnable	<pre>if (IG729_Cmd == IG729_GETSTATUS) XDAS_TRUE – Post filter is turned on. XDAS_FALSE – Post filter is turned off. if (IG729_Cmd == IG729_SETSTATUS) XDAS_TRUE – Turn on high-post filter. If post filter was already on, this set operation will have no ef- fect. XDAS_FALSE – Turn off high-pass filter. If post pass filter was already off, this set operation will have no effect.</pre>

Name	control - Runtime control and status function
Syntax	<code>val = handle->fxns->control(handle, cmd, status);</code>
Parameters	<pre>IG729DEC_Handle handle; /* IG729DEC object handle */ IG729_Cmd cmd; /* control command */ IG729DEC_Status status; /* Pointer to status structure */</pre>
Return Value	<code>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G729DEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG729DEC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG729_Cmd</code>.
Postconditions	<pre>if (cmd == IG729_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated and reflects the instance current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG729_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code> was successfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name**decode - Decoder function****Syntax**

```
numSamples = handle->fxns->decode(handle, in, out, packet-
Size);
```

Parameters

```
IG729DEC_Handle handle; /* IG729DEC object handle */
XDAS_Int8      *in;     /* Pointer to input buffer */
XDAS_Int16     *out;    /* Pointer to output buffer */
XDAS_UInt8     packetSize; /* Size of the input packet (number
                           of octets) */
```

Return Value

```
XDAS_Int8     numSamples; /* returns number samples in output
                           buffer */
```

Preconditions

- `handle` is pointing to a valid G729DEC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `in` is pointing to a valid ITU G.729 input packet.

Postconditions

```
if (numSamples < 0)
```

Decoding of the input samples, for some reason, failed.

```
if (numSamples >= 0)
```

The `out` buffer contains `numSamples` linear 16-bit little endian PCM samples.

The contents of the `in` buffer do not need to be the same as when entering this function.

Comments

None.

See Also

None.

G729ENC**G.729 Encoder Abstract Interface****Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ig729.h>
```

Interface**Types and Constants**

```
/*
 * ===== IG729ENC_Obj =====
 * This structure must be the first field of all G729ENC instance objects.
 */
typedef struct IG729ENC_Obj {
    struct IG729ENC_Fxns *fxns;
} IG729ENC_Obj;
/*
 * ===== IG729ENC_Handle =====
 * This handle is used to reference a G729ENC instance object.
 */
typedef struct IG729ENC_Obj *IG729ENC_Handle;
```

Creation Parameters

```
/*
 * ===== IG729ENC_Params =====
 * This structure defines the creation parameters for all G729ENC instance
 * objects.
 */
typedef struct IG729ENC_Params {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A (reduced encoder complexity) */
    XDAS_Bool    annexB;       /* Annex B (silence compression scheme) */
    XDAS_Bool    vadEnable;     /* Voice activity detector */
} IG729ENC_Params;
```

Status Parameters

```
/*
 * ===== IG729ENC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct IG729ENC_Status {
    Int          size;          /* Size of this structure */
    XDAS_Bool    annexA;       /* Annex A implementation (Read-Only) */
    XDAS_Bool    annexB;       /* Annex B implementation (Read-only) */
    XDAS_Bool    vadEnable;    /* Voice activity detector (Read/Write) */
} IG729ENC_Status;
```

Functions

```
/*
 * ===== IG729ENC_Fxns =====
 * This structure defines all of the operations on G729ENC objects.
 */
typedef struct IG729ENC_Fxns {
    IALG_Fxns    ialg;
    XDAS_Bool    (*control)(IG729ENC_Handle handle, IG729_Cmd cmd,
                           IG729ENC_Status *status);
    XDAS_Int8    (*encode)(IG729ENC_Handle handle, XDAS_Int16 *in, XDAS_Int8 *out);
} IG729ENC_Fxns;
```

Default Creation Parameters

```
/*
 * ===== IG729ENC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a G729ENC object.
 */
const IG729ENC_Params IG729ENC_PARAMS = {
    sizeof(IG729ENC_PARAMS),    /* Size of this structure */
    XDAS_TRUE,                  /* Annex A implementation */
    XDAS_TRUE,                  /* Annex B implementation */
    XDAS_TRUE,                  /* Voice Activity Detector turned on */
};
```

Description

The G729ENC module is used in applications such as videoconferencing, multimedia, and voice-email to encode speech according to the ITU G.729 recommendation. The interface also supports the reduced complexity version described in Annex A, and the silence compression scheme described in Annex B.

The input to the G729ENC is a frame of 80 samples of 16-bit little endian linear PCM data . The output is an ITU G.729 specified frame of 10 8-bit code words.

Annex A describes a reduced complexity version of the G.729 standard. They are, however, fully interoperable, so an annex A encoded packet can be decoded by a standard decoder, and vice versa.

Comments**Creation Parameters**

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>annexA</code>	<code>XDAS_TRUE</code> – Annex A (reduced complexity) implementation. <code>XDAS_FALSE</code> – Standard implementation.
<code>annexB</code>	<code>XDAS_TRUE</code> – Annex B (silence compression scheme) implementation. <code>XDAS_FALSE</code> – Standard implementation.
<code>vadEnable</code>	<code>if (annexB == XDAS_TRUE)</code> <code>XDAS_TRUE</code> – Turn on voice activity detector. <code>XDAS_FALSE</code> – Turn off voice activity detector. <code>if (annexB == XDAS_FALSE)</code> Value is ignored.

NOTE! `if (annexA == XDAS_TRUE)` and the implementation does not support Annex A, or `if (annexB == XDAS_TRUE)` and the implementation does not support Annex B, the `algInit()` function of the algorithm should fail.

Default Creation Parameters

The default creation parameters specify an annex A and annex B implementation with the voice activity detector turned on.

Status Parameters

<code>size</code>	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
<code>annexA</code>	<pre>if (IG729_Cmd == IG729_GETSTATUS) XDAS_TRUE – Annex A implementation XDAS_FALSE – Standard implementation. if (IG729_Cmd == IG729_SETSTATUS) This is a read-only parameter; value is ignored.</pre>
<code>annexB</code>	<pre>if (IG729_Cmd == IG729_GETSTATUS) XDAS_TRUE – Annex B implementation XDAS_FALSE – Standard implementation. if (IG729_Cmd == IG729_SETSTATUS) This is a read-only parameter, value is ignored.</pre>
<code>vadEnable</code>	<pre>if (IG729_Cmd == IG729_GETSTATUS) if (annexB == XDAS_TRUE) XDAS_TRUE – Voice activity detector is turned on. XDAS_FALSE – Voice activity detector is turned off. if (annexB == XDAS_FALSE) No VAD support. Value is ignored. if (IG729_Cmd == IG729_SETSTATUS) if (annexB == XDAS_TRUE) XDAS_TRUE – Turn on voice activity detector. XDAS_FALSE – Turn off voice activity detector. if (annexB == XDAS_FALSE) No VAD support. Value is ignored.</pre>

Name	control - Runtime control and status function
Syntax	<code>val = handle->fxns->control(handle, cmd, status);</code>
Parameters	<pre>IG729ENC_Handle handle; /* IG729ENC object handle */ IG729_Cmd cmd; /* control command */ IG729ENC_Status status; /* Pointer to status structure */</pre>
Return Value	<code>XDAS_BOOL val; /* XDAS_TRUE if call was successful */</code>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid G729ENC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>status</code> is pointing to a valid <code>IG729ENC_Status</code> structure. <input type="checkbox"/> <code>cmd</code> is a valid <code>IG729_Cmd</code>.
Postconditions	<pre>if (cmd == IG729_GETSTATUS) if (val == XDAS_TRUE) The status structure pointed to by <code>status</code> was successfully updated and reflects the instance's current state. if (val == XDAS_FALSE) The update of the status structure pointed to by <code>status</code>, for some reason, failed. if (cmd == IG729_SETSTATUS) if (val == XDAS_TRUE) The write parameters in the status structure pointed to by <code>status</code>, was successfully copied into the instance object. if (val == XDAS_FALSE) The update of the instance status write parameters, for some reason, failed.</pre>
Comments	None
See Also	None

Name**encode - Encoder function****Syntax**

```
numBytes = handle->fxns->encode(handle, in, out);
```

Parameters

```
IG729ENC_Handle  handle; /* IG729ENC object handle */
XDAS_Int8        *in;    /* Pointer to input buffer */
XDAS_Int8        *out;   /* Pointer to output buffer */
```

Return Value

```
XDAS_Int16      numBytes; /* returns number of bytes in en
                           coded buffer */
```

Preconditions

- `handle` is pointing to a valid G729ENC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `in` is pointing to a frame of 80 16-bit linear PCM little endian samples.

Postconditions

```
if (numBytes < 0)
```

The encoding of the input samples, for some reason, failed.

```
if (numBytes >= 0)
```

The `out` buffer contains `numBytes` encoded samples.

The contents of the `in` buffer should be the same as when entering this function.

Comments

None

See Also

None

LEC*Line Echo Canceller Abstract Interface***Includes**

```
#include <xdas.h>
#include <ialg.h>
#include <ilec.h>
```

Interface***Types And Constants***

```
/*
 * ===== ILEC_Obj =====
 * This structure must be the first field of all LEC instance objects.
 */
```

```
typedef struct ILEC_Obj {
    struct ILEC_Fxns *fxns;
} ILEC_Obj;
```

```
/*
 * ===== ILEC_Handle =====
 * This handle is used to reference all LEC instance objects.
 */
```

```
typedef struct ILEC_Obj *ILEC_Handle;
```

```
/*
 * ===== ILEC_Cmd =====
 * Control commands for a LEC instance object.
 */
```

```
typedef enum ILEC_Cmd {
    ILEC_GETSTATUS,
    ILEC_SETSTATUS
} ILEC_Cmd;
```

Creation Parameters

```
/*
 * ===== ILEC_Params =====
 * This structure defines the creation parameters for a LEC instance object.
 */
```

```
typedef struct ILEC_Params {
```



```
    Int          size;          /* Size of this structure */
    XDAS_Bool    adaptEnable;   /* Enable/Disable adaptation */
    XDAS_Bool    dTalkEnable;   /* Enable/Disable double talk detector */
    XDAS_UInt16  frameLen;      /* Frame length in number of samples */
    XDAS_Bool    nonLPEnable;   /* Enable/Disable non-linear processor */
    XDAS_UInt16  bulkDelay;     /* Delay in hybrid and system I/O */
    XDAS_UInt16  tailLen;       /* Tail length in number of samples */
} ILEC_Params;
```

Status Parameters

```
/*
 * ===== ILEC_Status =====
 * This structure defines the parameters that can be changed at runtime
 * (read/write), and the instance status parameters (read-only).
 */
typedef struct ILEC_Status {
    Int          size;          /* Size of this structure */
    XDAS_Bool    adaptEnable;   /* Adaptation on/off (Read/Write) */
    XDAS_Bool    dTalkEnable;   /* Double talk detector on/off (Read/Write) */
    XDAS_Bool    dTalkPresent; /* Double talk currently present (Read-Only) */
    XDAS_Void    *filterCoeffs; /* Pointer to filter coeffs (Read/Write) */
    XDAS_Bool    nonLPEnable;   /* Non-linear processor on/off (Read/write) */
} ILEC_Status;
```

Functions

```

/*
 * ===== ILEC_Fxns =====
 * This structure defines all of the operations on LEC objects.
 */
typedef struct ILEC_Fxns {
    IALG_Fxns  ialg;      /* ILEC extends IALG */
    XDAS_Bool (*control)(ILEC_Handle handle, ILEC_Cmd cmd,
                        ILEC_Status *status);
    XDAS_Int16 (*echoCancel)(ILEC_Handle handle, XDAS_Int16 *nearEndIn,
                            XDAS_Int16 *NearEndOut);
    XDAS_Int16 (*feedData)(ILEC_Handle handle, XDAS_Int16 *FarEndIn);
} ILEC_Fxns;

```

Default Creation Parameters

```

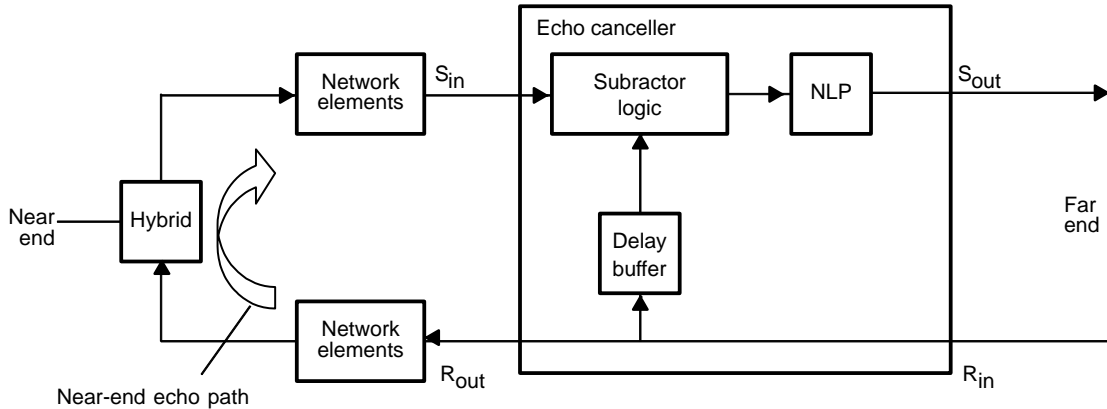
/*
 * ===== ILEC_PARAMS =====
 * This static initialization defines the default parameters used to
 * create an instances of a LEC object.
 */
const ILEC_Params ILEC_PARAMS = {
    sizeof(ILEC_PARAMS), /* Size of this structure */
    XDAS_TRUE,          /* Adaptation turned on */
    XDAS_TRUE,          /* Double talk detector turned on */
    40,                 /* 40 samples per frame, 5msec of data */
    XDAS_TRUE,          /* Non-linear processor turned on */
    240,                /* Bulk delay of 240 samples, 30msec of data */
    256,                /* Tail length (echo spread) 256 samples, 32msec */
};

```

Description

The LEC module is used in the 4-wire portion of a circuit to reduce the near-end echo present on the send path, by subtracting an estimation of that echo from the near-end echo. See Figure A-9.

Figure A-9. Line Echo Canceller



R_{in} (Receive input signal) – The signal arriving from the far end.

R_{out} (Receive output signal) – The signal transmitted to the near end.

S_{in} (Send input signal) – The signal arriving from the near end.

S_{out} (Send output signal) – The signal transmitted to the far end.

NLP – Non-linear processor.

Network Elements – I/O mechanisms delaying the signal, e.g. double buffering.

Comments**Creation Parameters**

size	Size of the creation parameter structure. If a vendor extends the creation parameter structure, size should reflect the size of the extended structure.
adaptEnable	XDAS_TRUE – Turn on filter adaptation. XDAS_FALSE – Turn off filter adaptation.
dTalkEnable	XDAS_TRUE – Annex B (silence compression scheme) implementation. XDAS_FALSE – Standard implementation.

vadEnable	<pre>if (adaptEnable == XDAS_TRUE)</pre> <p style="margin-left: 2em;">XDAS_TRUE – Turn on double talk detector. Double-talk means that Sin and the output of the delay buffer are active at the same time (e.g., a signal is being received from the far-end and from the near-end simultaneously). Filter coefficient adaptation is halted during double talk to avoid divergence of the coefficients.</p> <p style="margin-left: 2em;">XDAS_FALSE – Turn off double talk detector. Filter coefficient adapt regardless of double-talk.</p> <pre>if (adaptEnable == XDAS_FALSE)</pre> <p style="margin-left: 2em;">XDAS_TRUE – Double talk detector is turned on but no adaptation.</p> <p style="margin-left: 2em;">XDAS_FALSE – Double talk detector is turned off and no adaptation.</p>
frameLen	Number of samples in the frame passed to <code>feedData()</code> and <code>echoCancel()</code> for processing.
nonLPEnable	<p>XDAS_TRUE – Turn on non-linear processor (NLP).</p> <p>XDAS_FALSE – Turn off non-linear processor (NLP).</p>
bulkDelay	The delay in number of samples from Rout port to Sin port due to the delays in the near-end echo path. Inherent delays in the echo transmission facilities (pure delay) and delay in the network elements cause this.
tailLen	The length of the echo spread in the number of samples. Echo spread is also sometimes referred to as the dispersed signal.

Default Creation Parameters

The default creation parameters cancel a line echo with pure delay and system I/O delay of a total of 30ms and an echo spread of 32ms. In other words, echo between 30ms and 62ms will be cancelled;

Sampling rate is 8kHz

$$\text{bulkDelay} = 0.030 * 8000 = 240$$

$$\text{tailLen} = 0.032 * 8000 = 256$$

Status Parameters

size	Size of the creation parameter structure. If a vendor extends the creation parameter structure, <code>size</code> should reflect the size of the extended structure.
adaptEnable	<pre>if (ILEC_Cmd == ILEC_GETSTATUS) XDAS_TRUE – Filter adaptation is turned on. XDAS_FALSE – Filter adaptation is turned off. if (ILEC_Cmd == ILEC_SETSTATUS) XDAS_TRUE – Turn on filter adaptation. If filter adaptation is already on, this set operation will have no effect. XDAS_FALSE – Turn off filter adaptation. If filter adaptation is already off, this set operation will have no effect.</pre>
dTalkEnable	<pre>if (ILEC_Cmd == ILEC_GETSTATUS) XDAS_TRUE – Double-talk detection is turned on. XDAS_FALSE – Double-talk detection is turned off. if (ILEC_Cmd == ILEC_SETSTATUS) XDAS_TRUE – Turn on double talk detection. If double-talk detection is already on, this set op- eration will have no effect. XDAS_FALSE – Turn off double talk detection. If double-talk detection is already off, this set op- eration will have no effect.</pre>
dTalkPresent	<pre>if (ILEC_Cmd == ILEC_GETSTATUS) if (dTalkEnable == XDAS_TRUE) XDAS_TRUE – Double-talk is currently pres- ent. XDAS_FALSE – Double-talk is currently not present. if (dTalkEnable == XDAS_FALSE) Double-talk detection is turned off. Value is ignored. if (ILEC_Cmd == ILEC_SETSTATUS) This is a read-only parameter. Value is ignored.</pre>

```
*filterCoeffs  if ( ILEC_Cmd == ILEC_GETSTATUS)
                  Return pointer to filter coefficients.
                  if ( ILEC_Cmd == ILEC_SETSTATUS)
                    Set instance object filter coefficients to pointer.
nonLPEnable     if ( ILEC_Cmd == ILEC_GETSTATUS)
                  XDAS_TRUE – Non-linear processor is turned
                  on.
                  XDAS_FALSE – Non-linear processor is turned
                  off.
                  if ( ILEC_Cmd == ILEC_SETSTATUS)
                    XDAS_TRUE – Turn on non-linear processor. If
                    the non-linear processor is already on, this set
                    operations will have no effect.
                    XDAS_FALSE – Turn off non-linear processor. If
                    the non-linear processor is already off, this set
                    operations will have no effect.
```

Name**control - Runtime control and status function****Syntax**

```
val = handle->fxns->control (handle, cmd, status);
```

Parameters

```
ILEC_Handle handle;    /* ILEC object handle */
ILEC_Cmd      cmd;     /* control command */
ILEC_Status  status;  /* Pointer to status structure */
```

Return Value

```
XDAS_BOOL      val;    /* XDAS_TRUE if call was successful */
```

Preconditions

- `handle` is pointing to a valid LEC instance object.
- This function can only be called after successfully initializing the instance object pointed to by `handle`.
- `status` is pointing to a valid `ILEC_Status` structure.
- `cmd` is a valid `ILEC_Cmd`.

Postconditions

```
if (cmd == ILEC_GETSTATUS)
```

```
if (val == XDAS_TRUE)
```

The status structure pointed to by `status` was successfully updated and reflects the instance current state.

```
if (val == XDAS_FALSE)
```

The update of the status structure pointed to by `status`, for some reason, failed.

```
if (cmd == ILEC_SETSTATUS)
```

```
if (val == XDAS_TRUE)
```

The write parameters in the status structure pointed to by `status` were successfully copied into the instance object.

```
if (val == XDAS_FALSE)
```

The update of the instance status write parameters, for some reason, failed.

Comments

None

See Also

None

Name	echoCancel - Function to cancel near-end echo
Syntax	<pre>numSamples = handle->fxns->echoCancel(handle, nearEndIn, nearEndOut);</pre>
Parameters	<pre>ILEC_Handle handle; /* ILEC object handle */ XDAS_Int16 *nearEndIn; /* Ptr to near-end in buffer */ XDAS_Int16 *nearEndOut; /* Ptr to near-end out buffer */</pre>
Return Value	<pre>XDAS_Int16 numSamples; /* number of samples in nearEndOut buffer */</pre>
Preconditions	<ul style="list-style-type: none"> <input type="checkbox"/> <code>handle</code> is pointing to a valid LEC instance object. <input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>. <input type="checkbox"/> <code>nearEndIn</code> is pointing to <code>frameLen</code> 16-bit linear PCM little endian samples.
Postconditions	<pre>if (numSamples < 0)</pre> <p>If, for some reason, the algorithm detected that the processing resulted in incorrect results.</p> <pre>if (numSamples >= 0)</pre> <p><code>nearEndOut</code> buffer contains <code>numSamples</code> 16-bit linear little endian PCM samples.</p> <p>The contents of the <code>nearEndIn</code> buffer do not need to be the same as when entering this function.</p>
Comments	<p>The function should work properly for all possible configurations of <code>tailLen</code>, <code>bulkDelay</code> and <code>frameLen</code>.</p> <p>The consumer of the LEC instance object must ensure that <code>echoCancel()</code> run-to-completion before calling <code>feedData()</code>.</p> <p>If the algorithm does the processing in-place, it should set <code>nearEndOut = nearEndIn</code> before returning.</p>
See Also	<code>feedData()</code>

Name	feedData - Function to copy far-end input data into the instance object's delay buffer
Syntax	<pre>numSamples = handle->fxns->feedData(handle, farEndIn);</pre>
Parameters	<pre>ILEC_Handle handle; /* ILEC object handle */ XDAS_UInt16 *farEndIn; /* Pointer to far-end input buffer */</pre>
Return Value	<pre>XDAS_Int16 numSamples; /* number of samples in farEndIn */</pre>
Preconditions	<ul style="list-style-type: none"><input type="checkbox"/> <code>handle</code> is pointing to a valid LEC instance object.<input type="checkbox"/> This function can only be called after successfully initializing the instance object pointed to by <code>handle</code>.<input type="checkbox"/> <code>nearEndIn</code> is pointing to a 16-bit linear PCM little endian <code>frameLen</code> number of samples.
Postconditions	<pre>if (numSamples < 0)</pre> <p>If, for some reason, the algorithm detected that copying the data failed.</p> <pre>if (numSamples >= 0)</pre> <p><code>numSamples</code> number of 16 bit linear PCM little endian samples were successfully copied from R_{in} into the instance object's delay buffer.</p> <p>The contents of the <code>farEndIn</code> buffer should be the same as when entering this function.</p>
Comments	<p>The function should work properly for all possible configurations of <code>tailLen</code>, <code>bulkDelay</code> and <code>frameLen</code>.</p> <p>The consumer of the LEC instance object must ensure that <code>feedData()</code> run-to-completion before calling <code>echoCancel()</code>.</p>
See Also	<pre>echoCancel()</pre>