dsPIC® DSC SPEECH CODING SOLUTIONS USER'S GUIDE

Note the following details of the code protection feature on Microchip devices:

• Microchip products meet the specification contained in their particular Microchip Data Sheet.

• Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.

• There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.

• Microchip is willing to work with the customer who is concerned about the integrity of their code.

• Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
══ ISO/TS 16949:2002 ══

dsPIC® DSC SPEECH CODING SOLUTIONS USER'S GUIDE

Table of Contents

dsPIC® DSC Speech Coding Solutions User's Guide

dsPIC® DSC SPEECH CODING SOLUTIONS USER'S GUIDE

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE on-line help. Select the Help menu, and then Topics to open a list of available on-line help files.

INTRODUCTION

This chapter contains general information that is useful to know before you begin using the dsPIC® DSC Speech Encoding/Decoding Libraries. Items discussed include:

- Document Layout
- Conventions Used in this Guide
- Warranty Registration
- Recommended Reading
- The Microchip Web Site
- Development Systems Customer Change Notification Service
- Customer Support
- Document Revision History

DOCUMENT LAYOUT

This document describes how to use the dsPIC DSC Speech Encoding/Decoding Libraries as a development tool to emulate and debug firmware on a target board. The manual layout is as follows:

- **Chapter 1. Overview** – This chapter provides an overview of the dsPIC DSC Speech Encoding/Decoding Libraries and identifies the salient features of each library.
- **Chapter 2. Installation** – This chapter provides detailed instructions for installing the dsPIC DSC Speech Encoding/Decoding Libraries on your PC and setting them up to run with the MPLAB® Integrated Development Environment (IDE).
- **Chapter 3. Application Programming Interface** – This chapter provides information you need to interface the dsPIC DSC Speech Encoding/Decoding Libraries with your user application.

- **Chapter 4. Integrating Speech Encoding Into YourApplication** – This chapter provides information to help you understand how to integrate the speech encoding portion of the dsPIC DSC Speech Encoding/Decoding Libraries into your application and how to build with the library.
- **Chapter 5. Integrating Speech Decoding Into YourApplication** – This chapter provides information to help you understand how to integrate the speech decoding portion of the dsPIC DSC Speech Encoding/Decoding Libraries into your application and how to build with the library.
- **Chapter 6. Speech Encoding Utility** – This chapter describes the Speech Encoding Utility provided with the dsPIC DSC Speech Encoding/Decoding Libraries and provides instructions for creating speech files.
- **Chapter 7. Using Flash Memory for Speech Playback –** This chapter provides information on the use of external Flash memory with the library.
- **Chapter 8. Speech Coding Demos** – This chapter describes a sample application that demonstrates stand-alone speech encoding and playback from on-chip data EEPROM memory.
- **Appendix A. Si3000 Codec Configuration** – This appendix provides configuration details for the Si3000 codec interface.
- **Appendix B. External Flash Memory Reference Design** – This appendix provides circuit schematics for an interface to external 16-bit nonvolatile memory.

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
'b*nnnn*	A binary number where *n* is a digit	'b00100, 'b10
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier font:		
Plain Courier	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
Italic Courier	A variable argument	`file`.o, where `file` can be any valid filename
0x*nnnn*	A hexadecimal number where *n* is a hexadecimal digit	`0xFFFF, 0x007A`
Square brackets [ ]	Optional arguments	`mcc18 [options] file [options]`
Curly brackets and pipe character: {	}	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`

	Represents code supplied by user	`void main (void)`
		`{...`
		`}`

dsPIC® DSC Speech Coding Solutions User's Guide

WARRANTY REGISTRATION

Please complete the enclosed Warranty Registration Card and mail it promptly. Sending in the Warranty Registration Card entitles users to receive new product updates. Interim software releases are available on the Microchip web site.

RECOMMENDED READING

This user's guide describes how to use the G.711, G.726A and Speex Speech Encoding/Decoding libraries. The following Microchip documents are available and recommended as additional reference resources.

dsPIC30F Family Reference Manual (DS70046)

Refer this document for detailed information on dsPIC30F device operation. This reference manual explains the operation of the dsPIC30F Digital Signal Controller (DSC) family architecture and peripheral modules, but does not cover the specifics of each device. Refer to the appropriate device data sheet for device-specific information.

16-bit MCU and DSC Programmer's Reference Manual (DS70157)

This manual is a software developer's reference for dsPIC30F and dsPIC33F 16-bit DSC devices. This manual describes the instruction set in detail and also provides general information to assist you in developing software for the dsPIC30F/33F DSC family.

dsPIC33F/PIC24H Family Reference Manual Sections

Consult these documents for detailed information on dsPIC33F/PIC24H device operation. These reference manual sections explain the operation of the dsPIC33F/PIC24H DSC and MCU family architecture and peripheral modules, but do not cover the specifics of each device. Refer to the appropriate device data sheet for device-specific information.

MPLAB® Assembler, Linker and Utilities for PIC24 MCUs and dsPIC® DSCs User's Guide (DS51317)

This document helps you use Microchip Technology's language tools for dsPIC DSC devices based on GNU technology. The language tools discussed are:

- MPLAB Assembler PIC24 MCUs and dsPIC® DSCs
- MPLAB Linker PIC24 MCUs and dsPIC® DSCs
- MPLAB Archiver/Librarian PIC24 MCUs and dsPIC® DSCs
- Other Utilities

MPLAB® C Compiler for PIC24 MCUs and dsPIC® DSCs User's Guide (DS51284)

This document helps you use Microchip's MPLAB C compiler for dsPIC DSC devices to develop your application. The MPLAB C Compiler is a GNU-based language tool, based on source code from the Free Software Foundation (FSF). For detailed information about FSF, see www.fsf.org.

MPLAB® IDE, Simulator, Editor User's Guide (DS51025)

Refer this document for detailed information pertaining to the installation and implementation of the MPLAB Integrated Development Environment (IDE) Software.

To obtain any of these documents, contact the nearest Microchip sales location (see back page) or visit the Microchip web site at: www.microchip.com.

> **Note:** The latest versions of the following manufacturers' data sheets are also recommended as reference sources: **Si3000 Voiceband Codec with Microphone/Speaker Drive** (Silicon Laboratories Publication Si3000-DS11) **Am29F200B 2 Megabit (256 K x 8-Bit/128 K x 16-Bit) CMOS 5.0 Volt-only, Boot Sector Flash Memory** (AMD Publication 21526)

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at www.microchip.com. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups and Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers and other language tools. These include the MPLAB C18 and MPLAB C30 C compilers; MPASM™ and MPLAB ASM30 assemblers; MPLINK™ and MPLAB LINK30 object linkers; and MPLIB™ and MPLAB LIB30 object librarians.
- **Emulators** – The latest information on Microchip in-circuit emulators.This includes the MPLAB ICE 2000 and MPLAB ICE 4000.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debugger, MPLAB ICD 2.
- **MPLAB® IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB SIM simulator, MPLAB IDE Project Manager and general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include the MPLAB PM3 and PRO MATE® II device programmers and the PICSTART® Plus and PICkit™ 1 development programmers.

dsPIC® DSC Speech Coding Solutions User's Guide

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or FAE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through our web site at: http://support.microchip.com

DOCUMENT REVISION HISTORY

Revision A (September 2007)

- Initial release of this document.

Revision B (March 2010)

This revision incorporates the following updates:

- Note:
  - Added the following note in **Section 3.4.3 "G726_decode()Function"**: The 256 decoded speech samples returned in the `codecdata.sampleDecodeIpBuffer` pointer must be left-shifted by two bits to retain the correct sign of data.
  - Added the following note in **Section 3.4.5 "G726_encode()Function"**: The 256 raw speech samples must be right-shifted by two bits before assigning it to the `codecdata.sampleOpBuffer` pointer for encoding.
- Additional minor corrections such as language and formatting updates are incorporated throughout the document.

Chapter 1. Overview

The dsPIC DSC Speech Encoding/Decoding Libraries include G.711, G.726A and Speex Speech Encoding/Decoding software application solutions. These individual libraries provide toll-quality voice compression and decompression to help you generate speech-based embedded applications on the dsPIC30F and dsPIC33F families of digital signal controllers. This chapter provides an overview and feature listing of these three libraries. Topics covered include:

• Overview
• Other Features

1.1   OVERVIEW

The three speech coding techniques described in this document provide different sets of capabilities and consume different levels of computational resources. In each case, the objectives are to reduce the amount of data required to represent a speech signal while not compromising on the quality of speech when it is decoded.

In communication applications, the advantage of speech compression is to reduce the consumption of communication bandwidth, while for many other applications the advantage is to reduce the amount of memory required to store the recorded speech.

A comparison of the computational resource requirements used by the three algorithms is provided in **Section 3.2.2 "MIPS and Memory Requirements"**.

1.1.1   G.711 Speech Encoding/Decoding Library

The G.711 Library is an ITU-T standard speech coding method that utilizes A-law and µ-law compression/expansion (also known as companding). This technique provides a reduction in data (compression ratio) of 2:1, and the best decoded speech quality of the three techniques. For an input sampling rate of 8 kHz, the output bit rate obtained is 64 kbps. Compressed playback files require approximately 8 Kbytes of memory for each second of speech.

1.1.2   G.726A Speech Encoding/Decoding Library

The G.726A Library is another ITU-T standard speech coder. This library uses the Adaptive Differential Pulse Code Modulation (ADPCM) methodology. Table 1-1 lists the output bit rates provided for the corresponding compression ratios.

TABLE 1-1:   G.726A LIBRARY BIT RATES AND COMPRESSION RATIOS

Bit Rate	Compression Ratio
40 kbps	3.2:1
32 kbps	4:1
24 kbps	5.33:1
16 kbps	8:1

The 40 kbps and 32 kbps modes of G.726A provide a decoded speech quality similar to that of G.711.

Compressed playback files require 2 to 5 Kbytes of memory for each second of speech.

1.1.3 Speex Speech Encoding/Decoding Library

The Speex Library is based on the open-source Speex speech coder. The library samples speech at 8 kHz and compresses it to a rate of 8 kbps, resulting in a 16:1 compression ratio. The Speex encoding algorithm uses Code Excited Linear Prediction (CELP), which provides a reasonable trade-off between performance and computational complexity.

Compressed playback files require approximately 1 Kbyte of memory for each second of speech.

1.2   OTHER FEATURES

Regardless of the speech encoding/decoding algorithm used, these speech files can be stored on-chip, in program memory or data EEPROM, or externally in Flash memory, as shown in Figure 1-1.

FIGURE 1-1:   TYPICAL SPEECH ENCODING/DECODING APPLICATION
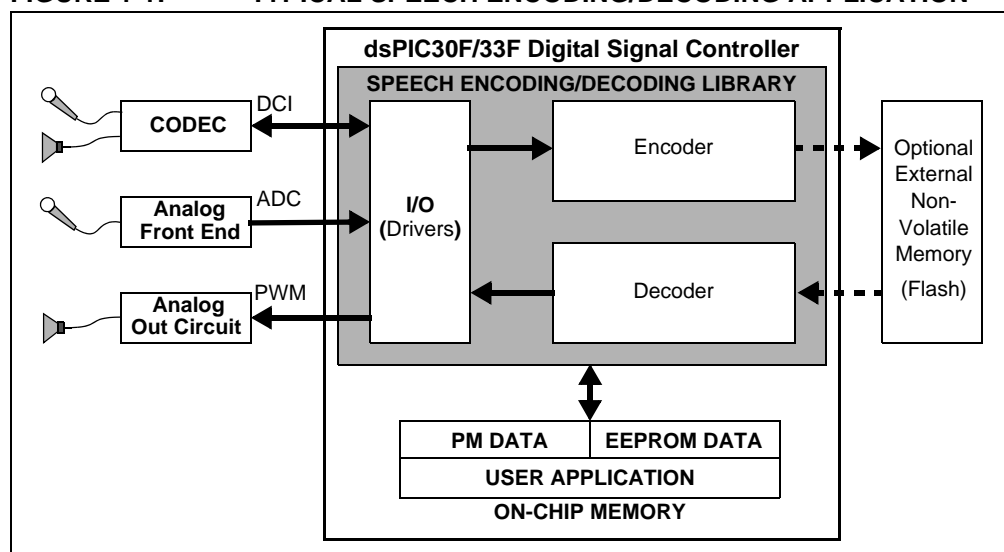
The flexible analog interface offers several design options. The speech encoder can sample input from either an external codec or the on-chip 12-bit Analog-to-Digital Converter (ADC). The speech decoder can play decoded speech through an external codec or the on-chip pulse-width modulator. With the Speex library, an optional Voice Activity Detection feature enhances compression by detecting voids in the incoming speech and compressing them at a higher ratio. All three libraries optimize computational performance and RAM usage. Well-defined APIs (see **Chapter 3. "Application Programming Interface"**) make it easy to integrate with your application.

Playback-only applications can benefit from the PC-based speech encoding utility (see **Chapter 6. "Speech Encoding Utility"**), which lets you encode speech files from your desktop using a microphone or the existing WAVE (.wav) files. Encoded speech files are built into your application through your MPLAB IDE project, like any source file. The speech encoding utility lets you to select four target memory areas for your speech file:

• Program memory
• RAM
• Data EEPROM (dsPIC30F only)
• External Flash memory (dsPIC30F and Speex only)

External Flash memory allows you to store several minutes of speech (one minute of speech requires 60 KB), and it is supported through a dsPIC general purpose I/O port.

Chapter 2. Installation

The dsPIC DSC Speech Encoding/Decoding Libraries do not execute directly from the CD. You must install them onto your laptop or desktop PC. This chapter includes the following installation information:

- Installation Procedure
- G.711 Library Files
- G.726A Library Files
- Speex Library Files

2.1 INSTALLATION PROCEDURE

Each of the libraries is packaged on a CD. To install the library, follow these steps:

1. Insert the library CD into the appropriate drive. The installation screen appears.
2. Select the **Click to Install Files** option. The installation location dialog appears to let you choose a directory for the library.
3. Browse to the directory of your choice, and then click **OK**. The License Agreement appears.
4. Review the license agreement and click **OK** to continue. The next dialog displays the installation progress, followed by the Installation Complete dialog.

The installation process creates a folder named `G.711 v1.0`, `G.726A v1.0` or `Speex v2.0` (depending on the specific library installed) in the user-specified root directory.

2.2 G.711 LIBRARY FILES

The G.711 Library creates a directory labeled `G711 v1.0`. This directory contains three folders with their corresponding sub-folders and files:

- `G711_dsPIC30F`
- `G711_dsPIC33F`
- `G711_PC`

2.2.1 G711_dsPIC30F

This folder contains all library archive, include and demo application files to support the G711 library on the dsPIC30F device family. This folder contains three sub-folders:

- `demo`
- `inc`
- `src`

2.2.1.1 DEMO

The `demo` sub-folder contains three additional sub-folders:

- `Communication`
- `Loopback`
- `Playback`

These sub-folders include all source, include, project and workspace files required for the Communication, Loopback and Playback demo applications (See **Chapter 8. "Speech Coding Demos"** for more details).

2.2.1.2   INC

The `inc` sub-folder contains all include files required by the library and by the applications integrating the library. The following files are provided:

- `g711.h`
- `G711Lib_common.h`
- `G711Lib_common.inc`
- `G711Lib_internal.h`
- `G711Lib_Si3000.h`

2.2.1.3   SRC

The `src` folder contains the source files for encoding and decoding speech using the G.711 algorithm. Any application integrating this library must include the following source files.

Two source files provided are:

- `g711_decoder.c`
- `g711_encoder.c`

2.2.2   G711_dsPIC33F

This folder contains all library archive, include and demo application files to support the G711 library on the dsPIC33F device family. This folder contains three sub-folders:

- `demo`
- `inc`
- `src`

2.2.2.1   DEMO

The `demo` sub-folder contains three additional sub-folders:

- `Communication`
- `Loopback`
- `Playback`

These sub-folders include all source, include, project and workspace files required for the Communication, Loopback and Playback demo applications (See **Chapter 8. "Speech Coding Demos"** for more details).

2.2.2.2   INC

The `inc` sub-folder contains all include files required by the library and by the applications integrating the library. The following files are provided:

- `g711.h`
- `G711Lib_common.h`
- `G711Lib_common.inc`
- `G711Lib_internal.h`
- `G711Lib_Si3000.h`

2.2.2.3   SRC

The `src` folder contains the source files for encoding and decoding speech using the G.711 algorithm. Any application integrating this library must include the following two source files:

- `g711_decoder.c`
- `g711_encoder.c`

2.2.3   G711_PC

This folder contains:

- Speech Encoding Utility (`dsPICSpeechRecord.exe`)
- DLL files for the Speech Encoding Utility (`SpeechRecord_G711.dll` and `SpeechRecord_G726.dll`)
- PC command-line based utility to decode speech using the G.711 decoder (`AWG711Decoder.exe`)

2.3   G.726A LIBRARY FILES

The G.726A Library creates a directory labeled `G726A v1.0`. This directory contains three folders with their corresponding sub-folders and files:

- `G726A_dsPIC30F`
- `G726A_dsPIC33F`
- `G726A_PC`

2.3.1   G726A_dsPIC30F

This folder contains all library archive, include and demo application files to support the G726A library on the dsPIC30F device family. This folder contains the following three sub-folders:

- `demo`
- `inc`
- `lib`

2.3.1.1   DEMO

The `demo` sub-folder contains three additional sub-folders:

- `Communication`
- `Loopback`
- `Playback`

These sub-folders include all source, include, project and workspace files required for the Communication, Loopback and Playback demo applications (See **Chapter 8. "Speech Coding Demos"** for more details).

2.3.1.2   INC

The `inc` sub-folder contains all include files required by the library and by the applications integrating the library. The following files are provided:

- `g726a.h`
- `G726ALib_common.h`
- `G726ALib_common.inc`
- `G726ALib_internal.h`
- `G726ALib_Si3000.h`

2.3.1.3   LIB

The `lib` folder contains a pre-compiled library archive file for encoding and decoding speech using the G.726A algorithm. Any application integrating this library must include this library archive file.

A single library archive is provided: `libg726a.a`

2.3.2   G726A_dsPIC33F

This folder contains all library archive, include and demo application files to support the dsPIC33F device family. The folder structure and contents are similar to the `G726A_dsPIC30F` folder.

This folder contains all library archive, include and demo application files to support the G726A library on the dsPIC33F device family. This folder contains the following three sub-folders:

• `demo`
• `inc`
• `lib`

2.3.2.1   DEMO

The `demo` sub-folder contains three additional sub-folders:

• `Communication`
• `Loopback`
• `Playback`

These sub-folders include all the source, include, project and workspace files required for the Communication, Loopback and Playback demo applications (See **Chapter 8. "Speech Coding Demos"** for more details).

2.3.2.2   INC

The `inc` sub-folder contains all include files required by the library and the applications integrating the library. The following files are provided:

• `g726a.h`
• `G726ALib_common.h`
• `G726ALib_common.inc`
• `G726ALib_internal.h`
• `G726ALib_Si3000.h`

2.3.2.3   LIB

The `lib` folder contains a pre-compiled library archive file for encoding and decoding speech using the G.726A algorithm. Any application integrating this library must include this library archive file.

A single library archive is provided: `libg726a.a`

2.3.3   G726A_PC

This folder contains:

• Speech Encoding Utility (`dsPICSpeechRecord.exe`)
• DLL files for the Speech Encoding Utility (`SpeechRecord_G711.dll` and `SpeechRecord_G726.dll`)
• PC command-line based utility to decode speech using the G.726A decoder (`AWG726ADecoder.exe`)

2.4   SPEEX LIBRARY FILES

The Speex Library creates a directory labeled `Speex v2.0`. This directory contains three folders with their corresponding sub-folders and files:

- `Speex_dsPIC30F`
- `Speex_dsPIC33F`
- `Speex_PC`

2.4.1   Speex_dsPIC30F

This folder contains all library archive, include and demo application files to support the dsPIC30F device family. This folder contains the following three sub-folders:

- `demo`
- `inc`
- `lib`

2.4.1.1   DEMO

The `demo` folder contains two additional sub-folders:

- `Communication`
- `Playback`

These sub-folders include all source, include, project and workspace files required for the Communication and Playback demo applications (See **Chapter 8. "Speech Coding Demos"** for more details).

2.4.1.2   INC

The `inc` folder contains all include files required by the library and the applications integrating the library. The following files are provided:

- `spxlib_common.h`
- `spxlib_common.inc`
- `spxlib_internal.h`
- `spxlib_Si3000.h`

2.4.1.3   LIB

The `lib` folder contains a pre-compiled library archive file for encoding and decoding speech using the Speex algorithm. Any application integrating this library must include this library archive file.

A single library archive is provided: `libSpeex.a`

2.4.2   Speex_dsPIC33F

This folder contains all library archive, include and demo application files to support the dsPIC33F device family. The folder structure and contents are similar to the `Speex_dsPIC30F` folder, except that the RecordPlay demo is not included.

This folder contains the following three sub-folders:

- `demo`
- `inc`
- `lib`

2.4.2.1   DEMO

The `demo` folder contains two additional sub-folders:

• `Communication`
• `Playback`

These sub-folders include all source, include, project and workspace files required for the Communication and Playback demo applications (See **Chapter 8. "Speech Coding Demos"** for more details).

2.4.2.2   INC

The `inc` folder contains all include files required by the library and the applications integrating the library. The following files are provided:

• `spxlib_common.h`
• `spxlib_common.inc`
• `spxlib_internal.h`
• `spxlib_Si3000.h`

2.4.2.3   LIB

The `lib` folder contains a pre-compiled library archive file for encoding and decoding speech using the Speex algorithm. Any application integrating this library must include this library archive file.

A single library archive is provided: `libSpeex.a`

2.4.3   Speex_PC

This folder contains three sub-folders containing various PC-based utilities:

• `ExternalFlashHexmaker`
• `ExternalFlashProgrammer`
• `PCEU`

2.4.3.1   ExternalFlashHexMaker

This sub-folder is an MPLAB IDE workspace provided to enable users to generate a hex file containing pre-encoded speech data that can be programmed into an external Flash memory device.

2.4.3.2   ExternalFlashProgrammer

This sub-folder is a dsPIC30F based program provided to enable users to download pre-encoded speech data through an RS-232 interface and program the data into an AMD29F200B external Flash memory device.

2.4.3.3   PCEU

This sub-folder contains:

• Speech Encoding Utility (`dsPICSpeechRecord.exe`)
• DLL file for the Speech Coding Utility (`SpeechRecord.dll`)
• PC command-line based utility to decode speech using the Speex decoder (`AWSpeexDec.exe`)

Chapter 3. Application Programming Interface

This chapter provides information needed to interface each of the dsPIC DSC Speech Encoding/Decoding Libraries with your user-assigned application. Topics covered include:

- Application Programming Interface
- System Requirements
- G.711 API
- G.726A API
- Speex API

3.1 APPLICATION PROGRAMMING INTERFACE

All three of the speech encoding/decoding libraries described in this document integrate with a user-assigned application running on the dsPIC30F or dsPIC33F device to provide support for handling speech in the application. The Application Programming Interface (API) functions are similar for all three libraries. Table 3-1 summarizes the API for each library.

TABLE 3-1: **dsPIC® DSC SPEECH ENCODING/DECODING LIBRARIES API**

Library	Source Files	Implementation
G.711	`g711_encoder.c` `g711_decoder.c`	The appropriate source file must be included in the application, depending on whether encoding (compression) or decoding (expansion) or both encoding and decoding (companding) is required.
G.726A	`libG726A.a`	This library archive contains functions for encoding raw speech, for decoding encoded speech and for encoder/decoder initialization. All functions in the library adhere to the Microchip C30 compiler function calling convention.
Speex	`libSpeex.a`	This archive contains both encoding and decoding functions.

3.2 SYSTEM REQUIREMENTS

3.2.1 Device Frequency Requirements

All three of the speech coding libraries require that speech be sampled and played back at a fixed rate of 8.0 kHz. Speech sampling is typically performed by an external audio codec that can interface with the dsPIC30F/dsPIC33F via its Data Converter Interface (DCI) module.

When sampling is performed with the DCI as the codec clock master (as in the demos included with this library), your application can use only a limited number of system frequencies to accommodate an 8.0 kHz sampling rate. In this mode, the dsPIC processor can execute only at multiples of 4.096 MHz. Therefore, the allowable execution speeds for applications using any of these libraries are 8.192 MHz, 12.288 MHz, 16.384 MHz and 24.576 MHz when the dsPIC30F/dsPIC33F is the codec clock master.

To accommodate these system frequencies for DCI master mode, operate the dsPIC using only the clock speeds shown in Table 3-2.

TABLE 3-2:   ALLOWED CLOCK SPEEDS IN DCI MASTER MODE

Processor Frequency[1]	Clock Frequency
8.192 MIPS	4.096 MHz
12.288 MIPS	6.144 MHz
16.384 MIPS	4.096 MHz
24.576 MIPS	6.144 MHz

Note 1: The decoder can run at these frequencies, but the encoder requires at least 19 MIPS.

To overcome the limitations that the processor frequency imposes on the sampling rate, the DCI can be configured for slave operation. In this case, the DCI and Si3000 use an external clock. The dsPIC DSC Speech Encoding/Decoding Libraries allow you to configure the DCI as a slave or master by providing #define statements in the spxlib_si3000.h file, as shown below:

```
#define DCIMODE   1
```

To configure the DCI as a slave, change the value to '0'. For the Si3000 codec register settings, the #define statement for each register is provided in the spxlib_si3000.h file separately for master and slave operations of the DCI.

When operating with any alternate sampling/playback interfaces, such as the on-chip 12-bit ADC and PWM (with some external analog signal conditioning), there are no restrictions on the system clock frequency provided the MIPS requirements of the algorithms are met.

3.2.2   MIPS and Memory Requirements

Memory requirements for the G.711, G.726A and Speex libraries (operating in a full-duplex configuration) are shown in Table 3-3.

TABLE 3-3:   MIPS, FLASH AND RAM REQUIREMENTS

Parameter	Library		
	G.711	G.726A	Speex
Device Speed	1 MIPS	13 MIPS	20 MIPS
Flash Memory Required	3.5 KB	6 KB	30 KB
RAM Required	3.5 KB	4 KB	7 KB
Memory needed to store 1 second of encoded speech	8 KB	2, 3, 4 or 5 KB	1 KB

3.2.3   Software Requirements

The dsPIC DSC Speech Encoding/Decoding Libraries require the following PC software:

• Windows 98/2000/XP
• MPLAB IDE V7.60 or higher
• MPLAB C30 Compiler V3.00 or higher

3.3 G.711 API

3.3.1 `codecsetup` Structure

The `codecsetup` structure is defined in the `G711lib_common.h` file. This structure is used to access:

- User-defined raw, encoded and decoded speech buffers (described in detail in the next two sections)
- Synchronization flags
- Speech sample counters used for encoding and decoding

A basic understanding of this structure is required for integrating the library with your application.

EXAMPLE 3-1: `codecsetup` **STRUCTURE**

```
struct _codecsetup
{
//Pointer to decoded Speech sample buffer1.
  volatile short *sampleExpandIpBuffer;

//Pointer to decoded Speech sample buffer2.
  volatile short *sampleExpandOpBuffer;

//Pointer to raw Speech sample buffer1.
  volatile short *sampleIpBuffer;

//Pointer to raw Speech sample buffer2.
  volatile short *sampleOpBuffer;

//Pointer to encoded speech sample buffer1.
  volatile char *sampleComprsIpBuffer;

//Pointer to encoded speech sample buffer2.
  volatile char *sampleComprsOpBuffer;

//Flag to indicate ping-pong buffer filled or empty.
  volatile char fBlockdone;

//Flag to start or stop speech playback.
  volatile char fStartPlay;

//Counter to keep count of number of blocks of data encoded.
  volatile int blockCount;

//Counter to keep count of number of blocks of data decoded.
  volatile int loadblockCount;

//Counter to keep track of number of samples stored.
  volatile int countFill;

//Counter to keep track of number of samples played.
  volatile int countLoad;
```

EXAMPLE 3-1: **codecsetup STRUCTURE(CONTINUED)**

```
//Counter to keep track of number of samples encoded.
  volatile unsigned long sampleCount;

//Number of samples in each frame.
  volatile char numOfSamplesPerFrame;

//Flag to indicate decoding is done.
  volatile char fBlockplayed;

//Flag to indicate compression is done.
  volatile char fCompressdone;

//Flag to indicate encoding is done.
  volatile char fEncodedone;

//Number of sets of data for the set ADC Buffer Length.
  volatile int setOfADCData;

//Pointer to ADCBUF0 register.
  volatile unsigned int* AdcBuf0Ptr;

//Number of bytes in the encoded speech.
  unsigned long arraysizeinbytes;

//Size of the recorded (encoded) speech in number of frames.
  long recordSize;

//G.711 companding method: 1 for A-law, 0 for µ-Law.
  char law;

//G.726A output bit-rate: 5 for 40 kbps, 4 for 32 kbps,
//                        3 for 24 kbps, 2 for 16 kbps.
  short rate;

//Used to set Master/Slave in communication applications.
  short initiator;
};
```

The structure codecdata of type codecsetup is defined for your use in
G711lib_common.h file.

```
  typedef struct _codecsetup codecsetup
  extern codecsetup codecdata;
```

Note: The G711lib_common.h file may be customized for each individual application, but modifying the codecdata structure is not recommended.

3.3.2   `g711si3000` **Structure**

The `g711Si3000` structure defined in the `G711lib_Si3000.h` file represents all the registers of the Si3000 Voiceband Codec. Applications that use the Si3000 codec as the sampling and/or playback interface for speech can use this structure. This structure also includes the settings for the DCI peripheral. This structure can be initialized using the `#define` statements provided in the `G711lib_Si3000.h` file.

Appendix A. "Si3000 Codec Configuration" contains detailed information about configuring the Si3000 registers.

EXAMPLE 3-2:   `g711si3000` STRUCTURE

```
struct _g711Si3000
    {
    int control1;                    //Si3000 Register 1
    int control2;                    //Si3000 Register 2
    int pLL1divideN1;                //Si3000 Register 3
    int pLL1multiplyM1;              //Si3000 Register 4
    int rxgaincontroL1;             //Si3000 Register 5
    int adcvolumecontrol;           //Si3000 Register 6
    int dacvolumecontrol;           //Si3000 Register 7
    int statusreport;               //Si3000 Register 8
    int analogattenuation;          //Si3000 Register 9
    char dcimode;                    //1=master, 0=slave
    char dciintpri;                  //DCI interrupt priority
    int bcg1;                        //bit clock generator
    };
```

Set the `#define` statement in `g711lib_Si3000.h` file for your application. Another `#define` statement is provided to create a data structure of type `g711Si3000`:

```
#define G711SI3000INIT   const g711Si3000 g711 = G711;
```

To make the Speex structure accessible to your source application, simply reference the `G711SI3000INIT` define in your source code, where you define your other data:

```
int my_variable;
G711SI3000INIT         // Si3000 data structure instantiation
                       // This defines the initialized Si3000
                       // data structure
```

3.3.3 `alaw_compress()/`µ`law_compress()` Function

This function is used to:

- Compress a block of 256 speech samples
- Generate an output block of compressed speech of 256 bytes

Return Value

None

Parameters

This function has three parameters.

Parameter	Slen
Data Type	long
Usage	Number of samples per block

Parameter	codecdata.sampleOpBuffer
Data Type	short *
Usage	Pointer to raw speech sample buffer 2

Parameter	codecdata.sampleComprsIpBuffer
Data Type	short *
Usage	Pointer to compressed speech sample buffer 1

3.3.4 `alaw_expand()/`µ`law_expand()` Function

This function is used to:

- Expand a block of 256 compressed speech samples
- Generate an output block of 256 expanded speech samples

Return Value

None

Parameters

This function has three parameters.

Parameter	Slen
Data Type	long
Usage	Number of samples per block

Parameter	codecdata.sampleComprsOpBuffer
Data Type	short *
Usage	Pointer to compressed speech sample buffer 2

Parameter	codecdata.sampleExpandIpBuffer
Data Type	short *
Usage	Pointer to expanded speech sample buffer 1

3.4   G.726A API

3.4.1   `codecsetup` Structure

The `codecsetup` structure is defined in the `G726Alib_common.h` file. This structure is used to access:

- User defined raw, encoded and decoded speech buffers (described in detail in the next two sections)
- Synchronization flags
- Speech sample counters used for encoding and decoding

A basic understanding of this structure is required for integrating the library with your application.

EXAMPLE 3-3:   `codecsetup` STRUCTURE

```
struct _codecsetup
{
//Pointer to decoded Speech sample buffer1.
  volatile short *sampleDecodeIpBuffer;

//Pointer to decoded Speech sample buffer2.
  volatile short *sampleDecodeOpBuffer;

//Pointer to raw Speech sample buffer1.
  volatile short *sampleIpBuffer;

//Pointer to raw Speech sample buffer2.
  volatile short *sampleOpBuffer;

//Pointer to encoded speech sample buffer1.
  volatile char *sampleEncodeIpBuffer;

//Pointer to encoded speech sample buffer2.
  volatile char *sampleEncodeOpBuffer;

//Flag to indicate ping-pong buffer filled or empty.
  volatile char fBlockdone;

//Flag to start or stop speech playback.
  volatile char fStartPlay;

//Counter to keep count of number of blocks of data encoded.
  volatile int blockCount;

//Counter to keep count of number of blocks of data decoded.
  volatile int loadblockCount;

//Counter to keep track of number of samples stored.
  volatile int countFill;

//Counter to keep track of number of samples played.
  volatile int countLoad;
```

EXAMPLE 3-3:   `codecsetup` **STRUCTURE (CONTINUED)**

```
//Counter to keep track of number of samples encoded.
  volatile unsigned long sampleCount;

//Number of samples in each frame.
  volatile char numOfSamplesPerFrame;

//Flag to indicate decoding is done.
  volatile char fBlockplayed;

//Flag to indicate compression is done.
  volatile char fCompressdone;

//Flag to indicate encoding is done.
  volatile char fEncodedone;

//Number of sets of data for the set ADC Buffer Length.
  volatile int setOfADCData;

//Pointer to ADCBUF0 register.
  volatile unsigned int* AdcBuf0Ptr;

//Number of bytes in the encoded speech.
  unsigned long arraysizeinbytes;

//Size of the recorded (encoded) speech in number of frames.
  long recordSize;

//G.711 companding method: 1 for A-law, 0 for m-Law.
  char law;

//G.726A output bit-rate: 5 for 40 kbps, 4 for 32 kbps,
//                        3 for 24 kbps, 2 for 16 kbps.
  short rate;
//Used to set Master/Slave in communication applications.
 short initiator;
};
```

The structure `codecdata` of type `codecsetup` is defined for your use in `G726Alib_common.h` file.

```
  typedef struct _codecsetup codecsetup
  extern codecsetup codecdata;
```

> **Note:** The `G726Alib_common.h` file can be customized for each individual application, but modifying the `codecdata` structure is not recommended.

3.4.2   `g726aSi3000` Structure

The `g726aSi3000` structure defined in the `G726Alib_Si3000.h` file represents all the registers of the Si3000 Voiceband Codec. Applications that use the Si3000 codec as the sampling and/or playback interface for speech can use this structure. This structure also includes the settings for the DCI peripheral. This structure can be initialized using the `#define` statements provided in the `G726Alib_Si3000.h` file.

Appendix A. "Si3000 Codec Configuration" contains detailed information about configuring the Si3000 registers.

EXAMPLE 3-4:   `g726asi3000` STRUCTURE

```
struct _g726aSi3000
    {
    int control1;                    //Si3000 Register 1
    int control2;                    //Si3000 Register 2
    int pLL1divideN1;                //Si3000 Register 3
    int pLL1multiplyM1;              //Si3000 Register 4
    int rxgaincontroL1;             //Si3000 Register 5
    int adcvolumecontrol;           //Si3000 Register 6
    int dacvolumecontrol;           //Si3000 Register 7
    int statusreport;               //Si3000 Register 8
    int analogattenuation;          //Si3000 Register 9
    char dcimode;                   //1=master, 0=slave
    char dciintpri;                 //DCI interrupt priority
    int bcg1;                       //bit clock generator
    };
```

Set the `#define` statement in `g726Alib_Si3000.h` file for your application. Another `#define` statement is provided to create a data structure of type `g726aSi3000`:

```
#define G726ASI3000INIT   const g726aSi3000 g726a = G726A;
```

To make the Speex structure accessible to your source application, simply reference the `G726ASI3000INIT` define in your source code, where you define your other data:

```
int my_variable;
G726ASI3000INIT       // Si3000 data structure instantiation
                      // This defines the initialized Si3000
                      // data structure
...
```

3.4.3   `G726_decode()` Function

The `G726_decode()` function is used to:

- Decode a block of 256 encoded speech samples
- Generate an output block of 256 decoded speech samples

Return Value

None

Parameters

This function has five parameters.

Parameter	`codecdata.sampleEncodeOpBuffer`
Data Type	`short *`
Usage	Pointer to encoded speech sample buffer 2

Parameter	`codecdata.sampleDecodeIpBuffer`
Data Type	`short *`
Usage	Pointer to decoded speech sample buffer 1

Parameter	`Slen`
Data Type	`long`
Usage	Number of samples per block

Parameter	`codecdata.rate`
Size	`short`
Usage	Output bit-rate

Parameter	`&decoder_state`
Size	`G726_state`
Usage	Instantiation of decoder state

> **Note:** The 256 decoded speech samples returned in the
> `codecdata.sampleDecodeIpBuffer` pointer must be left-shifted by
> two bits, to retain the correct sign of the data.

3.4.4   `G726_decoder_init()` Function

The `G726_decoder_init()` function is used to:

- Instantiate a structure to store the decoder state
- Initialize the decoder state variables

> **Note:**   This function must be called before `G726_decode()`.

Return Value

None

Parameters

This function has two parameters.

Parameter	`&decoder_state`
Size	`G726_state`
Usage	Instantiation of decoder state

Parameter	`codecdata.rate`
Size	`short`
Usage	Output bit-rate

3.4.5   `G726_encode()` Function

The `G726_encode()` function is used to:

- Encodes a block of 256 raw speech samples
- Generates an output block of 256 encoded speech samples

Return Value

None

Parameters

This function has five parameters.

Parameter	codecdata.sampleOpBuffer
Data Type	short *
Usage	Pointer to raw speech sample buffer 2

Parameter	codecdata.sampleEncodeIpBuffer
Data Type	short *
Usage	Pointer to encoded speech sample buffer 1

Parameter	Slen
Data Type	long
Usage	Number of samples per block

Parameter	codecdata.rate
Size	short
Usage	Output bit-rate

Parameter	&encoder_state
Size	G726_state
Usage	Instantiation of encoder state

> **Note:** The 256 raw speech samples must be right-shifted by two bits before assigning it to the `codecdata.sampleOpBuffer` pointer for encoding.

3.4.6 `G726_encoder_init()` Function

The `G726_encoder_init()` function is used to:

- Instantiates a structure to store the encoder state
- Initializes the encoder state variables

> **Note:** This function must be called before `G726_encode()`.

Return Value

None

Parameters

This function has two parameters.

Parameter	`&encoder_state`
Size	`G726_state`
Usage	Instantiation of encoder state

Parameter	`codecdata.rate`
Size	`short`
Usage	Output bit-rate

3.5   SPEEX API

3.5.1   `codecsetup` Structure

The `codecsetup` structure is defined in the `spxlib_common.h` file. This structure is used to access:

- User defined raw, encoded and decoded speech buffers (described in detail in the next two sections)
- Synchronization flags
- Speech sample counters used for encoding and decoding

A basic understanding of this structure is required for integrating the library with your application.

EXAMPLE 3-5:   `codecsetup` STRUCTURE

```
struct _codecsetup
{
//Pointer to decoded Speech sample buffer1.
  volatile short *sampleDecdIpBuffer;

//Pointer to decoded Speech sample buffer2.
  volatile short *sampleDecdOpBuffer;

//Pointer to raw Speech sample buffer1.
  volatile short *sampleIpBuffer;

//Pointer to raw Speech sample buffer2.
  volatile short *sampleOpBuffer;

//Pointer to encoded speech sample buffer1.
  volatile char *sampleEncdIpBuffer;

//Pointer to encoded speech sample buffer2.
  volatile char *sampleEncdOpBuffer;

//Flag to indicate ping-pong buffer filled or empty.
  volatile char fFramedone;

//Flag to start or stop speech playback.
  volatile char fStartPlay;

//Counter to keep count of number of frames of data encoded.
  volatile int frameCount;

//Counter to keep count of number of frames of data decoded.
  volatile int loadframeCount;

//Counter to keep track of number of samples stored.
  volatile int countFill;

//Counter to keep track of number of samples played.
  volatile int countLoad;

//Counter to keep track of number of samples encoded.
  volatile unsigned long sampleCount;
```

EXAMPLE 3-5:   **codecsetup STRUCTURE (CONTINUED)**

```
//Number of encoded samples in each frame.
  volatile char numOfencSamplesPerFrame;

//Flag to indicate decoding is done.
  volatile char fFrameplayed;

//Flag to indicate encoding is done.
  volatile char fEncodedone;

//Number of sets of data for the set ADC Buffer Length.
  volatile int setOfADCData;

//Pointer to ADCBUF0 register.
  volatile unsigned int* AdcBuf0Ptr;

//Number of bytes in the encoded speech.
  unsigned long arraysizeinbytes;

//Flag to indicate VAD enable (VAD disabled by default).
  char vad;

//Flag to indicate lostframe (cleared by default).
 char lostFrame;

//Size of the recorded (encoded) speech in number of frames.
  long recordSize;

//Used to set Master/Slave in communication applications.
 short initiator;
};
```

The structure codecdata of type codecsetup is defined for your use in spxlib_common.h file.

```
  typedef struct _codecsetup codecsetup
  extern codecsetup codecdata;
```

Note: The spxlib_common.h file may be customized for each individual application, but modifying the codecdata structure is not recommended.

3.5.2 `spxSi3000` Structure

The `spxSi3000` structure defined in the `spxlib_Si3000.h` file represents all the registers of the Si3000 Voiceband Codec. This structure can be used by applications that use the Si3000 codec as the sampling and/or playback interface for speech. This structure also includes the settings for the DCI peripheral. This structure can be initialized using the `#define` statements provided in the `spxlib_Si3000.h` file.

Appendix A. "Si3000 Codec Configuration" contains detailed information about configuring the Si3000 registers.

EXAMPLE 3-6:   `spxsi3000` STRUCTURE

```
struct _spxSi3000
    {
    int control1;                //Si3000 Register 1
    int control2;                //Si3000 Register 2
    int pLL1divideN1;            //Si3000 Register 3
    int pLL1multiplyM1;          //Si3000 Register 4
    int rxgaincontroL1;          //Si3000 Register 5
    int adcvolumecontrol;        //Si3000 Register 6
    int dacvolumecontrol;        //Si3000 Register 7
    int statusreport;            //Si3000 Register 8
    int analogattenuation;       //Si3000 Register 9
    char dcimode;                //1=master, 0=slave
    char dciintpri;              //DCI interrupt priority
    int bcg1;                    //bit clock generator
    };
```

Set the `#define` statement in `spxlib_Si3000.h` file for your application. Another `#define` statement is provided to create a data structure of type `spxSi3000`:

```
#define SPXSI3000INIT   const spxSi3000 speex = SPEEX;
```

To make the speex structure accessible to your source application, simply reference the `SPXSI3000INIT` define in your source code, where you define your other data:

```
int my_variable;
SPXSI3000INIT        // Si3000 data structure instantiation
                     // This defines the initialized Si3000
                     // data structure
...
```

3.5.3 `libDecoder()` Function

The `libDecoder()` function is used to:

- Decodes a frame of encoded speech (5 or 20 bytes)
- Generates an output of 160 speech integer samples for playback

> **Note:** `libDecoder()` is the primary decoder function.

Return Value

None

Parameters

None

3.5.4 `libDecoderInit()` Function

The `libDecoderInit()` function initializes the decoder state variables.

> **Note:** This function must be called before `libDecoder()`.

Return Value

None

Parameters

None

3.5.5 `libFullDuplexDecoder()` Function

The `libFullDuplexDecoder()` function is used to:

- Decodes a frame of encoded speech (5 or 20 bytes), typically obtained from a communication channel
- Generates an output of 160 speech integer samples for playback

> **Note:** `libFullDuplexDecoder()` function is used only in Full-Duplex applications (applications in which data is both encoded and decoded concurrently).

Return Value

None

Parameters

None

3.5.6 `libEncoder()` Function

The `libEncoder()` function is used to:

- Encodes a frame of 160 speech samples
- Generates an output frame of encoded speech 5 or 20 bytes long, depending on the state of `codecdata.vad` ('0' or '1', respectively)

> **Note:** `libEncoder()` is the primary encoder function.

Return Value

None

Parameters

None

3.5.7 `libEncoderInit()` Function

The `libEncoderInit()` function is used to:

- Dynamically allocates 1280 bytes of memory from the heap
- Initializes the encoder state variables

> **Note:** This function must be called before `libEncoder()`.

Return Value

None

Parameters

The `libEncoderInit()` function has one parameter:

Parameter	`vad_enabled`
Size	`char`
Usage	'1' = enable VAD '0' = disable VAD

3.5.8 `libEncoderKill()` Function

The `libEncoderKill()` function is used to:

- Frees the 1282 bytes dynamically allocated by `libEncoderInit()` function
- Clears the encoder state pointer

Return Value

None

Parameters

None

Chapter 4. Integrating Speech Encoding Into YourApplication

This chapter provides information to help you understand how to integrate the speech encoding portion of the G.711, G.726A and Speex libraries into your application and how to build with the library. Topics include:

- Integrating Speech Encoding
- Data Buffers
- Encoder Initialization
- Encoder Heap Utilization
- Data Sampling Initialization
- Data Sampling
- Encoding
- End Data Sampling

A basic understanding of the encoder and interrupt timing is required to ensure correct real-time operation of the library.

4.1 INTEGRATING SPEECH ENCODING

To interface your application with the encoder, you need to be familiar with:

- How the encoder is initialized to work with the input (the codec interface, on-chip ADC, or any other sampling interface)
- How data is sampled
- How data buffers are used by the encoder
- How the library interacts with its interrupt handlers

4.2 DATA BUFFERS

The encoder uses four data buffers, which you must define. Two of these buffers are input buffers used to store sampled speech data. The other two buffers are output buffers used to store encoded speech data. Table 4-1 defines the parameters for these buffers.

TABLE 4-1: SPEECH ENCODING DATA BUFFER REQUIREMENTS

Parameter	Library		
	G.711	**G.726A**	**Speex**
Input sampling rate	8 kHz (of 16-bit data)	8 kHz (of 16-bit data)	8 kHz (of 16-bit data)
Encoder frame size	32 ms	32 ms	20 ms
Buffer capacity	256 integer samples	256 integer samples	160 integer samples

When the encoder processes a frame of data, it generates an output array, whose size depends on the encoding algorithm being used. For example, the output data frame for Speex may be as large as 20 bytes. In this case, the two output buffers must be large enough to hold 20 bytes. Example buffer definitions are shown in Example 4-1:

EXAMPLE 4-1: **BUFFER DEFINITIONS EXAMPLE**

```
short RawBuf1[160], RawBuf2[160]; /* ping-pong input buffers */
char EncdBuf1[20], EncdBuf2[20]; /* ping-pong output buffers */
```

A pair of each type of buffer is needed since the encoding library ping-pongs, or alternates, between input/output buffer pairs.

For instance, when sampling begins, the RawBuf1 buffer is populated with speech data. At the end of each processing interval (20 ms in the case of Speex), a frame of data is received and RawBuf1 is filled. The library processes RawBuf1 and populates EncdBuf1, the encoded speech data.

Since the sampling process must be continuous, a second input buffer is needed to store the data sampled in Frame 1 while Frame 0 is processed by the library. After RawBuf1 is processed, the output is stored in EncdBuf1. Likewise, after RawBuf2 is processed, the output is stored in EncdBuf2. This process allows you to safely use the encoded data (for transmission or storage) in EncdBuf1, while EncdBuf2 is being populated (and vice-versa).

Table 4-2 shows how the pairs of input/output buffers are used by the library.

TABLE 4-2: **ENCODER BUFFER USAGE (SPEEX EXAMPLE)**

Buffer	Frame 0 (20 ms)	Frame 1 (20 ms)	Frame 2 (20 ms)	Frame 3 (20 ms)	Frame 4 (20 ms)
RawBuf1	Filled by ISR	Processed by library	Filled by ISR	Processed by library	Filled by ISR
RawBuf2	Idle	Filled by ISR	Processed by library	Filled by ISR	Processed by library
EncdBuf1	Idle	Loaded with Encoded in1	Available for user handling	Loaded with Encoded in1	Available for user handling
EncdBuf2	Idle	Idle	Loaded with Encoded in2	Available for user handling	Loaded with Encoded in2

4.3   ENCODER INITIALIZATION

4.3.1   G.711 Encoder Initialization

The G.711 (A-law and µ-law) encoder does not need to be initialized. Therefore, there is no initialization function for this encoder.

4.3.2   G.726A Encoder Initialization

The G.726A encoder is initialized by calling the G726_encoder_init() function with the desired output bit-rate at which the speech needs to be encoded (since this encoder supports multiple bit-rates).

The RATE setting is stored in the rate element in the codecdata structure. This structure is defined in the G726ALib_common.h include file. The codecdata.rate parameter is used as an argument to the G726_encoder_init() function.

The user application also needs to define an instantiation of the G726_state structure and pass its address as the other argument to the G726_encoder_init() function.

4.3.3 Speex Encoder Initialization

The Speex encoder is initialized by calling the `libEncoderInit()` function with the desired Voice Activity Detection (VAD) setting. When VAD is enabled, the library differentiates between speech and silence (background noise). Non-speech periods are encoded with just enough data (5 bytes per frame instead of 20 bytes) to reproduce the background noise.

The VAD setting is stored in the `codecdata.vad` structure. This structure is defined in `spxlib_common.h` and initialized by the `CODECDATA #define` statement. The `codecdata.vad` structure element is used as the argument for `libEncoderInit()`:

```
libEncoderInit (codecdata.vad);
```

When the `codecdata.vad` structure element is initialized to '1', VAD is enabled. When `codecdata.vad` is initialized to '0', VAD is disabled. The VAD feature cannot be enabled or disabled on a frame-by-frame basis. After `libEncoderInit()` is called, the VAD setting must not be modified.

4.4 ENCODER HEAP UTILIZATION

The G.711 and G.726A encoder and decoder algorithms do not use a heap. Therefore, they do not require heap initialization by the user application.

The Speex encoder requires 1282 bytes of scratch RAM. As a benefit to your application, this memory is allocated dynamically by `libEncoderInit()`. As a result, you recover this memory for your application after the encoder completes running. When building your application, you must define a heap size of 1282 bytes for the encoder. If you do not reserve at least 1282 bytes for the heap, your application will either not build or it will run incorrectly.

The Speex decoder does not require heap initialization.

4.5 DATA SAMPLING INITIALIZATION

After the encoder is initialized by calling `libEncoderInit()`, the sampling system (DCI module and Si3000 codec) must be initialized. The appropriate sampling interfaces are initialized using constants defined in the `spxlib_Si3000.h` (or `G711lib_Si3000.h` or `G726Alib_Si3000.h`) include file. The addresses of the four data buffers must then be assigned to the corresponding structures of the `codecdata` structure, as shown in Example 4-2 :

EXAMPLE 4-2:   SPEEX EXAMPLE

```
codecdata.sampleIpBuffer = RawBuf1;
codecdata.sampleOpBuffer = RawBuf2;
codecdata.sampleEncdIpBuffer = EncdBuf1;
codecdata.sampleEncdOpBuffer = EncdBuf2;
```

4.6 DATA SAMPLING

Data sampling is typically managed by either the DCI ISR, if you are using an external Voiceband Codec, or the 12-bit ADC ISR, if you are sampling from the 12-bit ADC. Each ISR reads speech samples from the respective peripheral and stores the data in your defined input buffers.

When a complete frame of 256 or 160 speech samples has been received by the ISR, the ISR should set the `codecdata.fFramedone` flag to '1' and perform input buffer management. This process allows new data to be collected by the ISR while the foreground library processes the newly received frame of speech data.

If you are using the DCI module, DCI ISRs can be configured to execute every 500 µs instead of the speech sample period of 125 µs (1/8 kHz) to minimize the impact of the ISR on your application. To do this, initialize the DCI with the buffer length control bits BL<1:0> = '11b'. This mode allows four data samples to be buffered between interrupts, thereby decreasing the interrupt rate by a factor of four.

4.7   ENCODING

4.7.1   G.711 Encoding

Speech encoding is performed by the `alaw_compress()` or `µlaw_compress()` function. This function can only be called after the sampling interface has received a full block of data for processing. When 256 samples of speech data have been received by the sampling ISR, the `codecdata.fBlockdone` flag should be set to '1', which signifies that `alaw_compress()` or `µlaw_compress()` can be called.

Since each data frame is 20 ms long, this function must be called 32 times each second to maintain continuous processing of data. To ensure efficient real-time performance, the `alaw_compress()` or `µlaw_compress()` function must not be run from an ISR.

Immediately after the `alaw_compress()` or `µlaw_compress()` function has been executed, the user application should swap the two ping-pong output buffer pointers, clear the `codecdata.fBlockdone` flag, and set the `codecdata.fCompressdone` flag.

At this point, the encoded data can be used (stored and/or transmitted, depending on the application) by the application. The encoded data will always be pointed by the `codecdata.sampleComprsOpBuffer` pointer, and the number of bytes encoded (256) will be stored in `codecdata.numOfSamplesPerFrame`. You must access the encoded data using these structure elements.

4.7.2   G.726A Encoding

Speech encoding is performed by the `G726_encode()` function. This function can only be called after the sampling interface has received a full frame of data for processing. When 256 samples of speech data have been received by the sampling ISR, the `codecdata.fBlockdone` flag should be set to '1', which signifies that `G726_encode()` can be called.

Since each data frame is 32 ms long, this function must be called 32 times each second to maintain continuous processing of data. To ensure efficient real-time performance, the `G726_encode()` function must not be run from an ISR.

Immediately after the `G726_encode()` function has been executed, the user application should swap the two ping-pong output buffer pointers, clear the `codecdata.fBlockdone` flag, and set the `codecdata.fEncodedone` flag.

At this point, the encoded data can be used (stored and/or transmitted, depending on the requirement) by the application. The encoded data will always be pointed by the `codecdata.sampleEncodeOpBuffer` pointer, and the number of bytes encoded (256) will be stored in `codecdata.numOfSamplesPerFrame`. You must access the encoded data using these structure elements.

4.7.3   Speex Encoding

Speech encoding is performed by the `libEncoder()` function. This function can only be called after the sampling interface (e.g., DCI) has received a full frame of data for processing. When 160 samples of speech data have been received by the sampling ISR, the `codecdata.fFramedone` flag should be set to '1', which signifies that `libEncoder()` can be called.

Since each data frame is 20 ms long, this function must be called 50 times each second to maintain continuous processing of data. To ensure efficient real-time performance, the `libEncoder()` function must not be run from an ISR.

Immediately after the `libEncoder()` function has been executed, the user application should swap the two ping-pong output buffer pointers, clear the `codecdata.fFramedone` flag, and set the `codecdata.fEncodedone` flag.

At this point, the encoded data can be used (stored and/or transmitted, depending on the application) by the application. The encoded data will always be pointed by the `codecdata.sampleEncdOpBuffer` pointer, and the number of bytes encoded (5 bytes or 20 bytes) will be stored in `codecdata.numOfencSamplesPerFrame`. You must access the encoded data using these structure elements.

4.8   END DATA SAMPLING

For your convenience, the number of speech frames encoded by the library is saved in these structure elements:

- `codecdata.blockCount` for G.711 and G.726A
- `codecdata.frameCount` for Speex

You can use this information to determine when to stop sampling. This can be required if you are storing the encoded data to memory and you are concerned about exceeding your application's storage capacity.

The `codecdata.recordSize` structure element is made available to store the number of frames you want to encode. Your application can compare `codecdata.frameCount` (or `codecdata.blockCount`) with `codecdata.recordSize` to determine when sampling must stop. If you want to use this feature, you must manually perform this comparison in your application, using the user-specified `RECORDSIZE` constant provided in `G711_common.h`, `G726A_common.h` or `spxlib_common.h`:

```
#define RECORDSIZE 750 // encode 750 frames (15 seconds)
```

Data sampling can be stopped by disabling the interrupt service routines of the sampling interface. The sample code sequence is as shown in Example 4-3 (for Speex) listing the steps that should be performed when sampling is stopped and you want to return the encoder to an idle state:

EXAMPLE 4-3:   SAMPLE CODE SEQUENCE FOR SPEEX

```
libEncoderKill ();             /* destroys Encoder state */
codecdata.fFramedone = 0x00; /* clear sampling flag */
codecdata.frameCount = 0x00; /* clear number of frames encoded */
```

The `libEncoderKill()` function will free the 1282 bytes of scratch memory reserved by `libEncoderInit()` from the heap. Your application can use this RAM after `libEncoderKill()` runs. For detailed information about heap requirements, see **Section 4.4 "Encoder Heap Utilization"**.

dsPIC® DSC Speech Coding Solutions User's Guide

NOTES:

Chapter 5. Integrating Speech Decoding Into YourApplication

This chapter provides information to help you understand how to integrate the speech decoding portion of the G.711, G.726A and Speex libraries into your application and how to build with the library. Topics include:

- Integrating Speech Decoding
- Data Buffers
- Decoder Initialization
- Decoder Heap Utilization
- Decoding the First Frame
- Speech Playback Initialization
- Speech Playback
- Decoding
- Ending Speech Playback

A basic understanding of the decoder and interrupt timing is required to ensure correct real-time operation of the library.

5.1   INTEGRATING SPEECH DECODING

To interface your application with the decoder, you need to be familiar with:

- How data buffers are used by the decoder
- How the decoder is initialized
- How speech playback works
- How the library decoder interacts with its interrupt handlers

5.2   DATA BUFFERS

The decoder uses four data buffers, which you must define. Two of these buffers are input buffers used to store encoded speech data. The other two buffers are output buffers used to store decoded speech data for playback. Table 5-1 defines the parameters for these buffers.

TABLE 5-1:   SPEECH DECODING DATA BUFFER REQUIREMENTS

Parameter	Library		
	G.711	G.726A	Speex
Output sampling rate	8 kHz (of 16-bit data)	8 kHz (of 16-bit data)	8 kHz (of 16-bit data)
Decoder frame size	32 ms	32 m	20 ms
Output Array	256 integer samples	256 integer samples	160 integer samples

For example, the encoded data frame for Speex may be as large as 20 bytes, while the decoded output will contain 160 samples. Example buffer definitions are shown in Example 5-1:

dsPIC® DSC Speech Coding Solutions User's Guide

Example 5-1: **Buffer Definitions Example**

```
char EncdBuf1[20], EncdBuf2[20];    /* ping-pong i/p buffers */
short DecdBuf1[160], DecdBuf2[160]; /* ping-pong o/p buffers */
```

A pair of each type of buffer is needed since the decoding library ping-pongs, or alternates between input/output buffer pairs.

For instance, data is loaded into EncdBuf1 for decoding, and it is processed by the library. After executing, the decoder populates the DecdBuf1 buffer with speech data. When output sampling begins, DecdBuf1 is played back through the respective ISR handler over the course of the next processing interval (20 ms in the case of Speex).

Since the speech playback process must be continuous, EncdBuf2 is filled and processed by the library, which populates the DecdBuf2 buffer, while DecdBuf1 is being played back.

Likewise, when DecdBuf2 is being played back, DecdBuf1 is being populated with new data obtained by decoding the data from EncdBuf1. The EncdBuf1 and EncdBuf2 buffers are also used in an alternating fashion, which allows one buffer to be optionally pre-loaded as the other input buffer is being processed.

Table 5-2 shows how the pairs of input/output buffers are used by the library.

TABLE 5-2: **DECODER BUFFER USAGE (SPEEX EXAMPLE)**

Buffer	Initialization	Frame 0 (20 ms)	Frame 1 (20 ms)	Frame 2 (20 ms)	Frame 3 (20 ms)
EncdBuf1	Filled and processed by library	Idle (available for filling)	Filled and processed by library	Idle (available for filling)	Filled and processed by library
EncdBuf2	Idle (available for filling)	Filled and processed by library	Idle (available for filling)	Filled and processed by library	Idle (available for filling)
DecdBuf1	Loaded with Decoded in1	Played out by ISR	Loaded with Decoded in1	Played out by ISR	Loaded with Decoded in1
DecdBuf2	Idle	Loaded with Decoded in2	Played out by ISR	Loaded with Decoded in2	Played out by ISR

5.3 DECODER INITIALIZATION

5.3.1 G.711 Decoder Initialization

The G.711 (A-law and μ-law) decoder does not need to be initialized. Therefore, there is no initialization function for this decoder.

5.3.2 G.726A Decoder Initialization

The G.726A decoder is initialized by calling the G726_decoder_init() function with the desired input bit-rate at which the speech has been encoded (since this decoder supports multiple bit-rates).

The RATE setting is stored in the rate element in the codecdata structure. This structure is defined in the G726ALib_common.h include file. The codecdata.rate parameter is used as an argument to the G726_decoder_init() function.

The user application also needs to define an instantiation of the G726_state structure and pass its address as the other argument to the G726_decoder_init() function.
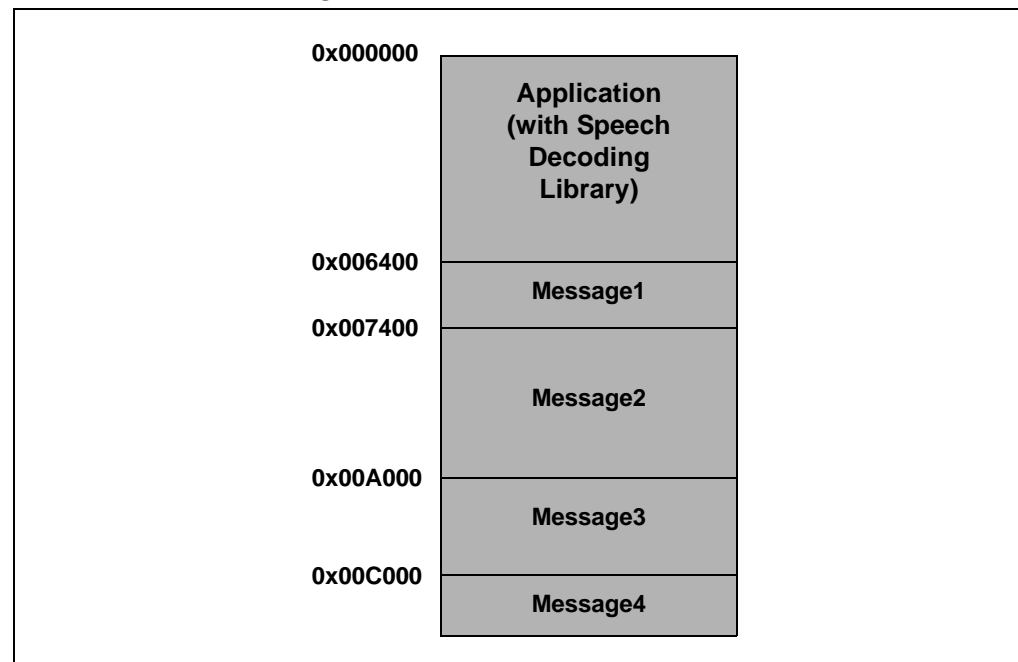
5.3.3 Speex Decoder Initialization

The Speex decoder is initialized by calling the `libDecoderInit()` function. This function initializes the decoder state variables. The decoder is automatically capable of processing frames of data that are encoded either with or without VAD. The VAD selection need not be specified by the user for the decoder to function correctly.

If you will be decoding a speech sample stored in data EEPROM, program Flash or external Flash memory, you must also initialize any registers (e.g., TBLPAG to access data EEPROM or program Flash memory) or external devices that are to be used.

The user application needs to define which encoded speech sample stored in memory will be decoded by the library. For instance, if you used the speech encoding utility to create four different messages for your application (Message1, Message2, Message3 and Message4) and stored them in program memory, they will be stored in arbitrary locations along with your application code and the library, as shown in Figure 5-1. All your messages must be created with a unique array name, which will allow them to be accessed by the library.

FIGURE 5-1:   **EXAMPLE OF MULTIPLE MESSAGES STORED IN PROGRAM MEMORY**

```
0x000000
            ┌──────────────────────┐
            │     Application       │
            │   (with Speech        │
            │     Decoding          │
            │      Library)         │
0x006400    ├──────────────────────┤
            │      Message1         │
0x007400    ├──────────────────────┤
            │                       │
            │      Message2         │
            │                       │
0x00A000    ├──────────────────────┤
            │      Message3         │
0x00C000    ├──────────────────────┤
            │      Message4         │
            └──────────────────────┘
```

You can specify and refer a list of unique array names in the `spxlib_common.inc` file provided with the library, as shown in Example 5-2. This file can be used to define a table with up to ten entries (depending on the memory available on the specific device being used). Note that there is a leading underscore in each array name.

EXAMPLE 5-2:   EXAMPLE OF UNIQUE ARRAY NAMES

```
.equ TABLENAME1,          _Message1
.equ TABLENAME2,          _Message2
.equ TABLENAME3,          _Message3
.equ TABLENAME4,          _Message4
.equ TABLENAME5,          0                 ; not used
.equ TABLENAME6,          0                 ; not used
.equ TABLENAME7,          0                 ; not used
.equ TABLENAME8,          0                 ; not used
.equ TABLENAME9,          0                 ; not used
.equ TABLENAME10,         0                 ; not used
```

To initiate the playback of one of the above messages from program memory, the user application can use an index variable to specify which message is to be played back, and use the corresponding table page and offset values to fetch encoded data from the appropriate array.

5.4   DECODER HEAP UTILIZATION

None of the three decoders described requires a heap. All memory used by the library is pre-allocated.

5.5   DECODING THE FIRST FRAME

Before the output sampling system is initialized, one frame of speech must be first decoded. If this step is not performed, uninitialized data stored in the decoder's output buffers will be played back, which can lead to undesirable results.

Decoding begins by first populating an input buffer of the decoder. After the decoder input data is read from memory, decoding is performed by calling `libDecoder()` and then performing buffer management (swapping the decoded data buffer pointers).

5.6   SPEECH PLAYBACK INITIALIZATION

After the decoder is initialized and one frame of speech has been decoded, the output sampling system (DCI module and Si3000 codec) must be initialized for speech playback. The appropriate sampling interfaces are initialized using constants defined in the `spxlib_Si3000.h` (or `G711lib_Si3000.h` or `G726Alib_Si3000.h`) include file. The addresses of the four data buffers must then be assigned to the corresponding structures of the `codecdata` structure, as shown in Example 5-3.

EXAMPLE 5-3:   SPEEX EXAMPLE

```
codecdata.sampleEncdIpBuffer = EncdBuf1;
codecdata.sampleEncdOpBuffer = EncdBuf2;
codecdata.sampleDecdIpBuffer = DecdBuf1;
codecdata.sampleDecdOpBuffer = DecdBuf2;
```

5.7   SPEECH PLAYBACK

Speech playback is typically managed by the DCI ISR, if you are using an external Voiceband Codec, or the 12-bit ADC ISR, if you are sampling from the 12-bit ADC. Each ISR writes decoded speech samples to its respective peripheral from the decoder's output buffers. When a complete frame of decoded speech (256 or 160 samples) has been played out by the ISR, the ISR should set the `codecdata.fFrameplayed` flag to '0', process the output buffer and perform buffer management. This process allows new decoded data to be played by the ISR while the foreground library code decodes another frame of speech.

If you are using the DCI module, DCI interrupt service routines can be configured to execute every 500 µs instead of the speech sample period of 125 µs (1/8 kHz) to minimize the impact of the ISR on your application. To do this, initialize the DCI with the buffer length control bits BL<1:0> = 'b11. This mode allows four data samples to be buffered between interrupts, thereby decreasing the interrupt rate by a factor of four.

5.8   DECODING

5.8.1   G.711 Decoding

Before decoding can be performed, the appropriate input buffer must be first loaded with data. The `codecdata.fBlockdone` and `codecdata.fBlockplayed` structure elements can be used to manage the loading of data into the correct input buffer and the playback of data from the correct output buffer, as shown in Table 5-3.
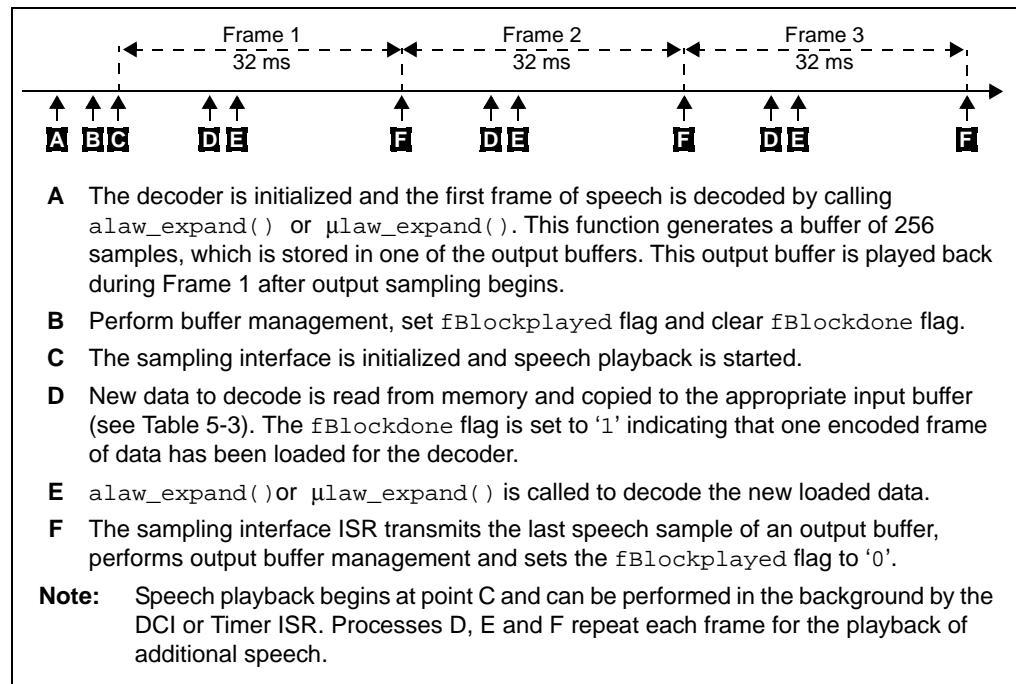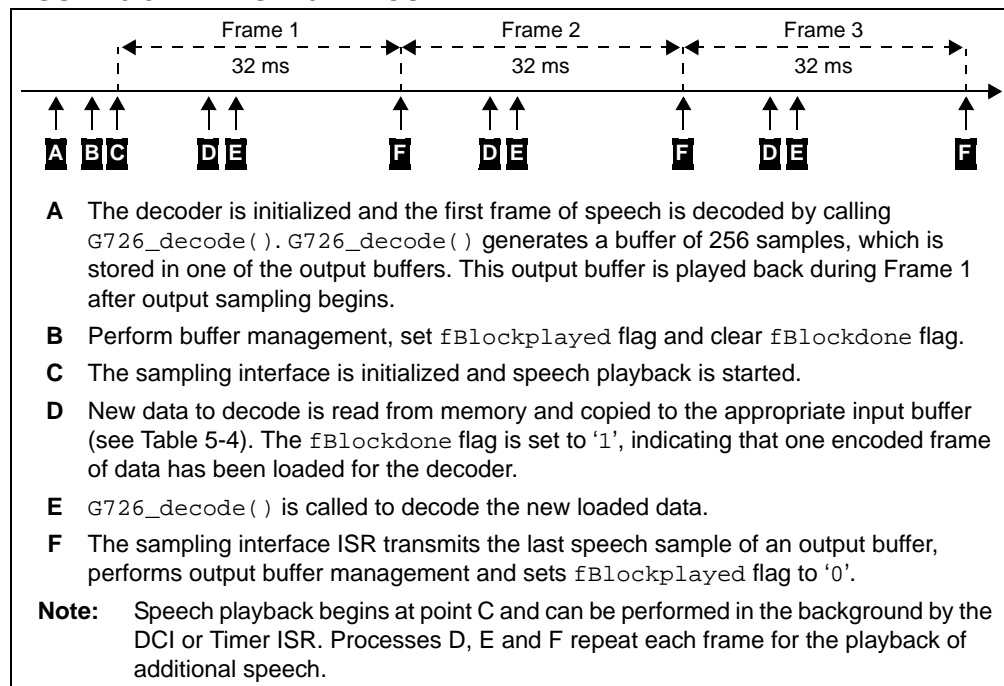
TABLE 5-3:   BUFFER MANAGEMENT DATA STRUCTURES

Element	Function
`codecdata.fBlockdone`	0 – Copy data to input buffer 1 – Process data from input buffer
`codecdata.fBlockplayed`	0 – Output buffer ready to be read (ready for playback) 1 – Output buffer is being read (playback in process)

If the `codecdata.fBlockdone` flag is '0' (indicating that data needs to be copied to a decoder input buffer) and the `codecdata.fBlockplayed` flag is '1' (indicating that an output buffer is being played back), a new frame of encoded data can be read into a decoder input buffer.

Once the new input data has been read and the `codecdata.fBlockdone` flag is '1', the `alaw_expand()` or `µlaw_expand()` function is called to perform the decoding. This function converts the encoded input buffer to a 256 word decoded buffer for speech playback. To maintain synchronization with the two sets of input/output buffers, decoded data buffer management must be performed after the `codecdata.fBlockplayed` flag is '0'. A decoder timeline is shown in Figure 5-2.

FIGURE 5-2:   G.711 DECODER TIMELINE EXAMPLE

A   The decoder is initialized and the first frame of speech is decoded by calling `alaw_expand()` or `µlaw_expand()`. This function generates a buffer of 256 samples, which is stored in one of the output buffers. This output buffer is played back during Frame 1 after output sampling begins.

B   Perform buffer management, set `fBlockplayed` flag and clear `fBlockdone` flag.

C   The sampling interface is initialized and speech playback is started.

D   New data to decode is read from memory and copied to the appropriate input buffer (see Table 5-3). The `fBlockdone` flag is set to '1' indicating that one encoded frame of data has been loaded for the decoder.

E   `alaw_expand()` or `µlaw_expand()` is called to decode the new loaded data.

F   The sampling interface ISR transmits the last speech sample of an output buffer, performs output buffer management and sets the `fBlockplayed` flag to '0'.

Note: Speech playback begins at point C and can be performed in the background by the DCI or Timer ISR. Processes D, E and F repeat each frame for the playback of additional speech.

The locations of events D and E are arbitrary and occur under application control. They can occur at any time within each 20 ms frame, as long as the fBlockdone and fBlockplayed state conditions defined above are satisfied. Your application can execute code between points C-D, D-E and E-F. When your application code is executing, you must always allow the sampling interface ISR to run unimpeded. Failure to let the ISR run as normal will result in degraded audio playback quality.

5.8.2   G.726A Decoding

Before decoding can be performed, the appropriate input buffer must be first loaded with data. The codecdata.fBlockdone and codecdata.fBlockplayed structure elements can be used to manage the loading of data into the correct input buffer and the playback of data from the correct output buffer, as shown in Table 5-4.
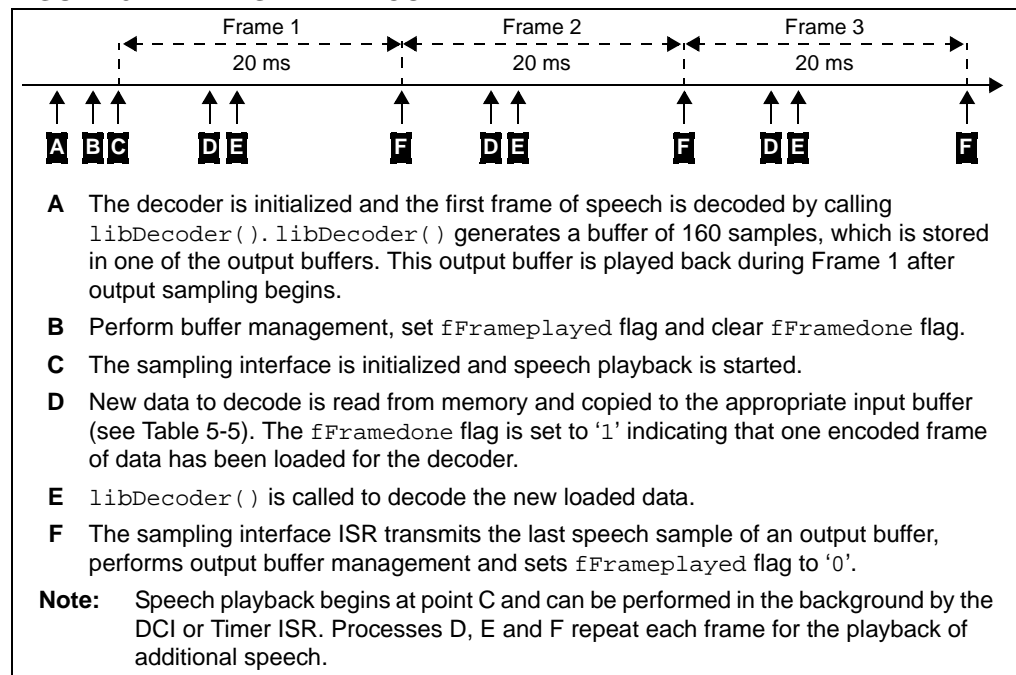
TABLE 5-4:   BUFFER MANAGEMENT DATA STRUCTURES

Element	Function
codecdata.fBlockdone	0 – copy data to input buffer 1 – process data from input buffer
codecdata.fBlockplayed	0 – output buffer ready to be read (ready for playback) 1 – output buffer is being read (playback in process)

A new frame of encoded data can be read into a decoder input buffer if these two conditions are met:

• The codecdata.fBlockdone flag is '0' (indicating that data needs to be copied to a decoder input buffer)
• The codecdata.fBlockplayed flag is '1' (indicating that an output buffer is being played back)

Once the new input data has been read and the codecdata.fBlockdone flag is '1', the G726_decode() function is called to perform the decoding. This function converts the encoded input buffer to a 256 word decoded buffer for speech playback. To maintain synchronization with the two sets of input/output buffers, decoded data buffer management must be performed after the codecdata.fBlockplayed flag is '0'. A decoder timeline is shown in Figure 5-3.

FIGURE 5-3:   G.726A DECODER TIMELINE EXAMPLE

A   The decoder is initialized and the first frame of speech is decoded by calling G726_decode(). G726_decode() generates a buffer of 256 samples, which is stored in one of the output buffers. This output buffer is played back during Frame 1 after output sampling begins.

B   Perform buffer management, set fBlockplayed flag and clear fBlockdone flag.

C   The sampling interface is initialized and speech playback is started.

D   New data to decode is read from memory and copied to the appropriate input buffer (see Table 5-4). The fBlockdone flag is set to '1', indicating that one encoded frame of data has been loaded for the decoder.

E   G726_decode() is called to decode the new loaded data.

F   The sampling interface ISR transmits the last speech sample of an output buffer, performs output buffer management and sets fBlockplayed flag to '0'.

Note: Speech playback begins at point C and can be performed in the background by the DCI or Timer ISR. Processes D, E and F repeat each frame for the playback of additional speech.

The locations of events D and E are arbitrary and occur under application control. They can occur at any time within each 20 ms frame, as long as the `fBlockdone` and `fBlockplayed` state conditions defined above are satisfied. Your application can execute code between points C-D, D-E and E-F. When your application code is executing, you must always allow the sampling interface ISR to run unimpeded. Failure to let the ISR run as normal can degrade audio playback quality.

5.8.3 Speex Decoding

Before decoding can be performed, the appropriate input buffer must be first loaded with data. The `codecdata.fFramedone` and `codecdata.fFrameplayed` structure elements can be used to manage the loading of data into the correct input buffer and the playback of data from the correct output buffer, as shown in Table 5-5.

TABLE 5-5:   BUFFER MANAGEMENT DATA STRUCTURES

Element	Function
`codecdata.fFramedone`	`0` – copy data to input buffer `1` – process data from input buffer
`codecdata.fFrameplayed`	`0` – output buffer ready to be read (ready for playback) `1` – output buffer is being read (playback in process)

A new frame of encoded data can be read into a decoder input buffer if these two conditions are met:

- The `codecdata.fFramedone` flag is '0' (indicating that data needs to be copied to a decoder input buffer)
- The `codecdata.fFrameplayed` flag is '1' (indicating that an output buffer is being played back)

Once the new input data has been read and the `codecdata.fFramedone` flag is '1', the `libDecoder()` function is called to perform the decoding. This function converts the 5-byte or 20-byte input buffer to a 160-word buffer for speech playback. To maintain synchronization with the two sets of input/output buffers, decoded data buffer management must be performed after the `codecdata.fFrameplayed` flag is '0'. A decoder timeline is shown in Figure 5-4.

FIGURE 5-4:   SPEEX DECODER TIMELINE EXAMPLE

A   The decoder is initialized and the first frame of speech is decoded by calling `libDecoder()`. `libDecoder()` generates a buffer of 160 samples, which is stored in one of the output buffers. This output buffer is played back during Frame 1 after output sampling begins.

B   Perform buffer management, set `fFrameplayed` flag and clear `fFramedone` flag.

C   The sampling interface is initialized and speech playback is started.

D   New data to decode is read from memory and copied to the appropriate input buffer (see Table 5-5). The `fFramedone` flag is set to '1' indicating that one encoded frame of data has been loaded for the decoder.

E   `libDecoder()` is called to decode the new loaded data.

F   The sampling interface ISR transmits the last speech sample of an output buffer, performs output buffer management and sets `fFrameplayed` flag to '0'.

Note: Speech playback begins at point C and can be performed in the background by the DCI or Timer ISR. Processes D, E and F repeat each frame for the playback of additional speech.

The locations of events D and E are arbitrary, and occur under your control. They can occur at any time within each 20 ms frame, as long as the `fFramedone` and `fFrameplayed` state conditions defined above are satisfied. Your application can execute code between points C-D, D-E and E-F. When your application code is executing, you must always allow the sampling interface ISR to run unimpeded. Failure to let the ISR run as normal can degrade audio playback quality.

5.9   ENDING SPEECH PLAYBACK

For your convenience, the number of bytes decoded by the library is saved in `codecdata.sampleCount`. The `codecdata.arraysizeinbytes` structure element is made available for you to store the size of your encoded speech record.

In your application, you can compare `codecdata.sampleCount` with `codecdata.arraysizeinbytes` to determine when the entire speech record has been played and stop the speech playback.

To use this feature, you must manually perform this comparison in your application, using the provided `#define` statement in the `G711_common.h`, `G726A_common.h` or `spxlib_common.h` file:

```
#define ARRAYSIZEINBYTES 2600 // record is 2600 bytes
```

Speech playback can be stopped by disabling the ISR of the sampling interface. The sample code sequence (for Speex), as shown in Example 5-4, lists the steps that should be performed when sampling is stopped and you want to return the decoder to an idle state.

EXAMPLE 5-4:   SAMPLE CODE SEQUENCE (FOR SPEEX)

```
codecdata.fFramedone     = 0x0;
codecdata.frameCount     = 0x0;
codecdata.sampleCount    = 0x0;
codecdata.fFrameplayed   = 0x0;
libStopPlay ();
libDecoderKill ();
```

Chapter 6. Speech Encoding Utility

Each of the Speech Encoding/Decoding libraries described in this document includes a PC-based speech encoding utility that allows you to create your own encoded speech files on your personal computer. The files created from the speech encoding utility can then be built into your application for playback on the dsPIC device using the corresponding Decoder function. This chapter describes how to use the speech encoding utility. Items discussed include:

- System Requirements
- Overview
- Encoding Speech from a Microphone
- Encoding Speech from a WAVE (.wav) file
- Recommendations for Encoding from a Microphone
- Using the Command Line Decoder

6.1 SYSTEM REQUIREMENTS

The system requirements are as follows:

- PC running Windows 95/98/ME/NT 4.0/2000/XP
- Sound card
- Microphone

6.2 OVERVIEW

The Speech Encoding and Decoding functions are designed to optimize computational performance and minimize RAM usage for speech-based applications embedded in dsPIC devices. The Speech Encoding Utility allows you to create encoded speech files from a microphone or from a pre-recorded WAVE (.wav) file, as shown in Figure 6-1, and target the encoded file for on-chip or off-chip memory.

FIGURE 6-1:   OVERVIEW OF SPEECH ENCODING UTILITY
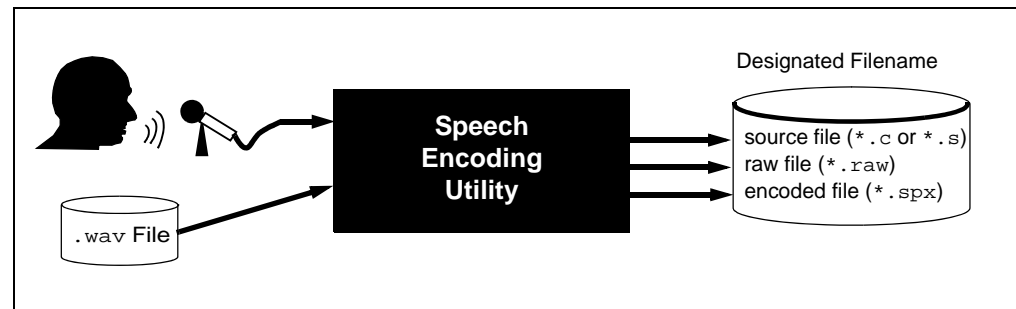
The encoding process creates three output files:

- Source file for your application, in either C (*.c) or assembly (*.s) format
- Raw uncompressed (8 kHz, 16-bit mono) speech file (*.raw)
- Encoded (*.spx) file

The Speech Encoding Utility allows you to select the type of memory in which to store your encoded speech file. The target memory selection ensures that the file is encoded in the correct format for:

- Program Memory
- RAM
- Data EEPROM (only supported for dsPIC30F and Speex Library)
- External Flash (only supported for dsPIC30F and Speex Library)

External Flash memory allows you to store several minutes of speech (one minute of speech requires 60 Kbytes of memory). It is supported through a dsPIC30F general purpose I/O port.

The encoded source file must be added to your MPLAB IDE project and built into your application. The `*.raw` and `*.spx` files remain on your PC for your use.

FIGURE 6-2:   **OVERVIEW OF SPEECH ENCODING UTILITY**
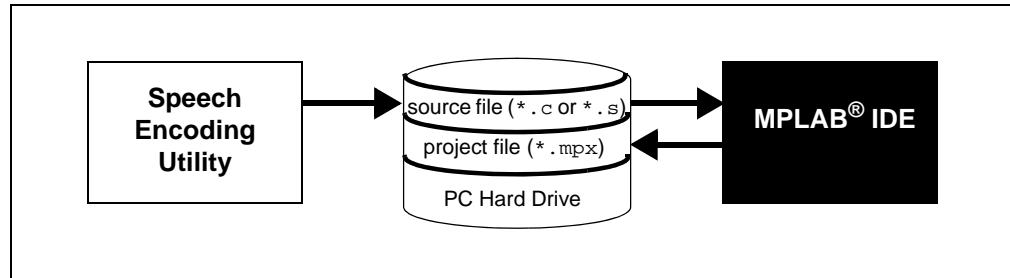
6.3 ENCODING SPEECH FROM A MICROPHONE

To create your own encoded speech file from a microphone, use this procedure.

Select microphone input:

1. Launch the Windows Master Volume Control
   (*Start>Programs>Accessories>Entertainment>Volume Control*). When the Master Volume dialog displays, select *Options>Properties*, as shown in Figure 6-3.

Note: The dialogs illustrated here reflect a PC running Windows® XP. The dialogs may vary slightly based on the operating system you are using.

FIGURE 6-3:   **MASTER VOLUME CONTROL**
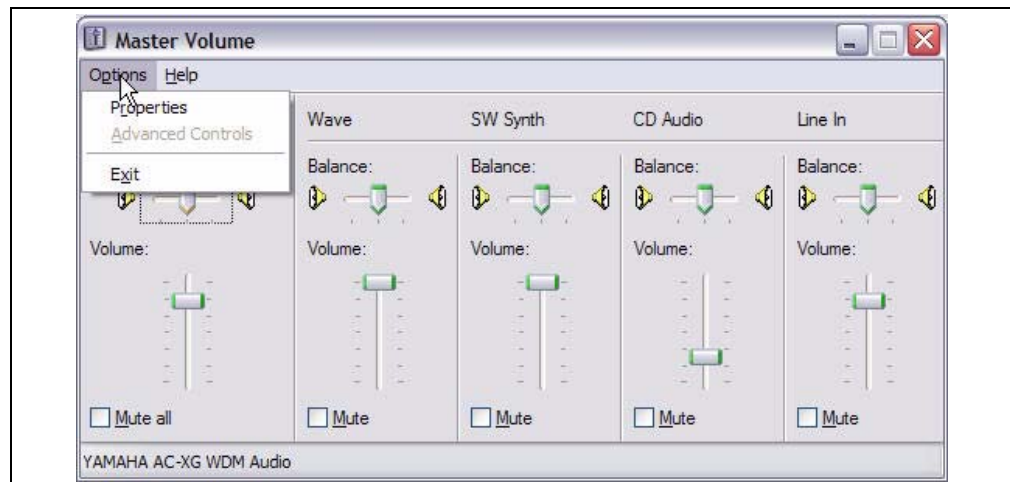
2. On the Properties dialog (Figure 6-4), select **Adjust volume for Recording** and **Microphone**, then click **OK**.

FIGURE 6-4: MASTER VOLUME PROPERTIES DIALOG
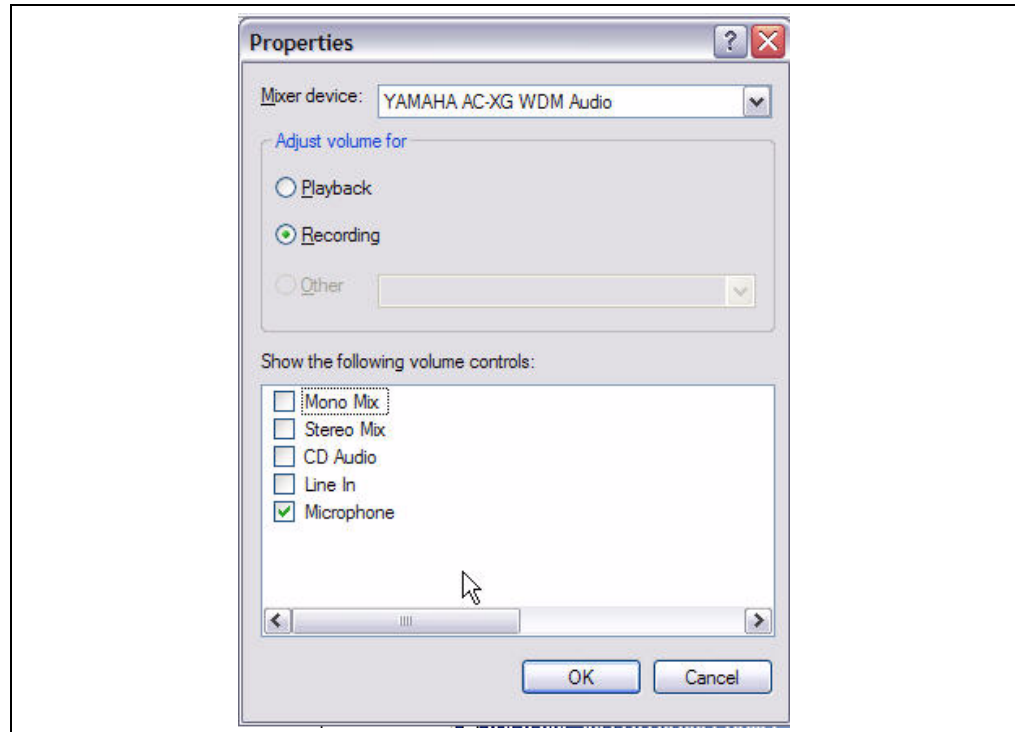
3. When the Recording Control dialog displays the microphone volume controls, adjust the settings for your environment.

FIGURE 6-5: RECORDING CONTROL DIALOG
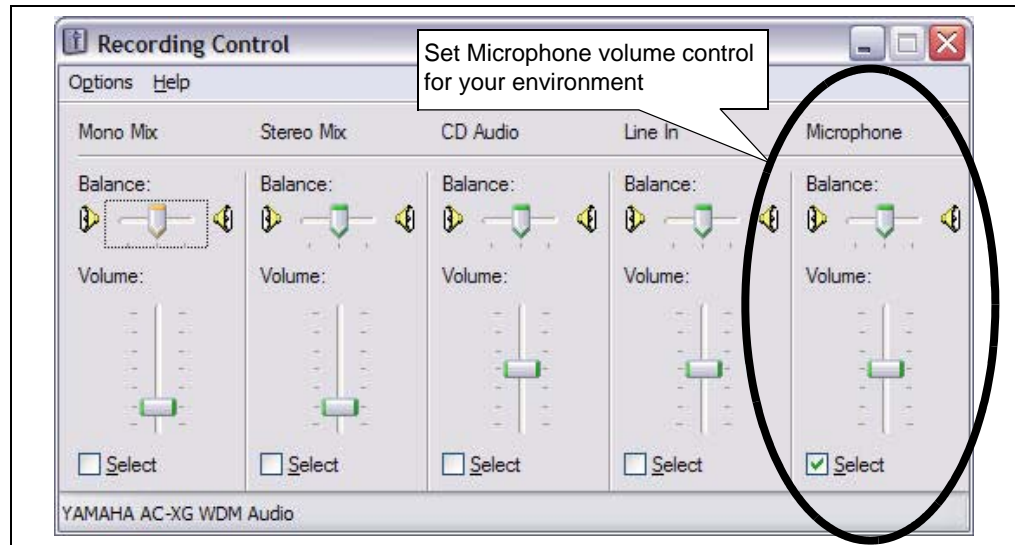
Configure Speech Encoding Utility:

1. Launch the speech encoding utility from the desktop icon or Quick Start menu set up in the installation process. If you choose not to install the icons, navigate to the dsPIC DSC Speech Encoding/Decoding Libraries installation folder and launch the `dsPICSpeechRecord.exe` file. The program window displays the current encoder settings, as shown in Figure 6-6.

FIGURE 6-6:   SPEECH ENCODING UTILITY
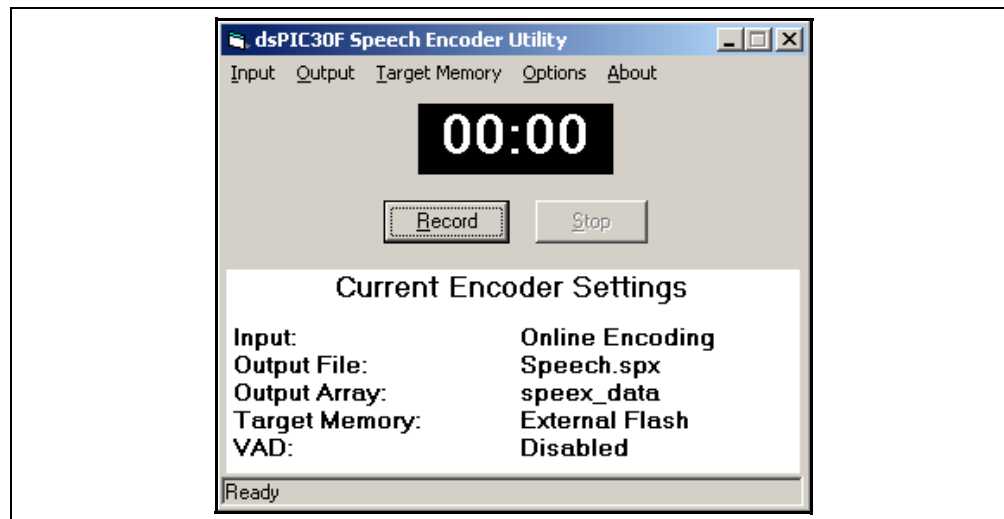
2. Select *Input>Mic*.
3. Select *Output>Array Name*. When the Array name dialog appears, as shown in Figure 6-7. Click **OK** to accept the default array name (speex_data).

FIGURE 6-7:   ARRAY NAME DIALOG

4. Select *Output>Filename*. When the Save As dialog appears, designate the file name and location.
5. From the Target Memory menu, select the type of memory you want to use (see Table 6-1).

TABLE 6-1:   TARGET MEMORY MENU FUNCTIONS

Memory Type	Encoded File Characteristics
Data EEPROM	Generate a "C" source file (`*.c`) to be stored in data EEPROM
External Flash	Generate an assembly source file (`*.s`) to be stored in external Flash memory
Program Memory	Generate an assembly source file (`*.s`) to be stored in program memory
RAM	Generate a "C" source file (`*.c`) to be stored in RAM

6. Select *Options>VAD*, to apply additional compression to voids (silent periods) in the speech file.

Record Your Message:

1. Click **Record** and speak into the microphone. Observe the time being used.

> **Note:** The speech encoding utility has no knowledge about the available memory in your system. You must ensure that the generated source file will fit within your application memory constraints. For instance, the data EEPROM on the dsPIC30F6014A is 4096 bytes, which can store approximately four seconds of Speex-encoded speech.

2. Click **Stop** to stop the recording. An Encoding Completed message displays the properties of the three output files generated, as shown in Figure 6-8.

FIGURE 6-8: ENCODING COMPLETE MESSAGE
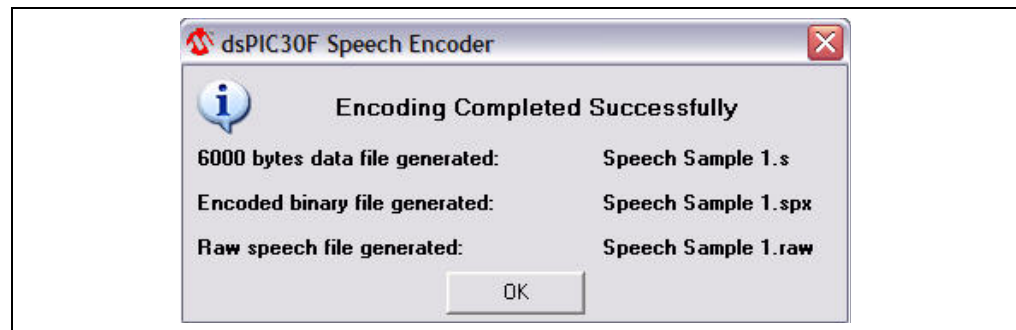
6.4 ENCODING SPEECH FROM A WAVE (`.wav`) FILE

To encode speech from a WAVE (`.wav`) file:

1. Select *Input>Speech File*.
2. Select the output filename and array name from the Output menu.
3. Select the target memory from the Target Memory menu (see Table 6-1).
4. Enable or disable VAD from the Options menu.
5. Press the **Encode** button.
6. Select the WAVE (`.wav`) file to encode.

> **Note:** You must ensure that the source WAVE (`.wav`) file has compatible characteristics. An incompatible format will generate an error message, as shown in Figure 6-9.

FIGURE 6-9: WAVE (`.wav`) FILE FORMAT ERROR MESSAGE
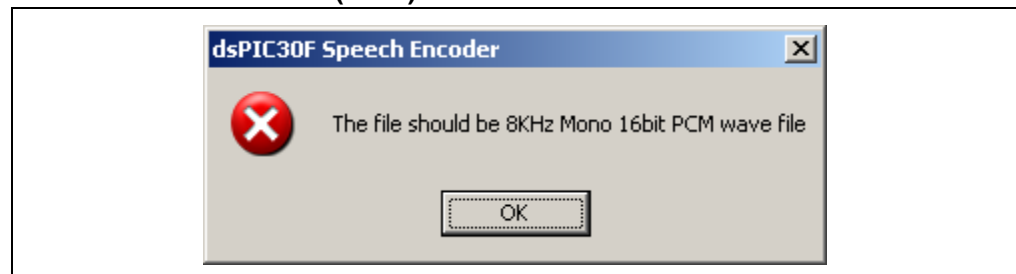
If everything is successful, an Encoding Completed message displays the properties of the three output files generated, as shown in Figure 6-8.

6.5   RECOMMENDATIONS FOR ENCODING FROM A MICROPHONE

When making encoded speech files from a microphone, it is recommended that you speak as clearly as possible in your natural tone of voice. The encoder is not language specific, so any language can be used with the speech encoding utility.

A wide variety of low-cost PC microphones are available in the marketplace. If you are not satisfied with the quality of the playback of the encoded file, try a different microphone. Testing at Microchip with the LabTec Axis301 headset microphone has demonstrated good results across a cross section of speakers.

6.6   USING THE COMMAND LINE DECODER

The encoded data available in the C file array is also available in the binary format in `.spx` file. The encoded data can be decoded using the decoder application file (`AWSpeexDec.exe`) in the speech encoding utility home directory. The decoded data will be in raw format. You may want to play a raw file to assess the quality of the Speex Codec or to use it for the PESQ based MOS evaluation. Since a decoded raw file is not a WAVE (`.wav`) file (it does not have headers), you will need an audio editing utility (such as CoolEdit) to listen to the file.

The decoder usage is:

```
decoder sourcefilename destinationfilename
```

The source file is the generated `.spx` file. The destination file is the raw file produced as a result of decoding. This file will have the extension that you give it (`.raw` is recommended).

Chapter 7. Using Flash Memory for Speech Playback

The Speex Speech Encoding/Decoding Library supports an external memory interface for dsPIC30F devices, which can be used to store real-time encoded data and/or play back encoded data. This chapter provides information on the use of external Flash memory with the library. Items discussed in this chapter include:

- Using External Flash Memory
- Storing Speech Encoding Utility Data to External Flash Memory
- Building a Loadable Hex File for External Flash Memory
- Programming the Hex File to External Flash Memory
- Running the EFP Utility
- Error Handling
- Other External Solutions

7.1   USING EXTERNAL FLASH MEMORY

For dsPIC30F devices, the Speex Speech Encoding/Decoding Library supports an external memory interface, which can be used to store encoded data and/or play back encoded data. This capability provides a solution for store and playback applications and memory-constrained, playback-only applications. The library includes Flash memory drivers for an AMD29F200B memory device. The AMD29F200B is a popular 5.0V Flash memory with a memory size of 128K x 16-bit and fast programming time.

Although the dsPIC30F does not have a dedicated external bus interface, you can interface to external Flash memory through general purpose I/O pins. A reference design for a 16-bit interface to the AMD29F200B in Word mode is provided in Appendix B. This reference design features a 2x30 header, which conveniently plugs into the top of header J19 of the dsPICDEM™ 1.1 Plus Development Board. The required I/O lines for this interface are shown in Table 7-1.

TABLE 7-1:   PINS USED FOR EXTERNAL MEMORY INTERFACE

dsPIC30F Pin	Application Function
RA6	Control RY/BY pin of the external memory
RA7	Control WE pin of the external memory
RC13	Control LE pin of the control circuitry
RD0-RD15	Transmit address to the external memory Receive data from the external memory
RF7	Control CE pin of the external memory
RF8	Control OE pin of the external memory

The reference design provided in Appendix B and the accompanying utility programming software only supports the lower 64K addresses of AMD29F200B memory. All 16 bits of PORTD are used to address external memory and the 17th address line is tied low. This 64 K word interface can store approximately two minutes of compressed speech. If 128 K words are required, the 17th address bit can be implemented from any unused general purpose I/O pin.

7.1.1 Encoding to External Flash Memory

In the store and playback applications such as voice recorders and answering machines, data encoded by the library must be stored in a manner that enables it to be played back later. All dsPIC30F processors feature Flash program memory. Unfortunately, the on-chip Flash memory is not suitable for storing real-time encoded speech because the processor is forced to stall for up to 2 ms while the Flash program memory is programmed. During this time, the processor is unable to service the DCI or ADC interrupts to sample incoming speech data. Ultimately, the use of on-chip flash memory to store speech would result in many lost frames of speech.

A better approach to the real-time storage issue is to use external Flash memory. The AMD29F200B features a 12 µs programming time (per word), which makes it suitable for use in the Speex library. Since speech frames are encoded to a maximum size of 20 bytes, approximately 120 µs are required to write to the flash every 20 ms. This modest amount of overhead makes the AMD29F200B a good choice for real-time speech storage.

7.1.2 Decoding Speech from Flash Memory

External Flash memory can be used as a memory source for decoding speech. The external Flash memory is useful for store-and-playback applications and playback-only applications that require more memory than is available on-chip.

Using the external Flash for speech playback is straightforward. The library provides the ability to read a block of data from Flash memory and use the data as input to the `libDecoder()` function. Approximately 20 instruction cycles are required to read one word from the external Flash memory.

7.2 STORING SPEECH ENCODING UTILITY DATA TO EXTERNAL FLASH MEMORY

The speech encoding utility (see **Chapter 6. "Speech Encoding Utility"**) can encode speech data from your personal computer and target it for storage in external Flash memory. The output of the speech encoding utility is a source file that contains only encoded speech data. The data in this file must be stored in external Flash memory.

Storing the data file to external Flash memory is a two step process:

• First, the source file must be built into a hex file so that it can be loaded into Flash memory.
• Then, a programmer or programming utility must program the hex file into Flash memory.

7.3 BUILDING A LOADABLE HEX FILE FOR EXTERNAL FLASH MEMORY

The loadable hex file is generated by using the MPLAB C30 Language Tools with an MPLAB IDE project that contains a special linker script file. A standard dsPIC30F linker script file (such as `p30f6014A.gld`) generates a hex file targeted for the dsPIC30F memory. The linker script contains information that creates sections for interrupt vector tables, program memory, data EEPROM and data memory. The memory map of the AMD29F200B Flash memory contains only one section, so you must use a custom linker script with the MPLAB C30 Language Tools to generate a hex file targeted for the AMD29F200B.

The installation directory, `\Speex_PC\ExternalFlashHexMaker`, contains an MPLAB IDE project with the custom linker script file `external_flash.gld`. To use this file, open the MPLAB IDE project titled `External_Flash.mcp`. You will see that the linker script (`external_flash.gld`) is already added to this project, so all you have to do is add the source file created from the speech encoding utility and build the project. Follow the given below steps:

1. Open the `External_Flash` project from MPLAB IDE.

> **Note:** The default path is `C:\Speex v2.0\Speex_PC\PCEU`

2. Locate the speech file that you want to load into Flash memory.
3. Add the assembly file (with `*.s` extension) generated from the speech encoding utility to the project.
4. Build the project.

After the project is built, a hex file, `External_Flash.hex` will be created. This file must be next programmed to external Flash memory.

7.4 PROGRAMMING THE HEX FILE TO EXTERNAL FLASH MEMORY

The Speex Library is distributed with a programming utility that runs on the dsPIC30F device and is capable of in-circuit programming of the external Flash memory. The External Flash Programmer (EFP) utility is distributed in the `\Speex_PC\ExternalFlashProgrammer` folder.

The EFP utility can erase and program the AMD29F200B Flash memory. It interfaces through a UART with a generic terminal program such as Windows HyperTerminal®. Programming is performed by sending a hex file to the EFP utility. The EFP utility parses and processes the hex file and programs the AMD29F200B Flash memory one word at a time. The EFP utility runs on the dsPICDEM 1.1 Plus Development Board, but the software can be tailored to run on your own hardware platform.

7.4.1 Building the EFP Utility

The EFP utility consists of the files shown in Table 7-2. To build the EFP project, follow the given below steps:

1. Launch MPLAB IDE and open the `efp.mcp` project located in the `\Speex_PC\ExternalFlashProgrammer` folder.
2. Build the project by selecting *Project>Build All*.

After the EFP utility is built, it is ready to be downloaded to your target for external Flash memory programming.

TABLE 7-2:   EFP SOURCE FILES

Filename	Purpose
`bin2asc.s`	Binary to ASCII conversion function
`config.c`	dsPIC30F configuration setting definitions
`dspic9600.ht`	HyperTerminal configuration file for 9600 baud
`dspic19200.ht`	HyperTerminal configuration file for 19200 baud
`flash.c`	Flash memory functions
`main.c`	EFP utility executive functions
`parser.c`	Hex file parser functions
`read.s`	Flash memory read function
`uart.c`	UART interface functions
`emp.h`	Header file for constants and type definitions

TABLE 7-2:   EFP SOURCE FILES (CONTINUED)

Filename	Purpose
emp_d.h	Header file for global data
emp_f.h	Header file for function prototypes
emp_m.h	Header file for macros

7.4.2   Modifying the EFP Utility

By default, the EFP utility interfaces with the terminal program via UART2 operating at a baud rate of 19200 with external clock and system frequency of 24.576 MIPS. The maximum baud rate at which you can operate the EFP is 19200. You can run the utility using a slower rate, but this will lengthen the external Flash programming time. You can alter the baud rate and system frequency using the #define statements in the emp.h header file, as shown in Example 7-1.

EXAMPLE 7-1:   #DEFINE STATEMENTS IN THE emp.h HEADER FILE

```
#define BAUD_RATE 192          /* 19200 baud (in hundreds) */
#define CLOCK 61440            /* 6.144MHz (in hundreds) */
#define PLL_MULTIPLY 16        /* PLL setting */
```

If you modify the system clock source or PLL setting, you must also modify the configuration bit setting defined in config.c before rebuilding the project:

```
    _FOSC(CSW_FSCM_OFF & ECIO_PLL16); // use EC with x16 PLL
```

7.4.3   PC UART Software

Windows HyperTerminal is used to transmit the target hex file to the dsPIC30F, enabling it to be programmed to external memory. HyperTerminal must be configured to operate in the following mode:

- Specified baud rate (19200 baud max)
- Eight data bits, no parity bits and one stop bit
- Flow control off

ANSI emulation:

- Echo typed characters locally
- Force incoming data to 7-bit ASCII

Start the HyperTerminal application and set the session as described above, or double click one of the provided configuration files (dspic19200.ht or dspic9600.ht) to set the communication parameters.

> **Note:** Any Windows terminal program that supports ASCII communication can be used to interface with the EFP utility. The maximum baud rate is 19200; however, lower baud rates can be used.

7.5   RUNNING THE EFP UTILITY

After you build the EFP utility for your system, as described in **Section 7.4.1 "Building the EFP Utility"**, you can use it to erase, program and read external Flash memory. The EFP utility distributed with the Speex Library is targeted specifically for the dsPICDEM 1.1 Plus Development Board. You may need to modify the EFP utility if you use your own hardware platform.

7.5.1   Erasing the External Flash

The external Flash memory must be erased before it can be programmed. You can erase external Flash memory from the dsPICDEM 1.1 Plus Development Board by following this procedure:

1. Press and release switch SW4. LEDs 1-4 begin to flash in unison.
2. Press and release switch SW3.
3. Press and release switch SW2.
4. Press and release switch SW1.

The chip erase cycle begins, lasting 2-5 seconds. LEDs 3-4 turn off while LEDs 1-2 remain on momentarily to indicate that the chip is being erased. When the erase cycle completes, LEDs 1-2 turn off and LED4 blinks five times.

You can now program the external Flash memory.

If the erase cycle fails, LED4 blinks continuously. To recover from this situation, reset the dsPIC30F and repeat the erase key sequence.

You can abort an erase cycle after initiating the key sequence by pressing switch SW4 again, while the LEDs are flashing. The EFP utility will return to its idle state. However, once the erase cycle has started (i.e., after you have pressed SW1 in sequence and the LEDs have stopped blinking), you must wait for the erase cycle to complete.

> **Note:** No programming or memory verification can take place once the erase sequence has been started. Always complete or abort the erase sequence before performing other operations.

7.5.2   Programming the External Flash

To program external Flash memory (after it has been erased), send the target hex file generated from the `ExternalFlashHexMaker` project (`External_Flash.hex`) to the EFP utility using Windows HyperTerminal (or other comparable terminal software). Follow this process:

1. Start the HyperTerminal application (using the required settings described in **Section 7.4.3 "PC UART Software"**).
2. Download the target hex file by selecting _Transfer>Send Text File_. As the download begins, you will see the hex file echoed on the HyperTerminal screen. Also, LEDs 1-4 on the dsPICDEM board will randomly light as the hex file loads.
3. After the download has successfully completed, LEDs 1-4 turn off and LED3 blinks five times.

> **Note:** If the programming has failed, LED3 will blink continuously. To recover from this situation, reset the dsPIC30F. Programming will fail if the external Flash memory has not been erased before programming begins.
> During programming, no erasing or verifying of memory can take place. You must wait until programming has completed to perform further operations.

4. Verify the programming as described in **Section 7.5.3 "Verifying the Programming of External Flash"**.

7.5.3   Verifying the Programming of External Flash

It is recommended that you compare the program you load into the external Flash memory with the contents of the hex source file. Pressing switch SW1 causes the EFP utility to read the last programmed memory locations (starting from address 0x0), and transmit them back over the UART to the HyperTerminal.

Use the following procedure to verify a program:

1. From the HyperTerminal, select *Transfer >Capture Text*.
2. On the dsPICDEM 1.1 board, press switch SW1. When the EFP utility detects the switch action, it reads the last programmed memory locations (starting from address 0x0) and transmits them via the UART to the HyperTerminal.
3. When the read operation completes, LED1 blinks five times.
4. From the HyperTerminal, select *Transfer>Capture Text>Stop* to stop the HyperTerminal capture.
5. Compare the data returned from the EFP utility with the contents of the `External_Flash.hex` file.

> **Note:**   If you press switch SW1 before the external Flash memory has been programmed, or after a dsPIC30F Reset, the first 256 words of external memory are read and transmitted. This content may not match the `External_Flash.hex` file.

7.5.4   Reading the External Flash

The EFP utility can be used to read the lower 64 K words of AMD29F200B memory. You may want to use this capability to examine what is stored in the Flash memory.

Use the following procedure to read the external Flash memory and store in a text file:

1. From the HyperTerminal, select *Transfer>Capture Text*.
2. On the dsPICDEM 1.1 Plus board, press switch SW2. When the EFP detects the switch action, it reads the lower 64K words (starting from address 0x0) and transmits them via the UART to the HyperTerminal.
3. When the read operation completes, LED2 blinks five times.
4. From the HyperTerminal *Transfer>Capture Text>Stop* to stop the HyperTerminal capture.

> **Note:**   The data transfer will take several minutes to complete. No other operations can be performed while the EFP utility is reading external memory.

7.6   ERROR HANDLING

The EFP utility presently does not recover from Flash memory program or erase errors. It will continue to process with other errors. If a Flash memory program or erase error occurs, the EFP utility must be reset. Error handling is summarized in Table 7-3.

TABLE 7-3:   EMF ERROR HANDLING

Error	Indication
Flash erase failure	LED4 toggles (blinks) indefinitely
Flash programming failure	LED3 toggles (blinks) indefinitely
Hex record checksum failure	Pin RG15 toggles on each hex record that fails, but processing continues
UART Receive Error (framing or overflow)	LED1 lights, but processing continues

7.7   OTHER EXTERNAL SOLUTIONS

The Speex Library includes drivers for interfacing with an AMD29F200B Flash memory. However, you can use any external memory that satisfies your application's requirements. Serial EEPROMs, byte-wide nonvolatile memories or other 16-bit nonvolatile memories can integrate with the library. Important memory selection considerations are device programming time and device read time.

To use an alternate external memory solution, your own set of drivers must be written and used in place of the drivers provided with the library. For detailed information on real-time interfacing with the library, review the guidelines in **Chapter 4. "Integrating Speech Encoding Into YourApplication"** and **Chapter 5. "Integrating Speech Decoding Into YourApplication"**.

NOTES:

Chapter 8. Speech Coding Demos

This chapter describes the demo applications included with the G.711, G.726A and Speex library packages. The intent of these sample applications is to demonstrate how the libraries can be integrated into some typical application types. These sample applications also help familiarize users with the library API.

The G.711 and G.726A libraries include three demos: Communication, Loopback and Playback. The Speex library also includes two demos: Communication and Playback. Each demo application is provided for both dsPIC30F and dsPIC33F devices. Topics covered include:

- Communication Demo
- Loopback Demo
- Playback Demo

8.1   COMMUNICATION DEMO

The Communication demo is installed in the following folders:

`...\G711_dsPIC30F\demo\Communication`

`...\G711_dsPIC33F\demo\Communication`

`...\G726A_dsPIC30F\demo\Communication`

`...\G726A_dsPIC33F\demo\Communication`

`...\Speex_dsPIC30F\demo\Communication`

`...\Speex_dsPIC33F\demo\Communication`

This demo represents a typical full-duplex communication application (e.g., a pair of walkie-talkie units). Two dsPICDEM 1.1 Plus development boards (DM300024) are used as the hardware platform for the demo, with each board representing a node in the communication link.

The communication channel is modeled using an RS-232 connection between the two boards. Both UART modules in each dsPIC device are used: one for synchronizing the sending and receiving of blocks of data, and the other for transferring the actual data.

The on-board Si3000 Voiceband Codec is used as the speech sampling and playback interface, with the DCI module in each dsPIC device used to communicate with the codec.

On each dsPIC, the raw speech samples from the microphone, or obtained through the Si3000 codec and DCI module are encoded and transmitted to the other dsPIC via the RS-232 communication link. Encoded data received via the RS-232 link is decoded and played on the speaker through the DCI module and Si3000 codec. The communication demo is illustrated in Figure 8-1.

FIGURE 8-1:   **COMMUNICATION DEMO**
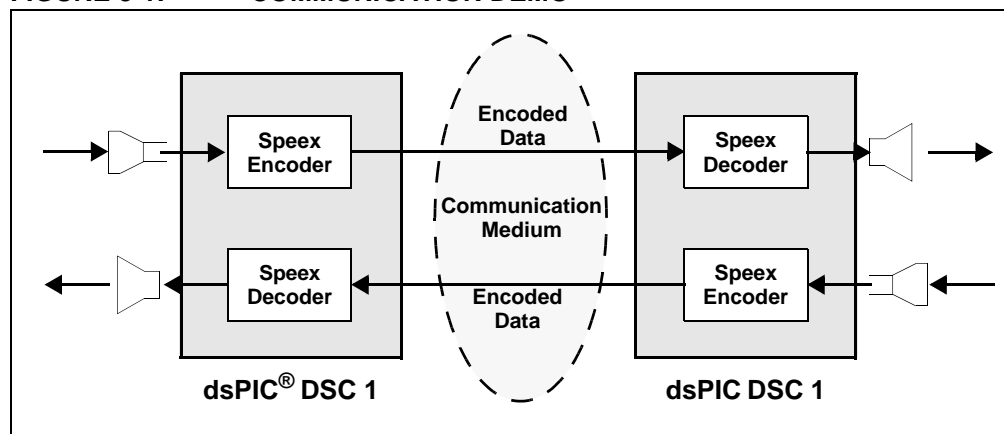
To set up and run the G711 Communication demo, perform the steps as shown below. The G.726A and Speex Communication demos use a similar procedure.

- Open the `G711Communication_30f.mcw` or `G711Communication_33f.mcw` workspace using MPLAB IDE.
- In the `G711Lib_common.h` include file, set the defined value of the `INITIATOR` constant to 1. Program one device.
- Set the `INITIATOR` constant to '`0`', and program the other device.
- Connect the RS-232 Port A of one dsPICDEM 1.1 Plus board to the RS-232 Port A of the other board.
- Similarly, interconnect Port B of the two boards.
- Connect a microphone to the MIC IN port of each dsPICDEM 1.1 Plus board.
- Connect a speaker to the SPKR OUT port of each dsPICDEM 1.1 Plus board.
- Run the program on both the dsPIC devices. Two people can now use the demo. Each person can speak into their microphone and can hear the other person on their speaker.

8.2   LOOPBACK DEMO

The Loopback demo is installed in the following folders:

`...\G711_dsPIC30F\demo\Loopback`

`...\G711_dsPIC33F\demo\Loopback`

`...\G726A_dsPIC30F\demo\Loopback`

`...\G726A_dsPIC33F\demo\Loopback`

This demo represents a full-duplex application, but the one not involving communication. Only one dsPICDEM 1.1 Plus development board is used as the hardware platform for this demo and no communication link is needed.

The on-board Si3000 Voiceband Codec is used as the speech sampling and playback interface, with the DCI module in the dsPIC device used to communicate with the codec.

The raw speech samples from the microphone, or obtained through the Si3000 codec and DCI module are encoded. This encoded data is immediately decoded and played on the speaker through the DCI module and Si3000 codec.

Essentially, the microphone signal is looped back to the speaker, but only after it has been encoded and decoded. The demo is illustrated in Figure 8-2.

FIGURE 8-2:   LOOPBACK DEMO
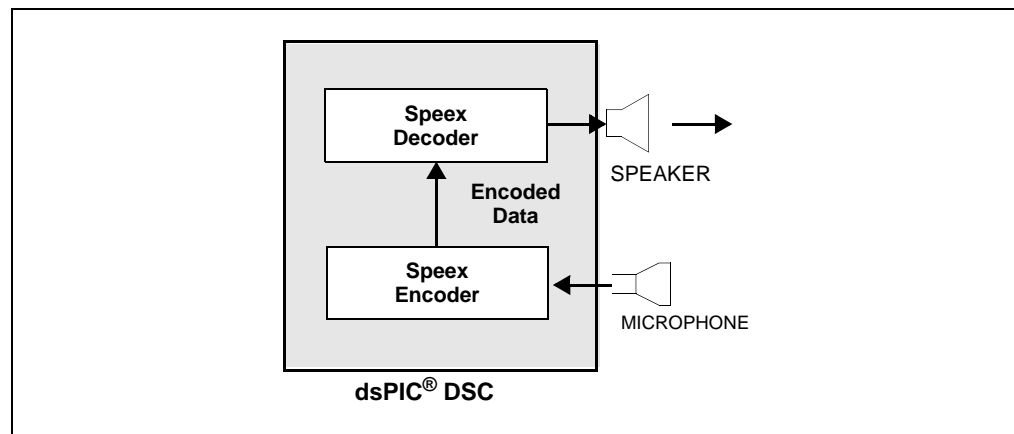
To set up and run the G711 Loopback demo, perform the steps as shown below. The G.726A Loopback demo uses a similar procedure. There is no Speex Loopback demo at this time.

- Open the `G711Loopback_30f.mcw` or `G711Loopback_33f.mcw` workspace using MPLAB IDE.
- Connect a microphone to the MIC IN port of the dsPICDEM 1.1 Plus board.
- Connect a speaker to the SPKR OUT port of the dsPICDEM 1.1 Plus board.
- Run the program on the dsPIC device. You can speak into the microphone and hear your own speech on the speaker. Notice the lack of degradation in the speech quality even after the encoding-decoding process.

dsPIC® DSC Speech Coding Solutions User's Guide

8.3 PLAYBACK DEMO

The Playback demo is installed in the following folders:

`...\G711_dsPIC30F\demo\Playback`

`...\G711_dsPIC33F\demo\Playback`

`...\G726A_dsPIC30F\demo\Playback`

`...\G726A_dsPIC33F\demo\Playback`

`...\Speex_dsPIC30F\demo\Playback`

`...\Speex_dsPIC33F\demo\Playback`

This demo represents a typical simplex application in which only the decoder is used (e.g., a talking toy or a security alarm). A single dsPICDEM 1.1 Plus development board is used as the hardware platform for the demo.

The on-board Si3000 Voiceband Codec is used as the speech sampling and playback interface, with the DCI module in the dsPIC device used to communicate with the codec. The on-chip Program Flash memory in the dsPIC is used to store pre-encoded speech recordings.

On the dsPIC, the pre-encoded data stored in Program Flash memory link is decoded and played on the speaker through the DCI module and Si3000 codec. Users can generate their own pre-encoded data arrays using the Speech Encoding Utility included with the library package. The demo is illustrated in Figure 8-3.

FIGURE 8-3:   PLAYBACK DEMO
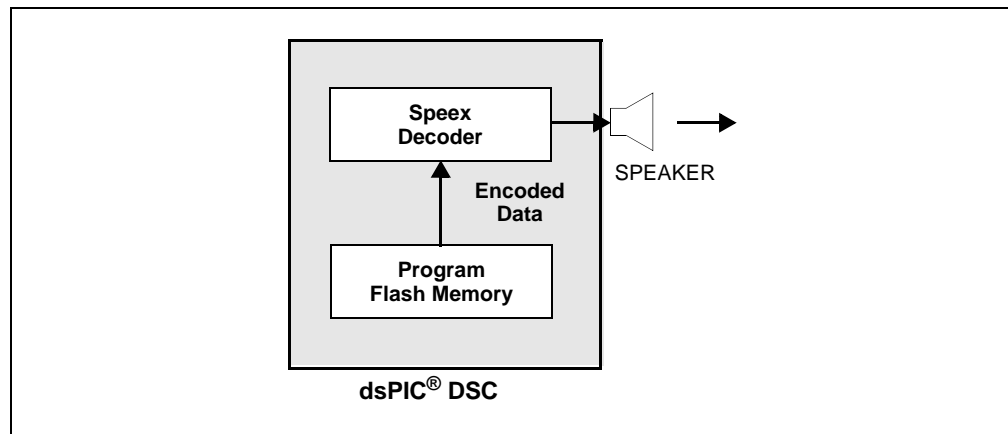
To set up and run the G711 Playback demo, perform the steps below. The G.726A and Speex Playback demos utilize a similar procedure.

- Open the `G711Playback_30f.mcw` or `G711Playback_33f.mcw` workspace using MPLAB IDE.
- Connect a speaker to the SPKR OUT port of each dsPICDEM 1.1 Plus board.
- Run the program on the dsPIC device. You can hear four successive recorded messages output through the speaker.

Appendix A. Si3000 Codec Configuration

A.1   INTRODUCTION

All three of the speech encoding/decoding library packages contain include files and demos that natively support the Silicon Labs Si3000 Voiceband Codec. When the Si3000 codec is used with the library, it must be initialized. Initialization consists of resetting the codec and programming its internal control registers. This section discusses the default configuration used by the library and how you can modify the configuration for your own system requirements.

> **Note:** For detailed information on the Si3000 codec, refer to the latest version of Silicon Laboratories Publication Si3000-DS11 (**Si3000 Voiceband Codec with Microphone/Speaker Drive**).

A.2   DEFAULT CONFIGURATION

The Si3000 configuration can be set in the `spxlib_Si3000.h` header file. The default configuration is shown in Table A-1.

TABLE A-1:   DEFAULT Si3000 CONTROL REGISTER SETTINGS

Register	dsPIC Master Setting	dsPIC Slave Setting	Comments
Control 1	0x10	0x10	Speaker drive active Mic bias selected
Control 2	0x0	0x0	Loopback enabled High-pass filter enabled
PLL1 Divide N1	0x0	0x2	Slave setting for external clock of 6.144 MHz (for 8 kHz sampling)
PLL1 Multiply M1	0x0	0x13	Slave setting for external clock of 6.144 MHz (for 8 kHz sampling)
RX Gain Control 1	0xEA	0xEA	Line input muted Mic gain 10 dB Handset input muted
ADC Volume Control	0x5C	0x5C	RX gain 0 dB Line out muted Handset out muted
DAC Volume Control	0x5F	0x5F	TX gain 0 dB Speaker left active Speaker right active
Status Report	0x0	0x0	Read-only register
Analog Attenuation	0x0	0x0	Line out 0 dB attenuation Speaker out 0 dB attenuation

By default, the dsPIC30F/dsPIC33F is the codec clock master, and is set by the DCIMODE symbol:

```
#define DCIMODE 1   // dsPIC30F clock master
```

The spxlib_common.h file defines several symbols, which must be set correctly for Si3000 operation when the dsPIC is the clock master. The value assigned to FCY automatically sets the BCG1 value of the DCI to produce the correct bit rate clock for 8 kHz sampling. Refer to Table 3-2 for valid system operating frequencies when the dsPIC is the clock master.

```
#define Fcy 24576000L                // Device instruction rate
#define Fs 8000L                     // Speech sampling rate in Hz
#define FSCKD (Fs * 256)             // DCI frame clock rate
#define BCG1 (( FCY / (2*FSCKD) ) - 1)   // DCI bit clock control bits
```

> **Note:** Setting FCY to 4.096 MHz results in a BCG1 value of '0', which disables the DCI. To run the decoder at 4.096 MHz, the dsPIC30F must be the clock slave.

A.3   SETTING THE dsPIC DSC AS A CLOCK SLAVE

If you want to operate your application using the Si3000 codec at an operating frequency different than those shown in Table 3-2, you will need to run the dsPIC30F as the DCI slave. To make the dsPIC30F the codec clock slave, set the DCIMODE symbol to '0'.

```
#define DCIMODE 0                    // dsPIC30F clock slave
```

When the dsPIC is the clock master, the dsPIC provides the frame sync and serial bit clocks to the Si3000 codec. However, when the dsPIC30F is the clock slave, the Si3000 generates the frame sync and serial bit clocks, and these signals are now inputs to the dsPIC.

To use the dsPIC as the clock slave (#define DCIMODE 0) on the dsPICDEM 1.1 Development Board, socket U6 must be populated with a clock oscillator. This clock oscillator is the clock input to the Si3000's PLL. The values for the PLL1 Divide N1 and PLL1 Multiply M1 must be set as described in the Si3000 Data Sheet to yield the required 8 kHz sample rate for your chosen clock oscillator. By default, these registers are set to work with an external 6.144 MHz clock (see Table A-1).

> **Note:** When using this mode on the dsPICDEM 1.1 Plus Development Board, move jumper J9 to the MASTER setting. This indicates that the Si3000 is the clock master.

A.4   MODIFYING THE CODEC GAIN AND VOLUME CONTROLS

The default Si3000 control register settings used by the library (as shown in Table A-1) may not be suitable for your application requirements. For example, you may need a louder output signal for speech playback or a softer input signal for speech encoding.

The following set of #define statements are provided for reference only, and are not contained in the distributed source files. If you wish to use them, you must add these symbols to the spxlib_Si3000.h header file. These symbols can be defined to set the DACVOLUMECONTROL in steps of 3 dB, as shown in Example A-1.

EXAMPLE A-1:   #define STATEMENT

```
#define DV_12_DB          0x007F     /* 12dB DAC volume gain */
#define DV_9_DB           0x0077     /* 9dB DAC volume gain */
#define DV_6_DB           0x006F     /* 6dB DAC volume gain */
#define DV_3_DB           0x0067     /* 3dB DAC volume gain */
#define DV_0_DB           0x005F     /* 0dB DAC volume gain */
#define DV_MINUS_3_DB     0x0057     /* -3dB DAC volume gain */
#define DV_MINUS_6_DB     0x004F     /* -6dB DAC volume gain */
#define DV_MINUS_9_DB     0x0047     /* -9dB DAC volume gain */
#define DV_MINUS_12_DB    0x003F     /* -12dB DAC volume gain */
```

To set the RXGAINCONTROL (only adjustable in steps of 10 dB), you can define the symbols, as shown in Example A-2.

Example A-2:

```
#define MIC_GAIN_30_DB    0x007A     /* 30dB MIC gain */
#define MIC_GAIN_20_DB    0x0072     /* 20dB MIC gain */
#define MIC_GAIN_10_DB    0x006A     /* 10dB MIC gain */
#define MIC_GAIN_0_DB     0x0062     /* 0dB MIC gain */
```

To set the ADCVOLUMECONTROL in steps of 3 dB, you can define the symbols, as shown in Example A-3.

EXAMPLE A-3:

```
#define AV_12_DB          0x007C     /* 12dB ADC volume gain */
#define AV_9_DB           0x0074     /* 9dB ADC volume gain */
#define AV_6_DB           0x006C     /* 6dB ADC volume gain */
#define AV_3_DB           0x0064     /* 3dB ADC volume gain */
#define AV_0_DB           0x005C     /* 0dB ADC volume gain */
#define AV_MINUS_3_DB     0x0054     /* -3dB ADC volume gain */
#define AV_MINUS_6_DB     0x004C     /* -6 dB ADC volume gain */
#define AV_MINUS_9_DB     0x0044     /* -9dB ADC volume gain */
#define AV_MINUS_12_DB    0x003C     /* -12dB ADC volume gain */
```

NOTES:

Appendix B. External Flash Memory Reference Design

B.1   OVERVIEW

This appendix provides a reference design for a 16-bit interface to the AMD29F200B Flash memory device for operation in Word mode. This design features a 2x30 header, which conveniently plugs into the top of header J19 of the dsPICDEM 1.1 Plus Development Board. The required I/O lines for this interface are shown in Table B-1.

TABLE B-1:   PINS USED FOR EXTERNAL MEMORY INTERFACE

dsPIC30F Pin	Application Function
RA6	Control RY/BY pin of the external memory
RA7	Control WE pin of the external memory
RC13	Control LE pin of the control circuitry
RD0-RD15	Transmit address to the external memory Receive data from the external memory
RF7	Control CE pin of the external memory
RF8	Control OE pin of the external memory

This design and the utility programming software included with the Speex Speech Encoding/Decoding Library only supports the lower 64K addresses of AMD29F200B memory. All 16 bits of PORTD are used to address external memory, and the 17th address line is tied low.

This 64K word interface can store approximately two minutes of compressed speech. If 128K words are required, the 17th address bit can be implemented from any unused general purpose I/O pin.

See **Chapter 7. "Using Flash Memory for Speech Playback"** for operational information.

> **Note:** This circuit is only applicable to the dsPIC30F. It is not applicable to the dsPIC33F.

FIGURE B-1: **EXTERNAL MEMORY INTERFACE SCHEMATIC**

Index

![Microchip logo]

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/05/10