MICROCHIP

dsPIC30F Speech Encoding/Decoding Library User's Guide

Table of Contents

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 preface contains general information that is useful to know before you begin using the dsPIC30F Speech Encoding/Decoding Library. Items discussed include:

- Introduction
- 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 dsPIC30F Speech Encoding/Decoding Library 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 dsPIC30F Speech Encoding/Decoding Library and outlines its salient features.
- **Chapter 2. Installation** – This chapter provides detailed instructions for installing the dsPIC30F Speech Encoding/Decoding Library on your PC and setting it up to run with MPLAB IDE.
- **Chapter 3. Application Programming Interface** – This chapter provides information needed to interface the dsPIC30F Speech Encoding/Decoding Library with your user application.

- **Chapter 4. Incorporating Speech Encoding** – This chapter provides information to help you understand how to integrate the speech encoding portion of the dsPIC30F Speech Encoding/Decoding Library into your application and how to build with the library.
- **Chapter 5. Incorporating Speech Decoding** – This chapter provides information to help you understand how to integrate the speech decoding portion of the dsPIC30F Speech Encoding/Decoding Library 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 dsPIC30F Speech Encoding/Decoding Library 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 Encoding Sample Application** – This chapter describes a sample application that demonstrates stand-alone speech encoding and playback from on-chip data EEPROM memory.
- **Chapter 9. Speech Decoding Sample Application** – This chapter describes a sample application that demonstrates stand-alone speech playback from on-chip program 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 non-volatile memory.
- **Appendix C. ADC/PWM Interface Reference Design** – This appendix provides schematics for circuitry that provides speech sampling through the Analog-to-Digital Converter and speech playback through the PWM.
- **Appendix D. Sample Applications** – This appendix provides the code for the sample encoding and decoding applications.

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	<u>*File>Save*</u>
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) {... }`

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 dsPIC30F Speech Encoding/Decoding Library. The following Microchip documents are available and recommended as supplemental reference resources.

dsPIC30F Family Reference Manual (DS70046)

Consult this document for detailed information on dsPIC30F device operation. This reference manual explains the operation of the dsPIC30F MCU 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.

dsPIC30F Programmer’s Reference Manual (DS70030)

This manual is a software developer’s reference for the dsPIC30F 16-bit MCU family of devices. This manual describes the instruction set in detail and also provides general information to assist you in developing software for the dsPIC30F MCU family.

dsPIC30F Family Overview (DS70043)

This document provides an overview of the functionality of the dsPIC® DSC product family. Its purpose is to help you determine how the dsPIC 16-bit Digital Signal Controller Family fits your specific product application. This document is a supplement to the *dsPIC30F Family Reference Manual*.

MPLAB® ASM30, MPLAB® LINK30 and Utilities 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 ASM30 Assembler
- MPLAB LINK30 Linker
- MPLAB LIB30 Archiver/Librarian
- Other Utilities

MPLAB® C30 C Compiler User’s Guide and Libraries (DS51284)

This document helps you use Microchip’s MPLAB C30 C compiler for dsPIC DSC devices to develop your application. MPLAB C30 is a GNU-based language tool, based on source code from the Free Software Foundation (FSF). For more information about the FSF, see www.fsf.org.

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

Consult this document for more 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 references 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, 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 receive e-mail notifications when 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.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support
- Development Systems Information Line

Customers should contact their distributor, representative or field application engineer (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 the web site at: http://support.microchip.com

In addition, there is a Development Systems Information Line which lists the latest versions of Microchip's development systems software products. This line also provides information on how customers can receive currently available upgrade kits.

The Development Systems Information Line numbers are:

1-800-755-2345 – United States and most of Canada

1-480-792-7302 – Other International Locations

DOCUMENT REVISION HISTORY

Revision A (May 2005)

- Initial Release of this Document.

Chapter 1. Overview

1.1 INTRODUCTION

The dsPIC30F Speech Encoding/Decoding Library is a development tool that provides toll-quality voice compression and decompression to help you generate speech-based embedded applications on the dsPIC30F family of digital signal controllers. This chapter provides an overview of the library.

1.2 HIGHLIGHTS

Information in this chapter includes:

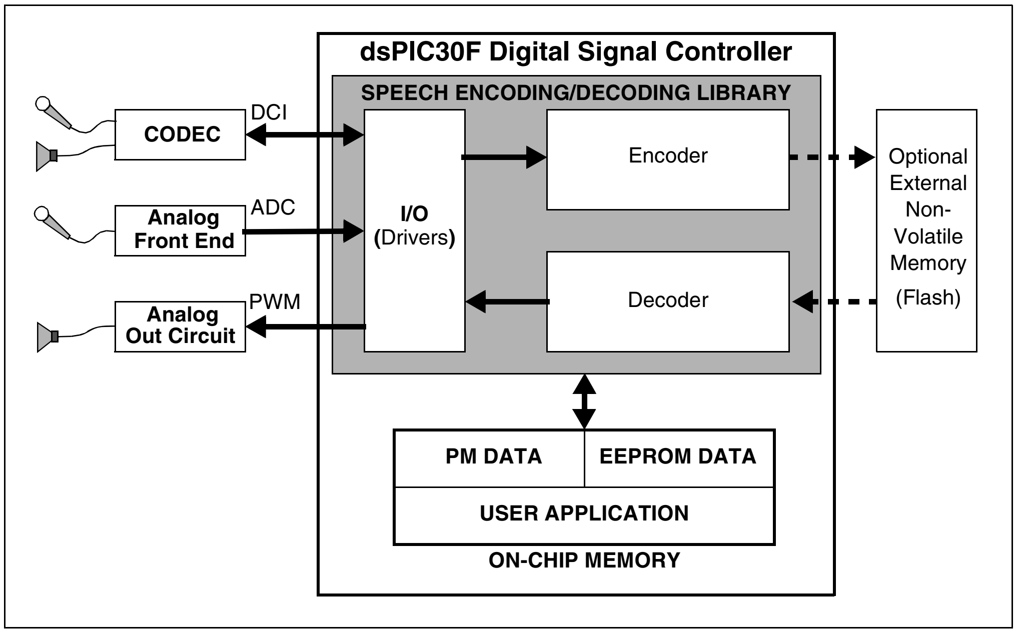
- Overview
- Features
- Resource Requirements

1.3 OVERVIEW

The Speech Encoding/Decoding Library is based on the Speex speech coder and designed specifically for the dsPIC30F family of Digital Signal Controllers. The library samples speech at 8 kHz and compresses it to a rate of 8 kbps featuring a 16:1 compression ratio. Encoding 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. 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: SPEECH CODING/DECODING LIBRARY OVERVIEW



The flexible analog interface gives your design several options to consider. The speech encoder can use either an external codec or the on-chip 12-bit analog-to-digital converter to sample speech input. The speech decoder can play decoded speech through an external codec or the on-chip pulse width modulator. An optional Voice Activity Detection feature enhances compression by detecting voids in the incoming speech and compressing them at a higher ratio.

Predominantly written in assembly language, the Speech Encoding/Decoding Library optimizes computational performance and minimizes RAM usage. A well-defined API (**Chapter 3. "Application Programming Interface"**) makes it easy to integrate with your application.

Playback-only applications will benefit from the PC-based speech encoding utility (**Chapter 6. "Speech Encoding Utility"**), which allows you to produce encoded speech files from your desktop using a microphone or an existing WAV file. Encoded speech files are added to your MPLAB® IDE project, just like a regular source file, and built into your application. The speech encoding utility allows you to select four target memory areas for your speech file: program memory, data EEPROM, RAM and external flash memory. External flash memory allows you to store several minutes of speech (1 minute of speech requires 60 KB), and it is supported through a dsPIC30F general purpose I/O port.

1.4 FEATURES

The library is appropriate for half-duplex systems such as answering machines, intercoms and walkie-talkies. With the decoder's small footprint, the library is also ideal for playback-only applications, such as building safety systems and smart appliances. Salient features of the dsPIC30F Speech Encoding/Decoding Library include:

- Fixed 8 kHz sample rate
- Fixed 8 kbps output rate
- MOS: 3.7 - 4.2 (based on PESQ testing)
- Code Excited Linear Prediction based coding
- Two analog input interfaces - codec or on-chip ADC
- Two analog output interfaces - codec or on-chip PWM
- Optional Voice Activity Detection
- Windows-based utility allows you to create your own encoded files for playback
- Compressed speech files require 1 Kbyte for each second of speech
- Off-chip support allows playback of long speech samples
- Royalty free
- Fully compliant with Microchip MPLAB C30 Language Tools
- Includes complete User's Guide
- Designed to run on dsPICDEM™ 1.1 General Purpose Development Board (DM300014)

1.5 RESOURCE REQUIREMENTS

The dsPIC30F Speech Encoding/Decoding Library requires the resources listed in Table 1-1.

TABLE 1-1: RESOURCE REQUIREMENTS

Resource	Requirement
Decoder	
Playback Interface	Si3000 Audio Codec or PWM
Computational Power	3 MIPs
Program Flash Memory	15 KB
RAM*	3.2 KB
Encoder	
Sampling Interface	Si3000 Audio Codec or 12-bit ADC
Computational Power	19 MIPs (worst case)
Program Flash Memory	33 KB
RAM*	5.4 KB (1.2 KB is scratch)

* Full-duplex support is presently not possible due to RAM requirements. Half-duplex support is possible.

1.5.1 Devices Supported

- dsPIC30F5011
- dsPIC30F5013
- dsPIC30F6011
- dsPIC30F6012
- dsPIC30F6013
- dsPIC30F6014

1.5.2 Host Requirements

- PC-compatible system with an Intel® Pentium® class or higher processor, or equivalent.
- RAM - 16 MB minimum
- Hard Drive space - 4 MB minimum
- CD-ROM drive
- Microsoft Windows® 95/98/ME/2000/XP or Windows NT® 4.0

NOTES:

Chapter 2. Installation

2.1 INTRODUCTION

This chapter provides instructions for installing the dsPIC30F Speech Encoding/Decoding Library on your PC.

2.2 HIGHLIGHTS

Information in this chapter includes:

- System Requirements
- Installation Procedure
- Uninstall Procedure

2.3 SYSTEM REQUIREMENTS

2.3.1 Host Requirements

The host system must be a Pentium-class PC with at least 16 MB RAM, 4MB available hard-disk space, CD-ROM drive with Microsoft Windows 95/98/ME/2000/XP or Windows NT 4.0.

2.3.2 Development Tools

You will need the following Microchip development tools:

- MPLAB IDE Integrated Development Environment (download free)
- MPLAB C30 Compiler, version 1.31 or higher
- ICD 2 In Circuit Debugger

2.4 INSTALLATION PROCEDURE

The dsPIC30F Speech Encoding/Decoding Library is packaged on a CD. To install the library follow these steps:

1. Insert the CD into the appropriate drive. When the Setup Wizard displays follow the procedure outlined in Figure 2-1 to accept the license agreement, select an install location and select your program launch preferences.
2. Follow the additional instructions in Figure 2-2 to complete the installation.

By default the dsPIC30F Speech Encoding/Decoding Library files are installed in:

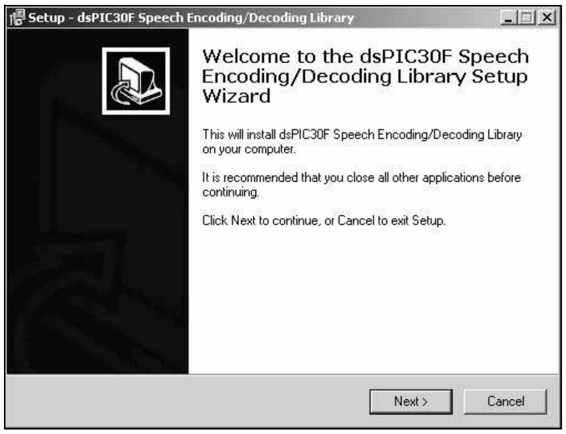
```
c:\program files\microchip\dsPIC30F_SEDLibrary.
```

The completed installation includes these folders:

```
...\dsPIC30F_SEDLibrary\demo\decoder
...\dsPIC30F_SEDLibrary\demo\encoder
...\dsPIC30F_SEDLibrary\doc
...\dsPIC30F_SEDLibrary\include
...\dsPIC30F_SEDLibrary\src
...\dsPIC30F_SEDLibrary\utils\ExternalFlashHexMaker
...\dsPIC30F_SEDLibrary\utils\ExternalFlashProgrammer
...\dsPIC30F_SEDLibrary\utils\SpeechEncodingUtility
```

FIGURE 2-1: INSTALLATION PROCESS WITH SETUP WIZARD

1. When the setup wizard welcome screen displays, click **Next>** to begin installation.



Setup - dsPIC30F Speech Encoding/Decoding Library

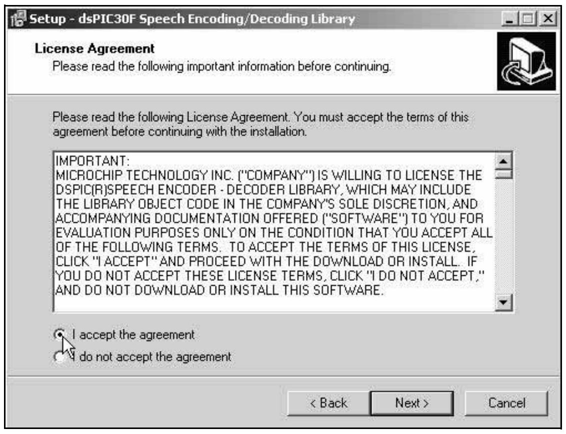
Welcome to the dsPIC30F Speech Encoding/Decoding Library Setup Wizard

This will install dsPIC30F Speech Encoding/Decoding Library on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

2. When the License Agreement screen displays, check "I accept the agreement" to allow the install process to proceed. Then click **Next>** to begin continue.



Setup - dsPIC30F Speech Encoding/Decoding Library

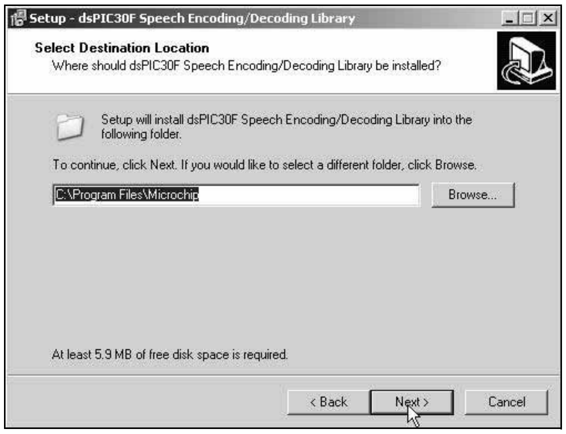
License Agreement
Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

IMPORTANT:
MICROCHIP TECHNOLOGY INC. ("COMPANY") IS WILLING TO LICENSE THE DSPIC(R)SPEECH ENCODER - DECODER LIBRARY, WHICH MAY INCLUDE THE LIBRARY OBJECT CODE IN THE COMPANY'S SOLE DISCRETION, AND ACCOMPANYING DOCUMENTATION OFFERED ("SOFTWARE") TO YOU FOR EVALUATION PURPOSES ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

[✓ I accept the agreement]
I do not accept the agreement

3. When the Select Destination Location screen displays, click **Next>** to begin continue.
   If you prefer to install the library in another location, browse to that location and then click **Next>**.



Setup - dsPIC30F Speech Encoding/Decoding Library

Select Destination Location
Where should dsPIC30F Speech Encoding/Decoding Library be installed?

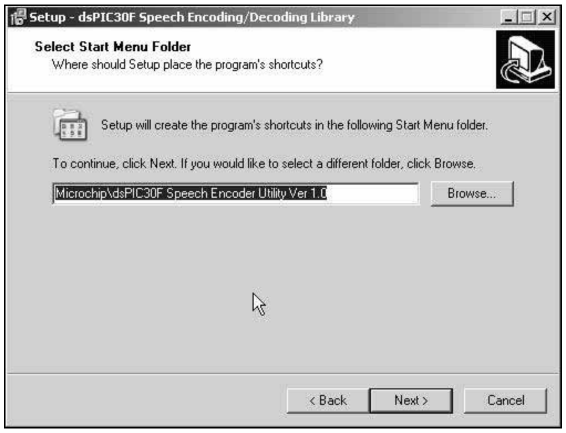
Setup will install dsPIC30F Speech Encoding/Decoding Library into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\Microchip

At least 5.9 MB of free disk space is required.

4. When the Select Start Menu Folder screen displays, click **Next>** to begin continue.
   If you prefer to place the Start menu in another location, browse to that location and then click **Next>**.



Setup - dsPIC30F Speech Encoding/Decoding Library

Select Start Menu Folder
Where should Setup place the program's shortcuts?

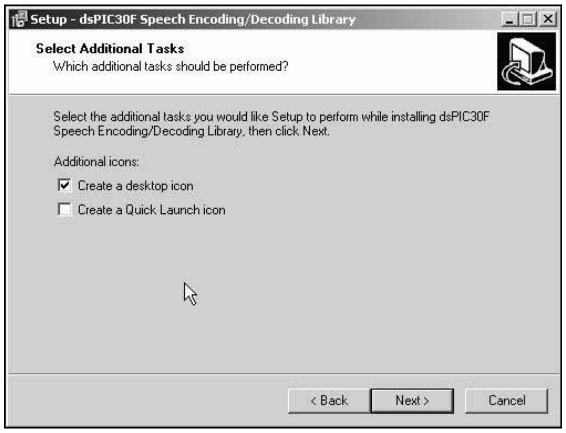
Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

Microchip\dsPIC30F Speech Encoder Utility Ver 1.0

FIGURE 2-2: INSTALLATION COMPLETION PROCESS

5. When the Select Additional Tasks screen displays, decide if you want to install launch icons on your desktop and on the Start menu, then click **Next>**.

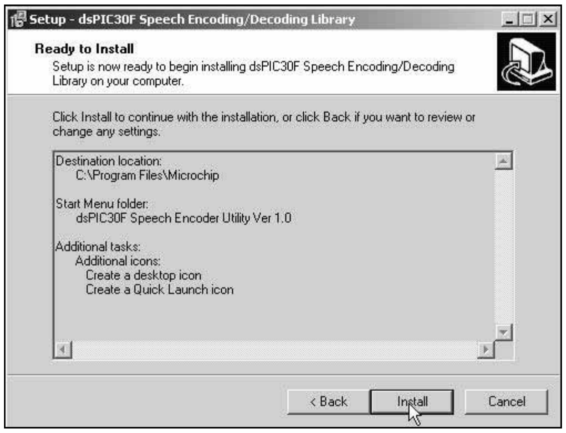


Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing dsPIC30F Speech Encoding/Decoding Library, then click Next.

Additional icons:
- [✓] Create a desktop icon
- [ ] Create a Quick Launch icon

6. When the Ready to Install screen displays, decide if you want to continue installing into the indicated locations, then click **Next>**. To change the location, click **<Back** and repeat step 3.



Ready to Install
Setup is now ready to begin installing dsPIC30F Speech Encoding/Decoding Library on your computer.

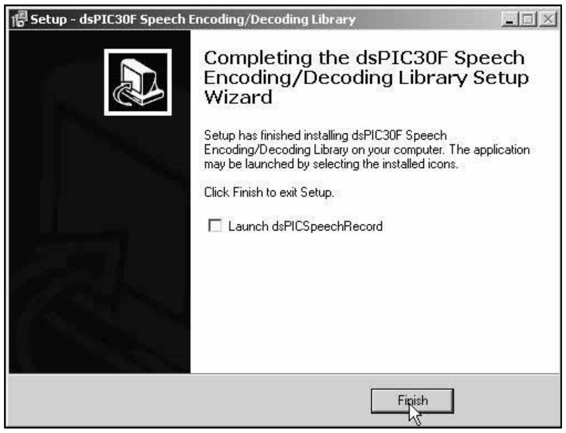
Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
    C:\Program Files\Microchip

Start Menu folder:
    dsPIC30F Speech Encoder Utility Ver 1.0

Additional tasks:
    Additional icons:
        Create a desktop icon
        Create a Quick Launch icon
```

7. When the completion screen displays, decide if you want to install launch icons on your desktop and on the Start menu, then click **Finish**.



Completing the dsPIC30F Speech Encoding/Decoding Library Setup Wizard

Setup has finished installing dsPIC30F Speech Encoding/Decoding Library on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

- [ ] Launch dsPICSpeechRecord

2.5 UNINSTALL PROCEDURE

To uninstall the dsPIC30F Speech Encoding/Decoding Library:

1. From **Start** select *Settings>Control Panel*.
2. From the Control Panel window, select **Add/Remove Programs**.
3. Select **dsPIC30F Speech Encoding/Decoding Library**.
4. Click **Change/Remove**.
5. Follow the Windows uninstall procedures.

NOTES:

Chapter 3. Application Programming Interface

3.1 INTRODUCTION

The dsPIC30F Speech Encoding/Decoding Library integrates with a user application running on the dsPIC30F device to provide support for handling speech in the application. This chapter provides information needed to interface the dsPIC30F Speech Encoding/Decoding Library with your user application.

The application programming interface for the dsPIC30F Speech Encoding/Decoding Library is implemented in these library archive files:

- `libspxdecoderlib.a`
- `libspxencoderlib.a`.

The `libspxdecoderlib.a` library archive contains functions for performing speech decoding and playing back speech through the Si3000 codec via the on-chip Data Converter Interface (DCI) or through the on-chip Pulse Width Modulator (PWM).

The `libspxencoderlib.a` library archive contains functions for performing speech encoding from the Si3000 codec via the DCI or on-chip 12-bit Analog-Digital Converter (ADC). All functions in the library adhere to the Microchip C30 compiler function calling convention.

3.2 HIGHLIGHTS

Items discussed in this chapter are:

- Requirements
- Data Structures
- Library Functions
- Required dsPIC30F System Resources

3.3 REQUIREMENTS

3.3.1 System Frequency Requirements

The dsPIC30F Speech Encoding/Decoding Library requires that speech be sampled and played back at a fixed rate of 8.0 kHz. Speech sampling is performed by an external codec that interfaces with the dsPIC30F via its Data Converter Interface (DCI) module or through the on-chip ADC. When sampling is performed with the DCI as the codec clock master, only a limited number of system frequencies can be used by your application to accommodate the 8.0 kHz sampling rate. The dsPIC30F processor can execute only at multiples of 4.096 MHz. Thus, the allowable execution speeds for the dsPIC30F Speech Encoding/Decoding Library are 8.192 MHz, 12.288 MHz, 16.384 MHz and 24.576 MHz when the dsPIC30F is the DCI clock master.

To accommodate these system frequencies for DCI master mode, operate the dsPIC30F using only the clock and PLL combinations shown in Table 1-1.

TABLE 3-1: CLOCK AND PLL COMBINATIONS FOR 8KHZ SAMPLING RATE (DCI MASTER MODE)

Processor Frequency*	Clock Frequency	PLL Setting
8.192 MIPS	4.096 MHz	8x
12.288 MIPS	6.144 MHz	8x
16.384 MIPS	4.096 MHz	16x
24.576 MIPS	6.144 MHz	16x

* Decoder may run at this frequency, but encoder requires at least 19 MIPS.

To overcome the limitations that the processor frequency imposes on the sampling rate, the DCI can be configured as slave. In this case, the DCI and Si3000 use an external clock. The dsPIC30F Speech Encoding/Decoding Library allows you to configure the DCI as slave or master by providing `#defines` in the `spxlib_si3000.h` file, as shown below:

```
#define DCIMODE   1
```

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

When operating with the alternate sampling/playback interfaces, 12-bit ADC and PWM, there are no restrictions on the system clock frequency.

3.3.2 Memory Requirements:

Memory requirements for the decode process are shown in Table 3-2

TABLE 3-2: SPEEX RAM REQUIREMENTS

Memory Segment	Decoder Requirements (Bytes)	Encoder Requirements (Bytes)	Decoder + Encoder Requirements (Bytes)
Near RAM	0	90	90
X RAM	1846	2418	3424
Y RAM	342	610	952
General RAM	124	122	154
Heap(1)	0	1282	1282
User Defined I/O Vectors(2)	680	680	680
Total	2992	5202	6582

Note 1: Heap memory becomes available when the encoder is not active.

2: User must define four input/output vectors with a total size of 680 bytes.

3.3.3 MIPS Requirements

The encoder requires 18.5 MIPS, worst case, which includes ADC or DCI interrupt service routine activity.

The decoder requires 3 MIPS, worst case, which includes DCI interrupt service routine activity.

When the decoder is run with the PWM interface, the MIPS requirements are as shown in Table 3-3 for PWM frequencies of 16 kHz, 24 kHz and 32 kHz. Due to the nature of the PWM interrupt service routine, the processor must be run at a rate higher than the MIPS required by the decoder. For instance, to operate the PWM at 32 kHz, the device must operate at no less than 18 MIPS, 8.5 MIPS of which will be used by the decoder.

TABLE 3-3: DECODER MIPS REQUIREMENTS WITH PWM

PWM Frequency	MIPS (worst case)	Minimum Processor Frequency (MIPS)
16 kHz	7.9	9.0
24 kHz	8.2	13.5
32 kHz	8.5	18.0

3.3.4 External Hardware Resources

The dsPIC30F Speech Encoding/Decoding Library requires the external hardware listed in Table 3-4.

TABLE 3-4: EXTERNAL HARDWARE REQUIREMENTS

Resource	Hardware Requirement
Audio codec	Si3000 (Silicon Laboratory)
Alternate input interface	ADC signal conditioning circuit
Alternate output interface	PWM signal conditioning circuit
External microphone	Linear frequency response up to 4 kHz (minimum)
Speaker/Headphone	32 ohms
Desktop/laptop computer	P3/Celeron/P4 Processor

3.3.5 Software Requirements

The dsPIC30F Speech Encoding/Decoding Library requires the following software:

- Windows 98/2000/XP
- MPLAB IDE V6.60 or higher
- MPLAB C30 Compiler V1.20.04 or higher

3.4 DATA STRUCTURES

The dsPIC30F Speech Encoding/Decoding Library uses the following data structures to interface with your user application.

3.4.1 spxSi3000 Structure

The `spxSi3000` structure defined in the `spxlib_Si3000.h` file includes all the Si3000 registers as its members. This structure also includes the settings for the DCI peripheral. You initialize the structure with `#define` statements provided in the `spxlib_Si3000.h` file. **Appendix A. "Si3000 Codec Configuration"** contains detailed information about the Si3000 registers.

```
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
   };
```

Once you have set the `#defines` in `spxlib_Si3000.h` for your application, another `#define` 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
...
```

3.4.2 codecsetup Structure

The `codecsetup` structure is defined in the `spxlib_common.h` file. This structure is used to access user defined raw and encoded speech buffers, synchronization flags and speech sample counters used for encoding and decoding. A basic understanding of this structure is required for integrating the dsPIC30F Speech Encoding/Decoding Library with your application. The library is built around two pairs of user-defined I/O buffers, which are referenced by the structure.

```
struct _codecsetup
{
 short         *sampleIpBuffer;     //Pointer to raw Speech
                                    //sample buffer1
 short         *sampleOpBuffer;     //Pointer to raw Speech
                                    //sample buffer2
 volatile char *sampleEncdIpBuffer; //Pointer to encoded speech
                                    //sample buffer1
 volatile char *sampleEncdOpBuffer; //Pointer to encoded speech
                                    //sample buffer2
 volatile char fFramedone;          //Flag for I/O buffer full
                                    //or empty
 volatile char fStartPlay;          //Flag to start or stop
                                    //speech play
 volatile int  frameCount;          //Counter for number of
                                    //frames of data.
 volatile int  countFill;           //Counter for number of
                                    //samples stored.
 volatile int  countLoad;           //Counter for number of
                                    //samples played.
 volatile unsigned long sampleCount;//Counter for number of
                                    //samples
 volatile char numOfencSamplesPerFrame; //Indicates the number
                                    //of encoded samples in
                                    //each frame.
 volatile char fFrameplayed;        //Flag to indicate decoding
                                    //is done.
 volatile char fEncodedone;         //Flag to indicate encoding
                                    //is done
 volatile int  setOfADCData;        //Number of speech samples
                                    //in each ADC result buffer
 volatile unsigned int* AdcBuf0Ptr; //Pointer to ADCBUF0
                                    //register.
 unsigned long arraysizeinbytes;    //Number of bytes in
                                    //encoded speech sample.
 char          vad;                 //Indicates if VAD enabled
                                    //Default is disabled.
 char          lostFrame;           //Indicates lost frame
                                    //Default is no lost frame
 long          recordSize;          //Indicates the size of
                                    //the recorded speech in
                                    //number of frames.
};
```

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

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

3.4.3 PWM Structure

The PWM structure is of data type `_spxPWM`, which is defined in `sxplib_pwm.h`. This structure contains elements for controlling the playback of speech through the dsPIC30F Output Compare module in PWM mode. Using `#defines` in `spxlib_pwm.h`, you can initialize the elements of the PWM structure. The `period_count` and `scalefactor` elements are set by expressions that should not be modified.

Any of the eight Output Compare modules (OC1 - OC8) can be used with the decoder to generate the PWM signal. This selection is made with the `occon` structure element. Either Timer 2 or Timer 3 can be selected as the time base for the PWM time base, and this selection is made with the timer structure element.

For more information about using the PWM, see **Section 5.4 "Using the PWM Alternate Playback Interface"**.

```
struct _spxPWM
{
   int fpwm;                // PWM frequency
   int tmrpresclval;        // Timer prescale setting
   int period_count;        // Timer period (computed)
   int timerxintpri;        // Timer interrupt priority
   float scalefactor;       // Scale factor (computed)
   char timer;              // Timer Selection - 2 or 3
   int count;               // Ratio of (PWM frequency / 8000)
   char occon;              // Output Compare Selection (1 - 8)
};
```

3.4.4 ADC12 Structure

The ADC12 structure is of data type `_spxADC12`, which is defined in `sxplib_adc.h`. This structure contains elements for all the ADC control registers. Using `#defines` in `spxlib_adc.h`, you can initialize the elements of the ADC12 structure to meet your specific requirements.

Refer to **Section 4.4 "Using the ADC Alternate Sampling Interface"** for information on initializing the structure for your specific application.

```
struct _spxADC12
{
   int adccon1val;          // ADC control register 1
   int adccon2val;          // ADC control register 2
   int adccon3val;          // ADC control register 3
   int adchsval;            // ADC input select register
   int adpcfgval;           // ADC port configuration register
   int adcsslval;           // ADC input scan select register
   int spxadcintpri;        // ADC interrupt priority
   int channelsinsequence;  // Number of channels scanned
   int adcbufferlength;     // ADC buffer length
};
```

3.5 LIBRARY FUNCTIONS

The dsPIC30F Speech Encoding/Decoding Library API is implemented in archive files `libspxdecoderlib.a` and `libspxencoderlib.a`.

The `libspxdecoderlib.a` library contains functions for performing speech decoding and playback through the Si3000 codec via the on-chip Data Converter Interface (DCI) or through the on-chip Pulse Width Modulator (PWM).

The `libspxencoderlib.a` library contains functions for performing speech encoding from the Si3000 codec via the DCI or on-chip 12-bit Analog-Digital Converter (ADC). All functions in the library adhere to the Microchip C30 compiler function calling convention.

3.5.1 libADCFillBuffer() Function

Overview

The `libADCFillBuffer()` function is called from the ADC interrupt service routine and performs these tasks:

- Moves data from ADC buffer registers ADCBUF0–ADCBUFF to the user defined speech sample ping-pong buffers. The number of speech samples read is the value of `ADC12.adcbufferlength` divided by the value of `ADC12.channelsinsequence`.
- Increments `codecdata.countFill` to keep count of the number of samples read.

Note: Set the ADC12 buffer length to a value that it is divisible by the number of channels in a sequence of conversions. You may need to modify this function as described in **Section 4.4 “Using the ADC Alternate Sampling Interface”**.

Return Value

None

Parameters

None

3.5.2 libADC12Init() Function

Overview

The `libADC12Init()` function initializes the ADC12 module based on the user defined settings in `spxlib_adc.h`.

Note: This function does not enable interrupts or initiate ADC sampling.

Return Value

None

Parameters

None

3.5.3 libADC12StartSampling() Function

Overview

The `libADC12StartSampling()` function:

- Initializes the four data buffer pointers of `codecdata` with the address of the four user-defined buffers:
  - `codecdata.sampleIpBuffer`
  - `codecdata.sampleOpBuffer`
  - `codecdata.sampleEncdIpBuffer`
  - `codecdata.sampleEncdOpBuffer`
- Clears `codecdata.frameCount`.
- Sets the ADC interrupt.
- Enables the ADC, which starts sampling process.

Return Value

None

Parameters

The `libADC12StartSampling()` function has four parameters, as shown in Table 3-5.

TABLE 3-5: LIBADC12STARTSAMPLING () FUNCTION PARAMETERS:

Parameter	Type	Usage
`buf1`	`(short*)`	Pointer to raw speech sample buffer1 (160 words)
`buf2`	`(short*)`	Pointer to raw speech sample buffer2 (160 words)
`buffer1`	`(char*)`	Pointer to encoded speech sample buffer1 (20 bytes)
`buffer2`	`(char*)`	Pointer to encoded speech sample buffer2 (20 bytes)

3.5.4 libADC12StopSampling() Function

Overview

The `libADC12StopSampling()` function:

- Clears the ADC interrupt flag.
- Disables the ADC interrupt.
- Disables the ADC module to stop sampling.

Return Value

None

Parameters

None

3.5.5 libarrayFillDecoderInputPM() Function

Overview

The `libarrayFillDecoderInputPM()` function:

- Moves one data frame of encoded speech (5 or 20 bytes) from program memory to the user-defined encoded speech data buffer.
- Performs buffer management of the decoder input buffers.
- Increments `codecdata.sampleCount` by the number of bytes read.
- Clears `codecdata.countFill.`
- Sets the `codecdata.fFramedone` flag to indicate that one encoded speech data buffer has been filled and is ready to be decoded.

Return Value

None

Parameters

None

3.5.6 libarrayFillDecoderExternalFlash() Function

Overview

The `libarrayFillDecoderExternalFlash()` function:

- Moves one data frame of encoded speech (5 or 20 bytes) from external flash memory to a user-defined encoded speech data buffer.
- Performs buffer management of the decoder input buffers.
- Increments `codecdata.sampleCount` by the number of bytes read.
- Clears `codecdata.countFill`.
- Sets the `codecdata.fFramedone` flag to indicate that one encoded speech data buffer has been filled and is ready to be decoded.

Return Value

None

Parameters

None

3.5.7 libarrayFillDecoderInputEEPROM() Function

Overview

The `libarrayFillDecoderInputEEPROM()` function:

- Moves one data frame of encoded speech (5 or 20 bytes) from data EEPROM to the user-defined encoded speech data buffer.
- Performs buffer management of the decoder input buffers.
- Increments `codecdata.sampleCount` by the number of bytes read.
- Sets the `codecdata.fFramedone` flag to indicate that one encoded speech data buffer has been filled and is ready to be decoded.

Return Value

None

Parameters

None

3.5.8 libBufManagerDecoder() Function

Overview

The `libBufManagerDecoder()` function:

- Manages the pointers to the user-defined raw speech buffers.
- Sets the `codecdata.fFrameplayed` flag to indicate that one frame of encoded speech data has been decoded and is ready for play back.
- Clears the `codecdata.fFramedone` flag to indicate that the decoder has completed execution.

> **Note:** This function can be called only after `libDecoder()` executes and `codecdata.fFrameplayed = '0'`.

Return Value

None

Parameters

None

3.5.9 libBufManagerEncoder() Function

Overview

The `libBufManagerEncoder()` function:

- Manages the pointers to the user-defined encoded speech buffers.
- Sets the `codecdata.fEncodedone` flag to indicate that one frame of speech data has been encoded.
- Clears the `codecdata.fFramedone` flag to indicate that a new frame of speech data must be collected.

Return Value

None

Parameters

None

3.5.10 libDecoder() Function

Overview

The `libDecoder()` function:

- 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

3.5.11 libDecoderInit() Function

Overview

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

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

Return Value

None

Parameters

None

3.5.12 libEncoder() Function

Overview

The `libEncoder()` function:

- 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

3.5.13 libEncoderInit() Function

Overview

The `libEncoderInit()` function:

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

Return Value

None

Parameters

The `libEncoderInit()` function has one parameter:

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

3.5.14 libEncoderKill() Function

Overview

The `libEncoderKill()` function:

- Frees the 1280 bytes dynamically allocated by `libEncoderInit()`.
- Clears the encoder state pointer.

Return Value

None

Parameters

None

3.5.15 libExtFlashErase() Function

Overview

The `libExtFlashErase()` function:

- Erases external Flash memory.
- Checks the erase status to determine if an erase failure occurred. If there is an erase failure, this function resets the flash device and calls `libExtFlashFailure()`.

Return Value

None

Parameters

None

3.5.16 libExtFlashFailure() Function

Overview

The `libExtFlashFailure()` function:

- Uses Timer 5 to blink LED4 continuously if Flash erase fails.
- Uses Timer 5 to blink LED3 continuously if Flash programming fails.

Note: This function is for development purposes only and must be modified to match the error handling process for your application.

Return Value

None

Parameters

The `libExtFlashFailure()` function has one parameter:

Parameter	`error`
Size	`int`
Usage	'`1`' = error erasing '`0`' = error programming

3.5.17 libExtFlashInit () Function

Overview

The `libExtFlashInit()` function:

- Configures the dsPIC30F general purpose I/O pins used for external Flash memory.
- Resets the flash read and write addresses.
- Sets PORTD<15:0>, PORTC<13>, PORTF<8:7> and PORTA<7> as outputs.
- Sets PORTA<6> as input.

Return Value

None

Parameters

None

3.5.18 libExtFlashReset() Function

Overview

The `libExtFlashReset()` function issues a reset command to the external Flash memory.

Return Value

None

Parameters

None

3.5.19 libExtFlashWrite() Function

Overview

The `libExtFlashWrite()` function:

- Stores encoded speech data into the external Flash memory.
- Checks the programming status to determine if there was a programming failure. If a failure has occurred, this function resets the flash device and calls the `libExtFlashFailure()` function.

Return Value

None

Parameters

The `libwriteExternalFlash()` function has two parameters, as shown in Table 3-6

TABLE 3-6: LIBEXTFLASHWRITE () FUNCTION PARAMETERS

Parameter	Type	Usage
`SampleEncdOpBuffer`	`short*`	Pointer to the encoded speech buffer to be stored.
`numOfencSamplesPerFrame`	`Char`	Number of words to be written to Flash memory.

3.5.20 libPWMInit() Function

Overview

The `libPWMInit()` function:

- Initializes the PWM speech playback interface.
- Initializes the selected timer (Timer 2 or Timer 3) for the PWM time base.
- Initializes the selected output compare channel (OC1 – OC8) for PWM mode.

Note: Select timer and output compare channel to be used in `spxlib_pwm.h`.

Return Value

None

Parameters

None

3.5.21 libPWMLoadSamples() Function

Overview

The `libPWMLoadSamples()` function:

- Updates the output compare secondary data register (OCxRS) every 8000 Hz.
- Increments `codecdata.countLoad`.

Note: This function is called from the Timer2 and Timer3 interrupt service routine.

Return Value

None

Parameters

None

3.5.22 libPWMStartSampling() Function

Overview

The `libPWMStartSampling()` function:

- Initializes the four data buffer pointers of `codecdata` with the address of the four user-defined buffers:
  - `codecdata.sampleIpBuffer`
  - `codecdata.sampleOpBuffer`
  - `codecdata.sampleEncdIpBuffer`
  - `codecdata.sampleEncOpBuffer`
- Clears the `codecdata.frameCount`.

Return Value

None

Parameters

The `libPWMStartSampling()` function has four parameters, as shown in Table 3-7.

TABLE 3-7: LIBPWMSTARTSAMPLING () FUNCTION PARAMETERS:

Parameter	Type	Usage
`buf1`	`short*`	Pointer to raw speech sample buffer 1 (160 words)
`buf2`	`short*`	Pointer to raw speech sample buffer 2 (160 words)
`buffer1`	`char*`	Pointer to encoded speech sample buffer 1 (20 bytes)
`buffer2`	`char*`	Pointer to encoded speech sample buffer 2 (20 bytes)

3.5.23 libRawBufManager() Function

Overview

The `libRawBufManager()` function:

- Manages the pointers to the user-defined raw speech sample buffers.
- Clears `codecdata.countFill`.
- Increments `codecdata.frameCount`.

> **Note:** This function is called from the ADC ISR when a frame of data (160 words) has been sampled.

Return Value

None

Parameters

None

3.5.24 libRwndOpRawBufPtr() Function

Overview

The `libRwndOpRawBufPtr()` function:

- Clears `codecdata.countLoad`.
- Clears `codecdata.fFrameplayed` to indicate that a user-defined raw speech buffer has been played.
- Rewinds the corresponding raw speech sample ping-pong buffer pointer to the beginning of the sample buffer to which it points.

> **Note:** This function is called from the Timer2 or Timer3 ISR.

Return Value

None

Parameters

None

3.5.25 libSi3000DCIFill() Function

Overview

The `libSi3000DCIFill()` function:

- Moves data from the DCI buffer registers, RXBUF0 – RXBUF3, to the user-defined raw speech sample buffers.
- Increments `codecdata.countFill` by 4.
- Performs speech buffer pointer management when a frame of 160 samples has been read from the DCI

> **Note:** This function is called from the DCI ISR when the DCI is the codec clock master.

Return Value

None

Parameters

None

3.5.26 libSi3000Init() Function

Overview

The `libSi3000Init()` function:

- Initializes the DCI module based on settings provided through `#define` statements in the `spxlib_common.h` file.
- Initializes the Si3000 codec based on user settings provided through `#define` statements in the `spxlib_Si3000.h` file.
- Clears `codecdata.countFill`, `codecdata.countLoad` and `codecdata.fFramedone`.

> **Note:** You can set up DCI as clock master or slave through `#define DCIMODE` in `spxlib_Si3000.h`. If it is set to '`1`', the DCI functions in Master mode. If it is set to '`0`', the DCI functions in Slave mode.

Return Value

None

Parameters

None

3.5.27 libSi3000LoadDCI() Function

Overview

The `libSi3000LoadDCI()` function performs the following tasks:

- Writes decoded speech data from a user-defined raw speech buffer to the DCI TXBUF0 – TXBUF3 registers.
- Increments `codecdata.countLoad` by 4.
- Checks the value of `codecdata.countLoad`. If the value equals 160, it resets `codecdata.countLoad` to '`0`', increments `codecdata.frameCount` by 1 and clears `codec.fFrameplayed`.

> **Note:** This function is called from the DCI ISR when the DCI is clock master.

Return Value

None

Parameters

None

3.5.28 libSi3000SlaveFillDCI() Function

Overview

The `libSi3000SlaveFillDCI()` function is used when DCI is configured as a slave. It performs the following tasks:

- Moves data from the DCI buffer registers, RXBUF0 – RXBUF3, to the user-defined raw speech sample buffers.
- Increments `codecdata.countFill` by 4.
- Performs speech buffer pointer management when a frame of 160 samples has been read from the DCI.

Note:	This function is called from the DCI ISR when the DCI is the clock slave.

Return Value

None

Parameters

None

3.5.29 libSi3000SlaveLoadDCI() Function

Overview

The `libSi3000SlaveLoadDCI()` function is used when DCI is configured as a slave. It performs the following tasks:

- Writes decoded speech data from user-defined raw speech buffer to the DCI TXBUF0 – TXBUF3 registers.
- Increments `codecdata.countLoad` by 4.
- Checks the value of `codecdata.countLoad`. If the value equals 160, it resets `codecdata.countLoad` to ‘`0`’, increments `codecdata.frameCount` by 1 and clears `codec.fFrameplayed`.

Note:	This function is called from the DCI ISR when the DCI is clock slave.

Return Value

None

Parameters

None

3.5.30 libSi3000StartSampling() Function

Overview

The `libSi3000StartSampling()` function:

- Initializes the four data buffer pointers of codecdata with the address of the four user-defined buffers:
  - `codecdata.sampleIpBuffer`
  - `codeddata.sampleOpBuffer`
  - `codecdata.sampleEncdIpBuffer`
  - `codecdata.sampleEncdOpBuffer`.
- Clears the four DCI transmit buffers, TXBUF0 – TXBUF3.
- Clears `codecdata.frameCount`.
- Enables the DCI and its interrupt.

Return Value

None

Parameters

The `libSi3000StartSampling` function has four parameters, as shown in Table 3-8.

TABLE 3-8: libSi3000StartSampling () FUNCTION PARAMETERS:

Parameter	Type	Usage
`buf1`	`short*`	Pointer to raw speech sample buffer1 (160 words)
`buf2`	`short*`	Pointer to raw speech sample buffer2 (160 words)
`buffer1`	`char*`	Pointer to encoded speech sample buffer1 (20 bytes)
`buffer2`	`char*`	Pointer to encoded speech sample buffer2 (20 bytes)

3.5.31 libSi3000StopSampling() Function

Overview

The `libSi3000StopSampling()` function:

- Clears the DCI interrupt flag.
- Disables the DCI interrupt.
- Disables the DCI module.

Return Value

None

Parameters

None

3.5.32 libStartPlay() Function

Overview

The `libStartPlay()` function sets the `codecdata.fStartPlay` flag. This flag is checked in the `libPWMLoadSamples()` and `libSi3000LoadDCI()` functions. Speech is played out through the selected interface only if the flag is '`1`'.

Return Value

None

Parameters

None

3.5.33 libStartPWM() Function

Overview

The `libStartPWM()` function performs the following tasks for the timer selected for the PWM time base (either Timer 2 or Timer 3):

- Sets the timer interrupt priority
- Clears the timer interrupt flag
- Enables the timer interrupt
- Enables the timer module

> **Note:** The time base for the PWM (Timer 2 or Timer 3) is selected in `spxlim_pwm.h`.

Return Value

None

Parameters

None

3.5.34 libStopPlay() Function

Overview

The `libStopPlay()` function clears the `codecdata.fStartPlay` flag. This flag is checked in the `libPWMLoadSamples()` and `libSi3000LoadDCI()` functions. Speech is played out through the selected interface only if the flag is '`1`'.

Return Value

None

Parameters

None

3.5.35 libStopPWM() Function

Overview

The `libStopPWM()` function performs the following tasks for the timer selected for the PWM time base (either Timer 2 or Timer 3):

- Clears the timer interrupt flag.
- Disables the timer interrupt.
- Disables the timer module.

Return Value

None

Parameters

None

3.5.36 libTblPtrSet() Function

Overview

The `libTblPtrSet()` function:

- Sets `pTblpag` and `pTbloff` with the address of the specified table number stored in program memory.
- Clears the `byteCount` variable, used for reading speech data from program memory.

Return Value

None

Parameters

The `libTblPtrSet()` function has one parameter:

Parameter	`tableno`
Size	`char`
Usage	The table number stored in program memory (see Figure 5-1)

3.5.37 libTblPtrSetEEPROM() Function

Overview

The `libTblPtrSetEEPROM` function sets `pTblpagEEPROM` and `pTbloffEEPROM` with the address of the specified table number stored in data EEPROM.

Return Value

None

Parameters

The `libTblPtrSetEEPROM()` function has one parameter:

Parameter	`tableno`
Size	`char`
Usage	The table number stored in data EEPROM.

Chapter 4. Incorporating Speech Encoding

4.1 INTRODUCTION

This chapter provides information to help you understand how to integrate the speech encoding portion of the dsPIC30F Speech Encoding/Decoding Library into your application and how to build with the library. A basic understanding of the encoder and interrupt timing is required to ensure correct real-time operation of the library.

4.2 HIGHLIGHTS

Items discussed in this chapter are:

- Integrating Speech Encoding
- Using the ADC Alternate Sampling Interface
- Speech Encoding Sample Application
- Using the Encoding and Decoding Libraries Together
- Building with the Encoder Library

4.3 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 (either the codec interface or the ADC), how data is sampled, how data buffers are used by the encoder, and how the library interacts with its interrupt handlers.

4.3.1 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 are output buffers used to store encoded speech data.

The input sampling rate is 8 kHz (of 16-bit data), and the encoder processes 20 milliseconds of data at a time. Each input buffer must have the capacity to store 160 integer samples. When the encoder processes a frame of data, it generates an output array, which may be as large as 20 bytes. The two output buffers must be large enough to hold 20 bytes. Example buffer definitions are shown below:

```
short in1[160], in2[160]; /* ping-pong input buffers */
char out1[20], out2[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 `in1` buffer is populated with speech data. After 20 msec, a frame of data is received and `in1` is filled. The library processes `in1` and populates `out1`, 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 `in1` is processed, the output is stored in `out1`. Likewise, after `in2` is processed, the output is stored in `out2`. This allows you to safely use the encoded data (i.e., for transmission or storage) in `out1`, while `out2` is being populated (and vise-versa). Table 4-1 shows how the pairs of input/output buffers are used by the library.

TABLE 4-1: ENCODER BUFFER USAGE

Buffer	Frame 0 (20 msec)	Frame 1 (20 msec)	Frame 2 (20 msec)	Frame 3 (20 msec)	Frame 4 (20 msec)
`in1`	Filled by interrupt service routine	Processed by library	Filled by interrupt service routine	Processed by library	Filled by interrupt service routine
`in2`	Idle	Filled by interrupt service routine	Processed by library	Filled by interrupt service routine	Processed by library
`out1`	Idle	Loaded with Encoded in1	Available for user handling	Loaded with Encoded in1	Available for user handling
`out2`	Idle	Idle	Loaded with Encoded in2	Available for user handling	Loaded with Encoded in2

4.3.2 Encoder Initialization

The 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`. 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 can not be enabled or disabled on a frame-by-frame basis. After `libEncoderInit()` is called, the VAD setting must not be modified.

4.3.3 Encoder Heap Utilization

The 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.

4.3.4 Data Sampling Initialization

After the encoder is initialized by calling `libEncoderInit()`, the sampling system must be initialized. If you are using the Si3000 codec for your sampling interface, initialization begins by calling `libSi3000Init()`. This function initializes the dsPIC30F's DCI module and the Si3000 based on the `#define` values set in `spxlib_Si3000.h`. For information on the Si3000 control registers, refer to **Appendix A. “Si3000 Codec Configuration”**.

After `libSi3000Init()` is called, sampling of the speech signal through the Si3000 will begin when `libSi3000StartSampling()` is called. This function must be called with the address of the four data buffers as the function parameters:

```
libSi3000StartSampling (&in1[0], &in2[0], &out1[0], &out2[0]);
```

Note:	Since the DCI interrupt service routine is shared between the encoder and decoder, you must define the global variable `EncoderState` and set it to ‘`1`’ before calling `libSi3000StartSampling()`. The DCI interrupt service routine references this variable and reads from either the Si3000 (when `EncoderState = 1`) or write to the Si3000 (when `EncoderState = 0`).

```
char EncoderState = 1;      // perform encoding
```

If you are using the 12-bit on-chip ADC for your sampling interface, data sampling initialization begins by calling `libADC12Init()`. This function initializes the ADC based on the `#define` values set in `spxlib_adc.h`. For details on setting up the ADC, refer to **Section 4.4 “Using the ADC Alternate Sampling Interface”**.

After `libADC12Init()` is called, sampling of the speech signal through the ADC will begin when `libADC12StartSampling ()` is called. This function must be called with the address of the four data buffers as the function parameters:

```
libADC12StartSampling (&in1[0], &in2[0], &out1[0], &out2[0]);
```

4.3.5 Data Sampling

Data sampling is managed by either the DCI interrupt service routine, if you are using the Si3000 codec, or the 12-bit ADC interrupt service routine, if you are sampling from the 12-bit ADC. Each interrupt service routine reads speech samples from the respective peripheral and stores the data in your defined input buffers. When a complete frame of 160 speech samples has been received by the interrupt service routine, the interrupt service routine sets the `codecdata.fFramedone` flag to ‘`1`’ and performs input buffer management, which allows new data to be collected by the interrupt service routine while the foreground library processes the newly received frame of speech data.

If you are using the Si3000 codec, DCI interrupt service routines will run only every 500 μsec instead of the speech sample period of 125μsec (1/8 kHz). To minimize the impact of the interrupt service routine on your application, the DCI is initialized with the buffer length control bits `BL<1:0>` set to ‘`11b`’. This mode allows four data samples to be buffered between interrupts and decreases the interrupt rate by a factor of four.

If you are using the 12-bit ADC, the frequency of the ADC interrupt service routine will depend on how many ADC channels you are converting and the ADC buffer length. **Section 4.4 “Using the ADC Alternate Sampling Interface”** explains this situation in depth.

4.3.6 Encoding

Speech encoding is performed by the `libEncoder()` function. This function can only be called after the respective sampling interface (DCI or 12-bit ADC interrupt service routine) has received a full frame of data for processing. When 160 samples of speech data have been received by the sampling interrupt service routine, the `codecdata.fFramedone` flag is set to ‘`1`’, which signifies that `libEncoder()` can be called.

The `libEncoder()` function takes a maximum of 370,000 instruction cycles to run. Since each data frame is 20 msec long, this function must be called 50 times each second to maintain continuous processing of data. The `libEncoder()` function should not be run from the DCI interrupt service routine, but only from your foreground code.

Immediately after `libEncoder()` completes running, `libBufManagerEncoder()` must be called. This function performs housekeeping duties related to the two output buffers, clears the `codecdata.fFramedone` flag and sets the `codecdata.fEncodedone` flag.

Once `libEncoder()` and `libBufManagerEncoder()` have run, you must do something with the encoded data before `libEncoder()` runs again. The encoded data will always be pointed to by the `codecdata.sampleEncdOpBuffer` pointer, and the number of bytes encoded (5 or 20) will be stored in `codecdata.numOfencSamplesPerFrame`. You must access the encoded data using these structure elements. For instance, if you are storing the data to external flash memory, you could use the `libwriteExternalFlash()` function provided by the library:

```
libwriteExternalFlash(codecdata.sampleEncdOpBuffer,
codecdata.numOfencSamplesPerFrame);
```

This function stores the correct number of recently encoded bytes to external flash memory.

4.3.7 End Data Sampling

For your convenience, the number of speech frames encoded by the library is saved in `codecdata.frameCount`. You can use this information to determine when to stop sampling. This may 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 you to store the number of frames you want to encode. Your application can compare `codecdata.frameCount` 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. You must also set `codecdata.recordSize` using the provided `#define` in `spxlib_common.h`:

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

Data sampling can be stopped by calling `libSi3000StopSampling()` for codec sampling or `libADC12StopSampling()` for ADC sampling. These functions prevent the DCI and ADC interrupt service routines, respectively, from running, and no new data will be sampled by the library.

When sampling is stopped and you want to return the encoder to an idle state, the following code sequence must be performed:

```
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 information about heap requirements, see **Section 4.3.3 “Encoder Heap Utilization”**.

4.4 USING THE ADC ALTERNATE SAMPLING INTERFACE

If you decide not to use a codec to sample speech, you can use the on-chip 12-bit ADC to sample speech for speech encoding. Since the Si3000 codec provides signal conditioning, convenient gain level adjustments and 16-bit data, you can expect degraded encoder performance using the 12-bit ADC. However, depending on your system requirements, you may be able to use the 12-bit ADC in your application. Appendix C presents a sample circuit you can use to interface a microphone to the ADC on the dsPIC30F. The circuit consists of an anti-aliasing filter, level shifter and audio amplifier.

Section 4.3 describes how the 12-bit ADC can be used with the Speech Encoding Library. The only difference between using the codec and the ADC is that now the ADC Interrupt handler is responsible for sampling data and populating the two input buffers. When the 12-bit ADC is used, data must still be sampled at a constant 8 kHz.

Note: For details on the dsPIC30F 12-bit ADC operation refer to the most recent copy of the *dsPIC30F Family Reference Manual* (DS70046).

4.4.1 Initializing the ADC

Initialization of the ADC for speech sampling is performed via the #defines in the spxlib_adc.h header file. The spxADC12 structure (defined in **Section 3.4.4 "ADC12 Structure"**) contains elements for each ADC-related special function register and several elements strictly for library use. By default, the ADC is configured to sample 3 channels (AN9, AN10 and AN11) with the microphone data on channel AN9.

TABLE 4-2: DEFAULT SPXADC12 STRUCTURE SETUP

Structure Element	Default Value	Comments
`adcon1val`	`0x03E4`	Signed fractional data, auto convert, auto sampling
`adcon2val`	`0x0438`	Scanning enabled, interrupts after each 15th sample/convert sequence
`adcon3val`	`0x123F`	8 kHz sampling based on system clock of 24.576 MIPs
`adchsval`	`0x0000`	Channel scanning enabled
`adpcfgval`	`0xF1FF`	Set AN9, AN10 and AN11 as analog inputs
`adcsslval`	`0x0E00`	Set AN9, AN10 and AN11 for input scanning
`spxadcintpri`	`5`	ADC interrupt priority level set to 5
`channelsinsequence`	`3`	3 channels in scanning sequence
`adcbufferlength`	`15`	Buffer length set to 15

A series of `#defines` in `spxlib_adc12.h` are provided for initialization. You can adjust these values based on the requirements for your specific application.

Once that you have set the `#defines` in `spxlib_adc12.h` for your application, another `#define` is provided to create a data structure of type `spxADC12`:

```
#define SPXADC12INIT   const spxADC12 ADC12 = ADC12CONFIG;
```

To make the ADC12 structure accessible to your source application, simply reference the `SPXADC12INIT` define in your source code, where you define your other data:

```
int my_variable1, my_variable2;     // my variables
SPXADCINIT        // ADC data structure instantiation
                  // This defines the initialized ADC12
                  // structure
...
```

4.4.2 ADC Buffer Length

The `ADC12.adcbufferlength` value must be carefully selected to meet two important requirements. The value must:

- Be an integer multiple of the number of channels being scanned
- Be an integer multiple of 160 x `ADC12.channelsinsequence`.

These requirements are necessary to ensure that buffer management is properly performed by the ADC interrupt service routine when a frame of data is collected for processing.

The value assigned to `adcbufferlength` must be equal to the value assigned the `SMPI<3:0>` bits in the `ADCON2` register. For instance, if four channels are being scanned the `adcbufferlength` can be set to 4, 8 or 16 (all integer multiples of 640). The `adcbufferlength` should not be set to 12, since this is not an integer multiple of 640.

4.4.3 Computing the Auto Sample Time

If your system has a different number of channels being scanned or uses a different system clock frequency, you will have to adjust the `ADCON3` register to correctly sample the input data at 8 kHz. Example 4-1 assumes that the auto sample and conversion mode is used, and that three ADC channels are being scanned with a system clock frequency of 24.576MHz. This example demonstrates how to compute the SAMC<4:0> and ADCS<5:0> fields of the ADCCON3 register.

EXAMPLE 4-1: AUTO SAMPLE TIME CALCULATION (AUTO SAMPLE MODE)

Assumptions:

F_{CY} = 24.576 MHz

Desired Sampling Frequency = 8000 Hz

Number of channels in sequence = 3

Determine the time bases:

$T_{CY} = 1/F_{CY}$ = 0.04069 μsec

Sampling Period = 1 / Sampling Frequency = 125 μsec

Time per channel = Sampling Period / Number of channels in sequence

= 125 μsec / 3

= 41.66 μsec

Compute T_{AD}:

$T_{AD} = T_{CY}/2$ (`<ADCS> + 1`) where T_{AD} is ADC clock period.

For convenience, set `ADCS<5:0>` to its maximum value, 63

T_{AD} = (0.04069/2) (64) = 1.30208 μsec

Compute the Conversion Time, T_{CONV}:

Conversion time, Tconv = 14 * T_{AD}

= 14 * 1.30208 μsec

= 18.229 μsec

Compute the Sampling time for each channel, T_{SAMP}:

T_{SAMP} = Sampling time for each channel – Conversion time

= 41.66 μsec – 18.229 μsec

= 23.431 μsec

Compute the Auto Sample Time Bits, `SAMC<4:0>`:

T_{SAMP} = x * T_{AD}, where x = Auto Sample Time Bits, `SAMC<4:0>`

x = T_{SAMP}/T_{AD}

= 23.431 μsec / 1.30208 μsec

= 18

In Example 4-1, the value ‘x’ corresponding to SAMC bits has to be loaded in the `ADCON3` register. With `ADCS<5:0>` taken as 63, the A/D conversion clock derived from the system clock and `SAMC<4:0>` equal to 18, the value to be loaded in the `ADCON3` register becomes 0x123F.

Table 4-3 shows different values for T_{SAMP} and the Auto Sample Time bits (`SAMC<4:0>`) corresponding to the number of channels in sequence based on a 24.576 MHz system clock frequency.

TABLE 4-3: DIFFERENT SAMC<4:0> VALUES FOR CHANNEL SCANNING

No. of Channels in Sequence	Sampling Time for Each Channel	T_{AD}*	Conversion Time (T_{CONV})	Sampling Time (T_{SAMP})	Auto Sample Time Bits
3	41.66 μsec	1.30208 μsec	18.229 μsec	23.42 μsec	18
4	31.25 μsec	1.30208 μsec	18.229 μsec	13.02 μsec	10
5	25.00 μsec	1.30208 μsec	18.229 μsec	6.77 μsec	6
6	20.83 μsec	1.30208 μsec	18.229 μsec	2.601 μsec	2

* T_{AD} computed with ADCS<5:0> set to 63

In Table 4-3, if the number of channels in sequence is less than three, then the value of the Auto Sample Time Bits exceeds the maximum value that can be set. Under this condition, you are not able to use the auto convert feature of the ADC and you must use an alternative conversion trigger source.

Likewise, if the number of channels in sequence is greater than six, then the value of the Auto Sample Time Bits will be less than the smallest value that can be set. In this situation, you can reduce T_{AD} to achieve a non-zero value for the Auto Sample Time Bits. Be aware that T_{AD} must have a minimum value of 667 nsec. If setting T_{AD} to 667 nsec still results in an Auto Sample Sample Time Bits of zero, you must use an alternate conversion trigger source.

4.4.4 Using the Timer 3 Conversion Trigger

If the computation for the `SAMC<4:0>` bits results in a value that exceeds the maximum SAMC value (31) or is less than one, you can configure the ADC to use Timer 3 as a conversion trigger source. This is performed by setting the `SSRC<2:0>` bits in the `ADCCON1` register to 2. In this instance, the period register of Timer 3 is set to a value, which is based on the difference between the total sampling time and the conversion time. Example 4-2 shows how to determine the period of Timer 3 for an application using just one ADC channel.

EXAMPLE 4-2: CONFIGURING TIMER 3 FOR ONE ADC CHANNEL

Assumptions:

F_{CY} = 24.576 MHz

Desired Sampling Frequency	= 8000 Hz
Number of channels in sequence	= 1

Determine the time bases:

T_{CY} = 1/F_{CY} = 0.04069 µsec

Sampling Period = 1 / Sampling Frequency = 125 µsec

Time per channel = Sampling Period / Number of channels in sequence
= 125 µsec / 1
= 125 µsec

Compute Tad:

T_{AD} = T_{CY}/2 (`<ADCS> + 1`) where T_{AD} is ADC clock period.

For convenience, set `ADCS<5:0>` to its maximum value, 63

Tad = (0.04069/2) (64) = 1.30208 µsec

Compute the Conversion Time, T_{CONV}:

Conversion time, T_{CONV} = 14 * T_{AD}
= 14 * 1.30208 µsec
= 18.229 µsec

Compute the Sampling time for each channel, T_{SAMP}:

T_{SAMP} = Sampling time for each channel – Conversion time
= 125 µsec – 18.229 µsec
= 106.771 µsec

Compute Timer 3 Period, PR3 (assume 1:1 prescale):

PR3 = 106.771 µsec / Tcy
= 106.771 µsec / 0.04069 µsec
= 2624

4.4.5 ADC Interrupt Service Routine

The ADC Interrupt Service Routine calls the function `libADCFillBuffer()` which reads data from the ADC Buffer and copies it to the appropriate encoder input buffer. You will need to modify `libADCFillBuffer()` to include reads of any other ADC channels that you are sampling. You will find `libADCFillBuffer()` in the file `libADCFillBuffer.c`, located in the `src` folder.

Note: `libADCFillBuffer()` assumes that the microphone input is the first channel being scanned.

4.5 SPEECH ENCODING SAMPLE APPLICATION

A speech encoding sample application is shown in Appendix D. The sampling interface selection, Si3000 or 12-bit ADC, is made from a preprocessor conditional called `CODEC_INTERFACE`.

4.6 USING THE ENCODING AND DECODING LIBRARIES TOGETHER

Due to the Encoding library and Decoding library RAM requirements, the libraries may not be run simultaneously (full-duplex) on current dsPIC30F devices. However, each library may both reside in your application and be used one library at a time (half-duplex). This means that while the Encoding library is running, the Decoding library must be idle, and vise-versa. When using both libraries in your application, only 1 pair of input/output buffers must be defined. To avoid confusion of naming the buffer as an input or output, you can name the shared buffers generically:

```
char buffer11[20]; // decoder input/encoder output buffer 1
char buffer12[20]; // decoder input/encoder output buffer 2
short buffer1[160];// decoder output/encoder input buffer 1
short buffer2[160];// decoder output/encoder input buffer 2
```

Note: Soon new dsPIC30F devices will be available which support full-duplex operation of the Speech Encoding and Decoding libraries.

4.7 BUILDING WITH THE ENCODER LIBRARY

The process of integrating the Speech Encoding Library with your application is described in **Section 4.3 "Integrating Speech Encoding"**. To actually build the library into your application, you must add the `spxlibencoderlib.a` file to your MPLAB IDE project, add the appropriate interrupt service routine file to your project and also reference the appropriate header files from your application source code. Failure to reference the correct header files will result in a build failure. As described in **Section 4.3.3 "Encoder Heap Utilization"**, a heap of 1282 bytes is required for the speech encoding library. The heap can be reserved using the linker build options in MPLAB IDE.

Note: The `spxlibencoderlib.a` file is distributed in the `Lib` folder.
Source files are distributed in the `Src` folder.
Header files are distributed in the `Include` folder.

Table 4-4 lists all the files associated with the library.

TABLE 4-4: DSPIC30F SPEECH ENCODING LIBRARY FILES

File	Usage
`adcisr.c`	Interrupt service routine source file for ADC (required for ADC sampling interface)
`dciisr.c`	Interrupt service routine source file for DCI (required for Si3000 sampling interface)
`libADCFillBuffer.c`	Source file for ADC sampling (called by interrupt service routine)
`libExtFlashFailure.c`	Source file for external Flash error handling
`libspxencoderlib.a`	Library file
`spxlib_adc.h`	Header file required for ADC sampling
`spxlib_common.h`	General library header file
`spxlib_Si3000.h`	Header file required for Si3000 sampling

Chapter 5. Incorporating Speech Decoding

5.1 INTRODUCTION

This chapter provides information to help you understand how to integrate the speech decoding portion of the dsPIC30F Speech Encoding/Decoding Library into your application and how to build with the library. A basic understanding of the decoder and interrupt timing is required to ensure correct real-time operation of the library.

5.2 HIGHLIGHTS

Items discussed in this chapter are:

- Integrating Speech Decoding
- Using the PWM Alternate Playback Interface
- Speech Decoding Sample Application
- Using the Decoding and Encoding Libraries Together
- Building with the Speech Decoding Library

5.3 INTEGRATING SPEECH DECODING

To interface your application with the library decoder you need to be familiar with the data buffers used by the decoder, how the decoder is initialized, how data is sampled, and how the library decoder works with its interrupt handlers.

5.3.1 Data Buffers

The decoder uses four data buffers which you must define. Two of these buffers are input buffers used to store sampled speech data. The other two are output buffers used to store encoded speech data.

The output sampling rate of the decoder is 8 kHz. The library processes one frame of data at a time, and each frame represents 20 milliseconds of speech. Each input buffer must have the capacity to store 20 bytes, which is the largest output buffer generated by the encoder. When the decoder processes a frame of data, it generates an output array of 160 integer samples. Example buffer definitions are shown below:

```
char in1[20], in2[20];     /* ping-pong input buffers */
short out1[160], out2[160];/* ping-pong output 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 `in1` for decoding and it is processed by the library. After executing, the decoder populates the `out1` buffer with speech data. When output sampling begins, `out1` is played back through the respective interrupt service routine handler over the course of the next 20 msec frame, one sample every 125 µsec. Since the speech playback process must be continuous, `in2` is filled and processed by the library, which populates the `out2` buffer, while `out1` is being played back. Likewise, when `out2` is being played back, `out1` is being populated with new decoded data from `in1`. The `in1` and `in2` 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 5-1 shows how the pairs of input/output buffers are used by the library.

TABLE 5-1: DECODER BUFFER USAGE

Buffer	Initialization	Frame 0 (20 msec)	Frame 1 (20 msec)	Frame 2 (20 msec)	Frame 3 (20 msec)
in1	Filled and processed by library	Idle (available for filling)	Filled and processed by library	Idle (available for filling)	Filled and processed by library
in2	Idle (available for filling)	Filled and processed by library	Idle (available for filling)	Filled and processed by library	Idle (available for filling)
out1	Loaded with Decoded in1	Played out by interrupt service routine	Loaded with Decoded in1	Played out by interrupt service routine	Loaded with Decoded in1
out2	Idle	Loaded with Decoded in2	Played out by interrupt service routine	Loaded with Decoded in2	Played out by interrupt service routine

5.3.2 Decoder Initialization

The 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. No user-defined VAD selection is required for the decoder.

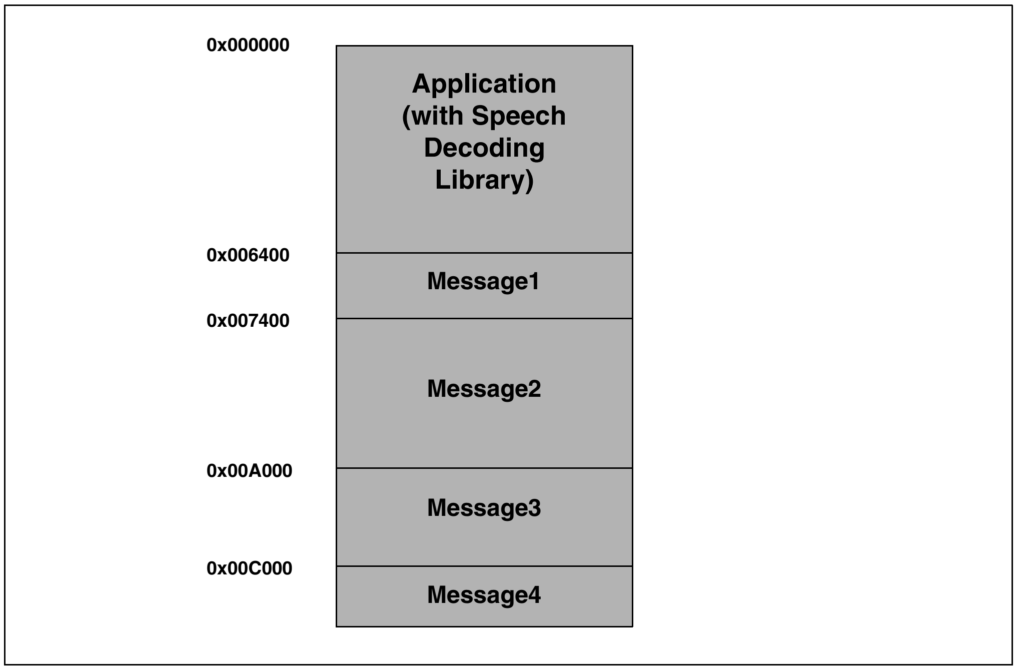
If you will be decoding a speech sample stored in data EEPROM, external Flash memory or program memory, you must also initialize the variables used by the library to read from these locations. The functions shown in Table 5-2 perform this task and they must be called as part of decoder initialization.

TABLE 5-2: MEMORY READ INITIALIZATION FUNCTIONS

Memory Type	Function
Data EEPROM	`libTblPtrSetEEPROM(char tableno)`
External Flash	`libExtFlashInit()`
Program Memory	`libTblPtrSet(char tableno)`

The functions `libTblPtrSetEEPROM()` and `libTblPtrSet()` allow you 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



You must reference your list of unique array names in `spxlib_common.inc`, which defines a table with ten entries. Note that there is a leading underscore in each array name.

```
.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 initialize the playback of Message2 from program memory, you call `libTblPtrSet(2)`, since Message2 is the second table entry. In this example, this will force the decoder to begin decoding from program memory address 0x7400 (the starting location of Message2).

5.3.3 Decoder Heap Utilization

The decoder does not require a heap. All memory used by the library is pre-allocated.

5.3.4 Decoding the First Frame

Before the output sampling system is initialized, one frame of speech must first be 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 populating an input buffer of the decoder. This task is performed by the functions shown in Table 5-3. These functions read a frame of data to decode from the data EEPROM, external Flash and program memory locations, respectively. Select the function to use based on the location of your encoded data.

TABLE 5-3: MEMORY READ FUNCTIONS

Memory Type	Function
Data EEPROM	`libarrayFillDecoderInputEEPROM()`
External Flash	`libarrayFillDecoderFlash()`
Program Memory	`libarrayFillDecoderInputPM()`

After the decoder input data is read from memory, decoding is performed by calling `libDecoder()` and then calling `libBufManagerDecoder()`, which performs buffer management.

The method to decode the first frame of speech from Program Memory is shown here:

```
libarrayFillDecoderInputPM(); //Copy data from Program Memory
libDecoder();                 //Decode data
libBufManagerDecoder();       //Rewind sample buffer pointer.
```

5.3.5 Speech Playback Initialization

After the decoder is initialized and one frame of speech has been decoded, the output sampling system must be initialized for speech playback. If you are using the Si3000 codec for your playback interface, initialization begins by calling `libSi3000Init()`. The function `libSi3000Init()` initializes the dsPIC30F's DCI module and the Si3000 based on the `#define` values set in `spxlib_Si3000.h`. For information on the Si3000 control registers, refer to **Appendix A. "Si3000 Codec Configuration"**.

After `libSi3000Init()` is called, playback of the speech signal through the Si3000 begins when `libSi3000StartSampling()` and `libStartPlay()` are called. `libSi3000StartSampling()` must be called with the address of the four data buffers as the function parameters:

```
Char EncoderState = 0;
libSi3000StartSampling (&out1[0], &out2[0], &in1[0], &in2[0]);
libStartPlay();
```

Note: Since the DCI interrupt service routine is shared between the encoder and decoder, you must define the global variable `EncoderState` and set it to '`0`' before calling `libSi3000StartSampling()`. The DCI interrupt service routine references this variable and either reads from the Si3000 (when `EncoderState = 1`) or writes to the Si3000 (when `EncoderState = 0`)

If you are using the PWM (Output Compare module) for your speech playback interface, playback initialization begins by calling `libPWMInit()`. The function `libPWMInit()` initializes the PWM based on the `#define` values set in `spxlib_pwm.h`. For details on setting up the PWM, refer to **Section 5.4 "Using the PWM Alternate Playback Interface"**.

After `libPWMInit()` is called, the speech signal is played back through the PWM when `libStartPWM()` and `libStartPlay()` are called:

```
    libPWMInit();
    libStartPWM();
    libStartPlay();
```

5.3.6 Speech Playback

Speech playback is managed by either the DCI interrupt service routine, if you are using the Si3000 codec, or the timer interrupt service routine (Timer2 or Timer3), if you are using the PWM.

Each interrupt service routine writes decoded speech samples to its respective peripheral from the decoder's output buffers. When a complete frame of decoded speech (160 samples) has been output by the interrupt service routine, the interrupt service routine sets the `codecdata.fFrameplayed` flag to '`0`' and processes the output buffer. Buffer management allows new decoded data to be played by the interrupt service routine, while the foreground library code decodes another frame of speech.

If you are using the Si3000 codec, DCI interrupt service routines only run every 500 µsec instead of the speech sample period of 125 µsec (1/8KHz). To minimize the impact of the interrupt service routine on your application, the DCI is initialized with the buffer length control bits `BL<1:0>` set to `11b`. This mode allows four data samples to be buffered between interrupts and decreases the interrupt rate by a factor of four.

If you are using the PWM, the frequency of the Timer2 or Timer3 interrupt service routine will depend on how fast you are running the PWM. The PWM must always be run at a multiple of 8 kHz: either 16 kHz, 24 kHz or 32 kHz. Table 5-4 shows the timer interrupt period for each possible PWM setting.

TABLE 5-4: TIMER INTERRUPT PERIOD FOR SPEECH PLAYBACK THROUGH PWM

PWM Frequency	Timer Interrupt Period
16 kHz	62.50 µsec
24 kHz	41.67 µsec
32 kHz	31.25 µsec

5.3.7 Decoding

Before decoding can be performed, the appropriate input buffer must first be loaded with data. The `codecdata.fFramedone` and `codecdata.fFrameplayed` structure elements are 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

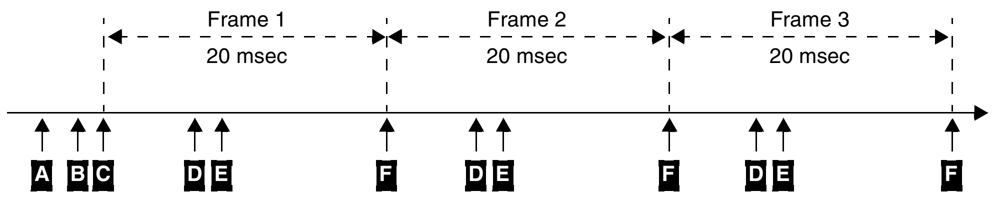
Element	Function
`codecdata.fFramedone`	'`0`' - copy data to input buffer '`1`' - process data from input buffer
`codecdata.fFrameplayed`	'`0`' - output buffer ready to be read (ready for playback) '`1`' - output buffer is being read (playback in process)

If the `codecdata.fFramedone` flag is '`0`' (indicating that data needs to be copied to a decoder input buffer) and the `codecdata.fFrameplayed` 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. Data is read into an input buffer using the appropriate function from Table 5-3.

Once the new input data has been read and the `codecdata.fFramedone` flag is '`1`', `libDecoder()` 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, `libBufManagerDecoder()` must be called after the `codecdata.fFrameplayed` flag is '`0`'.

The `libDecoder()` function takes approximately 56000 cycles to run. Since this function runs in a such a short amount of time, it is important to adhere to the guidelines provided above. A decoder timeline is shown in Figure 5-2.

FIGURE 5-2: DECODER TIMELINE



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 `libBufManagerDecoder()` is called, which performs buffer management. `libBufManagerDecoder()` also sets `fFrameplayed` flag to '`1`' and sets `fFramedone` flag to '`0`'.

C The sampling interface is initialized. Speech playback begins when `libStartPlay()` is called.

D New data to decode is read from memory and copied to the appropriate input buffer (see Table 5-3). 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. `libDecoder()` requires ~ 56000 instruction cycles.

F The sampling interface interrupt service routine transmits the last speech sample of an output buffer, performs output buffer management and sets `fFrameplayed` flag to '`0`'. You must now call `libBufManagerDecoder()` which performs input buffer management, sets `fFrameplayed` flag to '`1`' and sets `fFramedone` flag to '`0`'.

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

The location of events D and E are arbitrary, and occur under your control. They may occur at any time within each 20-msec 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 interrupt service routine to run unimpeded. Failure to let the interrupt service routine run as normal will result in degraded audio playback quality.

5.3.8 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 to 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. You can initialize `codecdata.arraysizeinbytes` using the provided `#define` in `spxlib_common.h`:

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

Speech playback can be stopped by calling `libSi3000StopSampling()` for codec playback or `libStopPWM()` for PWM playback. These functions prevent the DCI and Timer interrupt service routines, respectively, from running and thereby stop the playback of speech.

After sampling is stopped by calling `libSi3000StopSampling()` or `libStopPWM()`, the following code sequence must be performed to return the decoder to a passive state:

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

5.4 USING THE PWM ALTERNATE PLAYBACK INTERFACE

If you decide not to use the codec to play decoded speech, you can use the on-chip Output Compare module in PWM mode instead. Since the Si3000 codec provides signal conditioning, convenient gain level adjustments and a 16-bit interface, you can expect slightly degraded playback performance using the PWM. However, depending on your system requirements, you still may be able to use it in your application. See **Appendix C. "ADC/PWM Interface Reference Design"** for a sample circuit you can use to interface the PWM to an audio output circuit for driving a headset. The circuit consists of a low-pass filter and audio amplifier with adjustable gain.

Section 5.3 "Integrating Speech Decoding" describes how the PWM can be used with the library decoder. The only difference between using the codec and the PWM is that the interrupt handler of the PWM's time base (either Timer 2 or Timer 3) is responsible for outputting the data for playback (instead of the DCI Interrupt). When the PWM is used, data must still be output at a constant 8 kHz. However, the PWM must be run at a higher multiple frequency (two to four times) to obtain acceptable audio quality.

Note: For information on the dsPIC30F Output Compare peripheral, refer to the most recent copy of the *dsPIC30F Family Reference Manual* (DS70046).

5.4.1 Initializing the PWM

The PWM is initialized for speech playback with the `#defines` in the `spxlib_pwm.h` header file. The `spxPWM` structure provides a convenient interface for using the PWM. By default, the PWM is configured for 32 kHz using Timer 2 as the time base on Output Compare channel 5. Table 5-6 shows the default values of the spxPWM structure.

TABLE 5-6: DEFAULT spxPWM STRUCTURE SETUP

Structure Element	Default Value	Comments
`fpwm`	32000	PWM Frequency in Hz
`tmrprsclval`	1	For 1:1 prescaler
`period_count`(1)	767	Based on system clock of 24.576 MIPs
`samshift`	0	Not used
`max`(2)	0xFFFF	16-bit range
`timerxintpri`	4	Selected timer interrupt priority level set to 4
`scalefactor`(1)	0.0117	Used to compute PWM secondary data register
`timer`	2	Use Timer 2
`count`	4	PWM oversampling ratio (fpwm/8000)
`occon`	5	Use Output Compare channel 5

Note 1: These elements are automatically computed from an expression in spxlib_pwm.h that you must not modify.

2: The value of the `max` parameter must not be modified.

A series of `#defines` in `spxlib_pwm.h` are provided for initialization. You can adjust some of these values based on the requirements for your specific application. The dsPIC30F allows only Timer 2 or Timer 3 to be used as the PWM time base, so the `timer` element should only be set to '`2`' or '`3`'. Any available output compare channel can be used with the library decoder.

Once you have set the `#defines` in `spxlib_pwm.h` for your application, another `#define` in `spxlib_pwm.h` is provided to initialize the `spxPWM` structure:

```
#define SPXPWMINIT spxPWM PWM = PWMCONFIG;
```

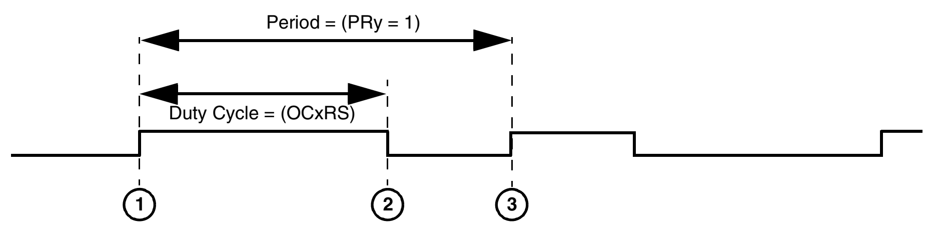
To make the `spxPWM` structure accessible to your application, simply reference the `SPXPWMINIT` defined in your source code, where you define your other data:

```
    int my_variable1, my_variable2;  // my variables
    SPXPWMINIT                       // PWM data structure
                                     // instantiation. Defines
                                     // the initialized PWM
                                     // structure.
```

5.4.2 Setting the PWM Frequency

The PWM frequency, set by the `#define FPWM` in `spxlib_pwm.h`, controls the frequency with which a new duty cycle is started on the output compare pin (see Figure 5-3). The PWM signal frequency must be much higher than the desired bandwidth of the signals to be produced. Generally, the higher the PWM frequency, the lower the order of filter is needed to interface with the PWM signal, which results in a higher quality audio signal.

FIGURE 5-3: PWM OUTPUT WAVEFORM



1. Timer is cleared and new duty cycle value is loaded from OCxRS into OCxR.
2. Timer value equals value in the OCxR register, OCx pin is driven low.
3. Timer overflow, value from OCxRS is loaded into OCxR, OCx pin driven high. TYIF interrupt flag is asserted.

The PWM frequency must always be a multiple of the native output sampling rate of the decoder, namely 8 kHz. The library decoder can run with a PWM frequency of 32 khz, 24 kHz or 16 kHz. The maximum PWM frequency is a function of FCY, the instruction rate of the dsPIC30F. Best performance is achieved with a PWM frequency of 32 kHz. Refer to Table 3-3 to determine the minimum processor frequency to operate with you PWM setting.

The `spxPWM.count` structure element sets the ratio of output oversampling (PWM frequency/8 kHz). This value is used to update the output compare secondary data register (OCxRS) at a constant rate of 8kHz. For instance, if the PWM frequency is 32 kHz, the OCxRS will be updated only every 4th timer interrupt. This is controlled by setting the structure element `spxPWM.count` to ‘4’. `spxPWM.count` is initialized by the `#define COUNT` in `spxlib_pwm.h` and should always be set based on the PWM frequency as shown in Table 5-7.

TABLE 5-7: COUNT VALUE AS A FUNCTION OF PWM FREQUENCY

PWM Frequency	Count Value
32 kHz	4
24 kHz	3
16 kHz	2

5.4.3 Timer 2/Timer 3 Interrupt Service Routine

Your selected PWM time base, Timer 2 or Timer 3, controls the duty cycle of the PWM signal generated on the output compare pin. The timer interrupt service routine will automatically run at your chosen PWM frequency as shown in Table 5-8. When the timer interrupt service routine runs, it will update the appropriate duty cycle register only every 125 µsec, which supports the 8 kHz output sampling rate of the decoder. As with the DCI interrupt handler, when a full frame of 160 output samples has been delivered to the PWM, output buffer management automatically maintains continuous playback of your decoded speech signal.

TABLE 5-8: TIMER ISR PERIOD AS A FUNCTION OF PWM FREQUENCY

PWM Frequency (µsec)	Timer ISR period (µsec)	OCxRS update period (µsec)
32 kHz	31.25	125
24 kHz	41.67	125
16 kHz	62.50	125

5.5 SPEECH DECODING SAMPLE APPLICATION

A speech decoding sample application using external flash is shown in Appendix D. The sampling interface selection, Si3000 or PWM, is made from a preprocessor conditional called `CODEC_INTERFACE`. This example demonstrates how to determine when sampling should be stopped after a speech sample is played back.

5.6 USING THE DECODING AND ENCODING LIBRARIES TOGETHER

Due to the amount of RAM required for encoding and decoding, the library decode and library encoder can not be run simultaneously (full-duplex) on current dsPIC30F devices. However, each library can reside in your application and be used one library at a time (half-duplex). This means that while the library encoder is running, the library decoder must be idle, and vise-versa. When you use both libraries in your application, you only need to define one set of input/output buffers. To avoid confusion of naming the buffer as an input or output, you can name the shared buffers generically:

```
char buffer11[20];   // decoder input/encoder output buffer 1
char buffer12[20];   // decoder input/encoder output buffer 2
short buffer1[160];  // decoder output/encoder input buffer 1
short buffer2[160];  // decoder output/encoder input buffer 2
```

5.7 BUILDING WITH THE SPEECH DECODING LIBRARY

The process of integrating the Speech Decoding Library with your application is described in **Section 5.3 "Integrating Speech Decoding"**. To actually build the library into your application, you must add the `spxlibdecoderlib.a` file to your MPLAB IDE project, add the appropriate interrupt service routine file to your project and also reference the appropriate header files from your application source code. Failure to reference the correct header files will result in an application build failure.

If you are accessing encoded speech files stored in data EEPROM or program memory, you will also have to edit `spxlib_common.inc` and add this file to your MPLAB IDE project. This file contains the list of symbols, which reference each speech sample (see **Section 5.3.2 "Decoder Initialization"**) your application will playback.

If you are playing back speech from program memory, you must add the `libTblPtrSet.s` file to your project. If you are playing back speech from data EEPROM, you must add the `libTblPtrSetEEPROM.s` file to your project. If you require support for more than ten messages, you will need to modify these files.

Note: The `spxlibencoderlib.a` file is distributed in the `Lib` folder.
Source files are distributed in the `Src` folder.
Header files are distributed in the `Include` folder.

Table 5-9 lists all the files associated with the library.

TABLE 5-9: DSPIC30F SPEECH DECODING LIBRARY FILES

File	Usage
`dciisr.c`	Interrupt service routine source file for DCI (required for Si3000 output interface)
`libspxdecoderlib.a`	Library file
`libTblPtrSet.s`	Source file for initializing playback from program memory
`libTblPtrSetEEPROM.s`	Source file for initializing playback from data EEPROM
`spxlib_common.h`	General library header file
`spxlib_common.inc`	Header file which defines symbol table for speech playback
`spxlib_pwm.h`	Header file required for PWM output interface
`spxlib_Si3000.h`	Header file required for Si3000 output interface
`timer2isr.c`	Interrupt service routine source file required for Timer 2 (PWM output interface)
`timer3isr.c`	Interrupt service routine source file required for Timer 3 (PWM output interface)

NOTES:

Chapter 6. Speech Encoding Utility

6.1 INTRODUCTION

The dsPIC30F Speech Encoding/Decoding Library 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 dsPIC30F with the library decoder. This chapter describes how to use the speech encoding utility.

6.2 HIGHLIGHTS

Items discussed in this chapter are:

- System Requirements
- Installation
- Encoding Speech From a Microphone
- Encoding Speech from a WAV file
- Recommendations for Encoding from a Microphone
- Using the Command Line Decoder

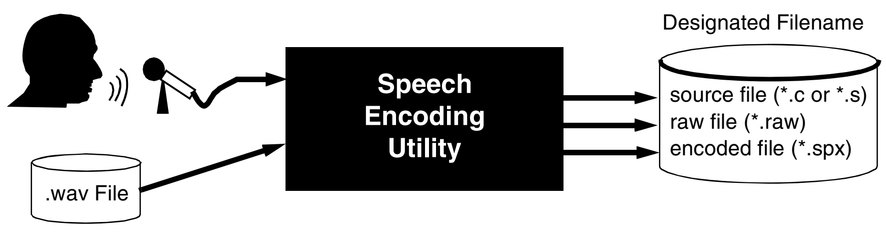
6.3 SYSTEM REQUIREMENTS

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

6.4 OVERVIEW

The dsPIC30F Speech Encoding/Decoding Library is designed to optimize computational performance and minimize RAM usage for speech-based applications embedded in dsPIC30F devices. The Speech Encoding Utility allows you to create encoded speech files from a microphone or a pre-recorded `.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

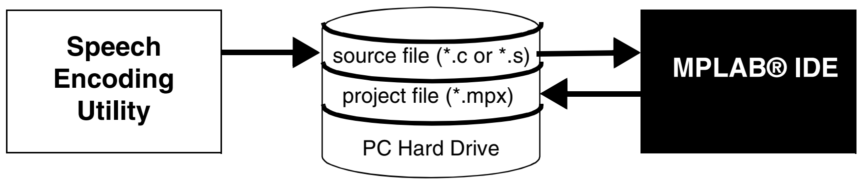
The Speech Encoding Utility allows you to select the type of memory in which to store your encoded speech file. The target memory selection ensures that the file is encoded in the correct format for:

- Program Memory
- Data EEPROM
- RAM
- External Flash

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



6.5 INSTALLATION

The speech encoding utility is automatically installed at the time the dsPIC30F Speech Encoding/Decoding Library is set up. The speech encoding utility is installed in the `dsPIC30F_SEDLibrary\Utils\SpeechEncoderUtility` folder under the root installation directory (which defaults to `C:\Program Files\Microchip`).

The speech encoding utility consists of the files listed in Table 6-1.

TABLE 6-1: SPEECH ENCODING UTILITY FILES

Filename	Purpose
`AWSpeexDec.exe`	Command line Speex decoder
`dsPICSpeechRecord.exe`	Utility executable
`SpeechRecord.dll`	Utility DLL

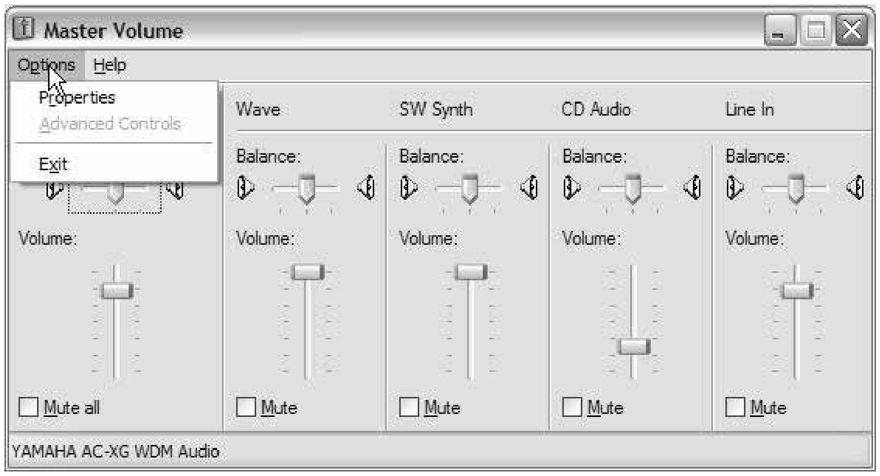
6.6 ENCODING SPEECH FROM A MICROPHONE

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

Select microphone input:

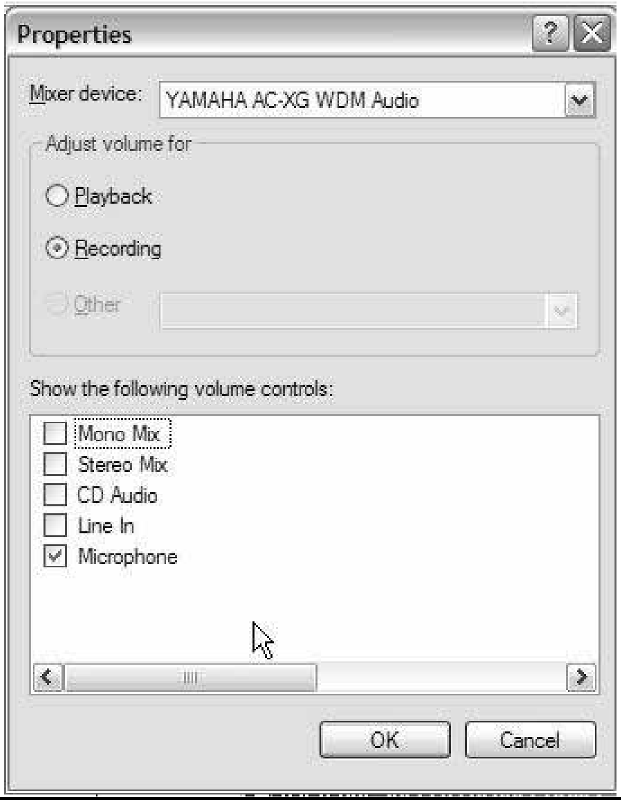
1. Launch the Windows Master Volume Control (*Start>Programs>Accessories>Entertainment>Volume Control*). When the Master Volume dialog displays select *Options>Properties*, as shown in Figure 6-3.

FIGURE 6-3: MASTER VOLUME CONTROL



2. On the Properties dialog (Figure 6-4), check **Adjust volume for Recording** and **Microphone,** then click **OK**.

FIGURE 6-4: MASTER VOLUME PROPERTIES DIALOG


Properties

Mixer device: YAMAHA AC-XG WDM Audio

Adjust volume for

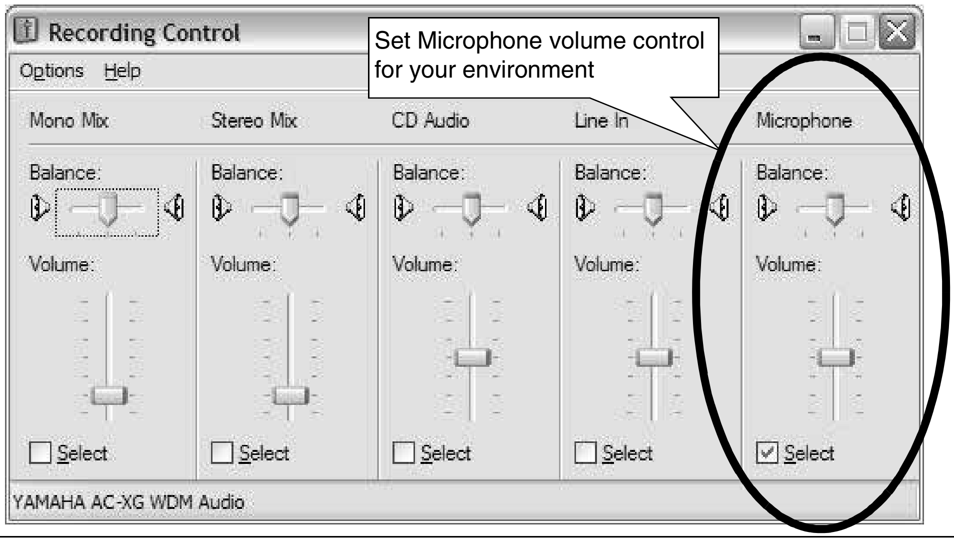
- Playback
- [✓ Recording]
- Other

Show the following volume controls:

- ☐ Mono Mix
- ☐ Stereo Mix
- ☐ CD Audio
- ☐ Line In
- ☑ Microphone

3. When the Recording Control dialog displays the microphone volume controls, adjust the settings for your environment.

FIGURE 6-5: RECORDING CONTROL DIALOG

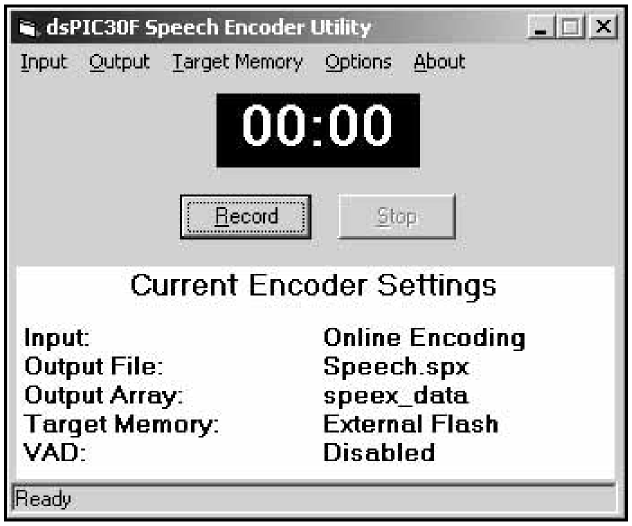


Note:	The dialogs illustrated here reflect a PC running Windows XP. Your dialogs may be slightly different to match your operating system.

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 chose not to install the icons, navigate to the dsPIC30F Speech Encoding/Decoding Library 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

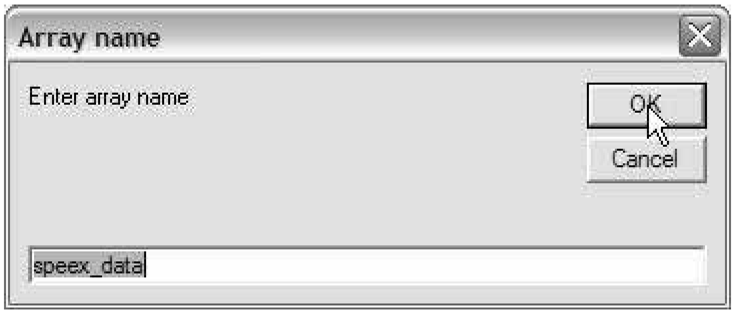

dsPIC30F Speech Encoder Utility
Input Output Target Memory Options About
00:00
Record
Stop
Ready

Current Encoder Settings

Input:	Online Encoding
Output File:	Speech.spx
Output Array:	speex_data
Target Memory:	External Flash
VAD:	Disabled

2. From the *Input* menu, select *Mic*.
3. From the *Output* menu, select *Array Name*. When the Array name dialog displays (Figure 6-7), click **OK** to accept the default array name (speex_data).

FIGURE 6-7: ARRAY NAME DIALOG



4. From the *Output* menu, select *Filename*. When the Save As dialog displays, designate the file name and location. The default directory for storing files generated by the Speech Coding Utility is `c:\Program Files\Microchip\dsPIC30F_SEDLibrary\Utils\SpeechEncoderUtility`.
5. From the *Target Memory* menu, select the type of memory you want to use (see Table 6-2).

TABLE 6-2: 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. From the *Options* menu, decide if you want to use Voice Activity Detection (VAD) to apply additional compression to voids (silent periods) in the speech file. A check means this option is selected.

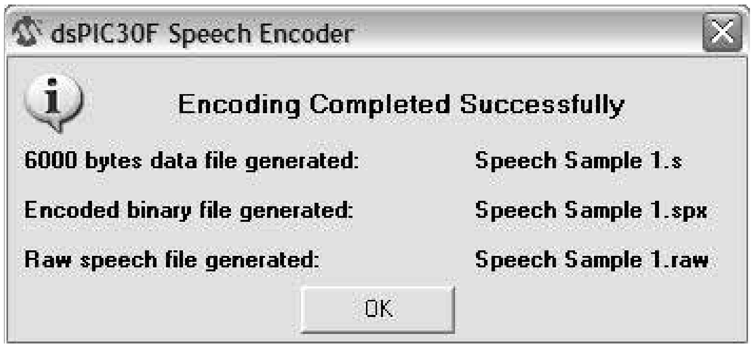
Record your message:

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 dsPIC30F6014 is 4096 bytes, which can store approximately 4 seconds of encoded speech.

2. When you are finished click **Stop**. 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



Encoding Completed Successfully

6000 bytes data file generated:	Speech Sample 1.s
Encoded binary file generated:	Speech Sample 1.spx
Raw speech file generated:	Speech Sample 1.raw

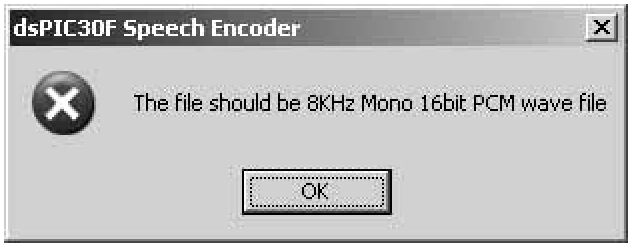
6.7 ENCODING SPEECH FROM A WAV FILE

To encode speech from a WAV file:

1. From the *Input* menu select *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 6-2).
4. Enable or disable VAD from the *Options* menu.
5. Press the **Encode** button.
6. Select the WAV file to encode.

Note: You must ensure that the source WAV file has compatible characteristics. An incompatible format will generate an error message, as shown in Figure 6-9.

FIGURE 6-9: WAV FILE FORMAT ERROR MESSAGE



The file should be 8KHz Mono 16bit PCM wave file

An Encoding Completed message displays the properties of the three output files generated, as shown in Figure 6-8.

6.8 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 has shown good results with the LabTec Axis301 headset microphone, which has demonstrated good results across a cross-section of speakers.

6.9 USING THE COMMAND LINE DECODER

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

7.1 INTRODUCTION

The dsPIC30F Speech Encoding/Decoding Library supports an external memory interface, 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.

7.2 HIGHLIGHTS

Items discussed in this chapter are:

- 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

7.3 USING EXTERNAL FLASH MEMORY

The dsPIC30F 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 5.0 volt 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 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-Kword interface can store approximately two minutes of compressed speech. If 128 Kwords are required, the 17th address bit can be implemented from any unused general-purpose I/O pin.

7.3.1 Encoding to External Flash Memory

In store and playback applications such as voice recorders and answering machines, data encoded by the library must be stored in a manner that it can be played back at a future time. 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 two milliseconds, 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-µsec programming time (per word), which makes it suitable for use in the library. Since speech frames are encoded to a maximum size of 20 bytes, approximately 120 µsec are required to write to the flash every 20 msec. This modest amount of overhead makes the AMD29F200B a good choice for real-time speech storage.

The external Flash functions listed in Table 7-2 are provided by the library for use with the encoder. As with all flash memories, the locations to be programmed must first be erased before they are programmed. The function `libExtFlashErase()` will erase the entire flash memory. Consult the AMD29F200B Data Sheet for sector erase information.

TABLE 7-2: ENCODER EXTERNAL FLASH MEMORY FUNCTIONS

Function	Purpose
`libExtFlashInit()`	Initializes the I/O lines to interface with Flash memory
`libExtFlashReset()`	Resets Flash memory
`libExtFlashErase()`	Erases entire Flash memory
`libExtFlashWrite()`	Writes last encoded speech frame to Flash memory
`libExtFlashError()`	Flash memory error handler

7.3.2 Decoding Speech from Flash Memory

External Flash memory can be used as a memory source for decoding encoded 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 straight forward. 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 external flash memory.

The functions listed in Table 7-3 are provided by the library for use with the decoder. The function `libExtFlashRead()` performs the task of reading a frame of encoded data into the correct library data buffer for decoding and managing the pertinent internal structure elements of the `codecdata` structure.

TABLE 7-3: DECODER EXTERNAL FLASH MEMORY FUNCTIONS

Function	Purpose
`libExtFlashInit()`	Initializes I/O lines to interface with Flash memory
`libExtFlashReset()`	Resets Flash memory
`libExtFlashRead()`	Reads a frame of encoded speech for libDecoder() and manages read pointers

7.4 STORING SPEECH ENCODING UTILITY DATA TO EXTERNAL FLASH MEMORY

The speech encoding utility (see **Chapter 6. "Speech Encoding Utility"**) provides the ability to 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 only contains 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.

7.5 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 `p30f6014.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. Since the memory map of the AMD29F200B contains only of one section, a custom linker script can be used with the MPLAB C30 Language Tools to generate a hex file targeted for the AMD29F200B. The installation directory `\Utils\ExternalFlashHexMaker` contains an MPLAB IDE project with the custom linker script file `external_flash.gld`. To use this file, simply 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 these steps:

1. Open the External_Flash project from MPLAB IDE.

> **Note:** The default path is `c:\Program Files\Microchip\dsPIC30F_SEDLibrary\Utils\ExternalFlashMaker\external_flash.mcp`

2. Locate the speech file that you want to load into Flash memory. If you used the default location when you created the speech file (see**Chapter 6. "Speech Encoding Utility"**), the file should be in the dsPIC30F_SEDLibrary\Utils\SpeechEncodingUtility folder.
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 called `External_Flash.hex` will be created. This file must now be programmed to external Flash memory.

7.6 PROGRAMMING THE HEX FILE TO EXTERNAL FLASH MEMORY

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

The EFP utility is capable of erasing and programming 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 Development Board, but the software can be tailored to run on your own hardware platform.

7.6.1 Building the EFP Utility

The EFP utility consists of the files shown in Table 7-4. To build the EFP project, follow these steps.

1. Launch MPLAB IDE and open the `efp.mcp` project located in the `Utils\ExternalFlashProgrammer` folder.
2. From the *Project>Build All* menu, build the project.

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

TABLE 7-4: 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
`emp_d.h`	Header file for global data
`emp_f.h`	Header file for function prototypes
`emp_m.h`	Header file for macros

7.6.2 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. You can alter the baud rate and system frequency using `#defines` in the `emp.h` header file. 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.

```
#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 & EC_PLL16);  // use EC with x16 PLL
```

7.6.3 PC UART Software

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

- Specified baud rate (19200 baud max)
- 8 data bits - no parity bits - 1 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. The maximum baud rate is 19200; however, lower baud rates can be used.

7.7 RUNNING THE EFP UTILITY

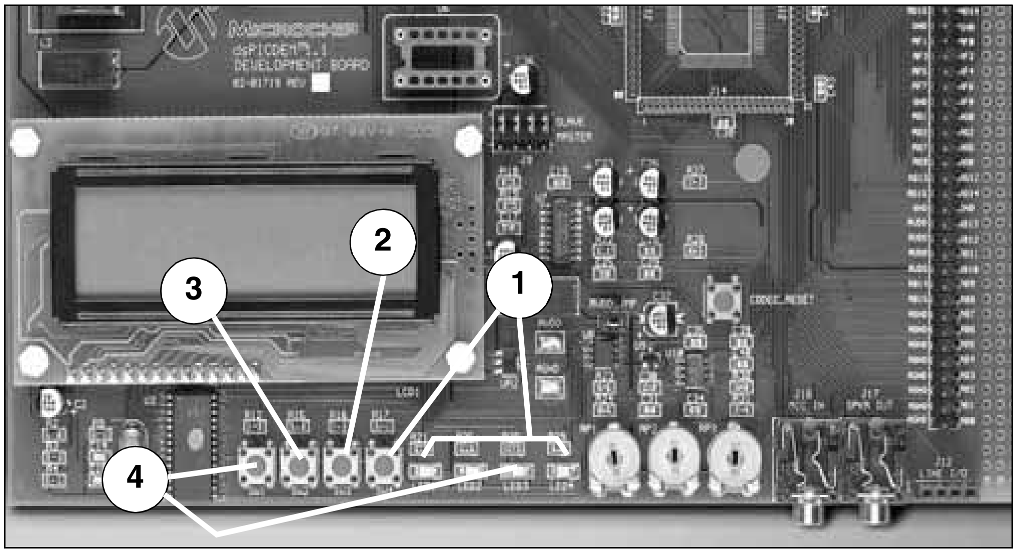
After building the EFP utility for your system as described in **Section 7.6.1 "Building the EFP Utility"**, it can be used to erase, program and read external flash memory. The EFP utility distributed with the dsPIC30F Speech Encoding/Decoding Library is targeted specifically for the dsPICDEM 1.1 Development Board. You may need to modify the EFP utility if you use your own hardware platform.

7.7.1 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 Development Board by following this procedure (see Figure 7-1):

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

FIGURE 7-1: EXTERNAL FLASH ERASE PROCESS FROM dsPICDEM™ 1.1



The chip erase cycle begins after the last switch (SW1) has been pressed in the proper sequence. You will want to observe the LEDs at that point. The erase cycle takes just a few (2-5) seconds. When the erase cycle is successfully completed, LED4 blinks five times and then turns off. At that point, you can program the external Flash memory.

If the erase cycle fails, LED4 blinks continuously. The only way to recover from this situation is to reset the dsPIC30F and repeat the erase key sequence.

You can abort an erase cycle after initiating the key sequence by simple pressing Switch 4 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've 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.

7.7.2 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 7.6.3 "PC UART Software"**)
2. Using the *Transfer>Send Text File* menu, download the target hex 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, the LEDs 1-4 turn off and LED3 blinks 5 times.

Note: If the programming has failed, LED3 will blink continuously. The only way to recover from this situation is to reset the dsPIC30F. Programming will fail if the external Flash memory has not been erased before programming begins.
During programming, no erase or verifying of memory can take place. You must wait until programming has completed to perform these other operations.

4. Proceed to **Section 7.7.3 "Verifying the Programming of External Flash"**.

7.7.3 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 1 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 this procedure to verify a program:

1. From the HyperTerminal *Transfer* menu select *Capture Text*.
2. On the dsPICDEM 1.1 board, press switch SW1. When the EFP 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 *Transfer* menu select *Capture Text>Stop* to stop HyperTerminal capture.
5. Compare the data returned from the EFP utility with the contents of the `External_Flash.hex` file.

Note: If switch SW1 is pressed 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.

7.7.4 Reading the External Flash

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

Use this procedure to read the external Flash memory and store it to a text file:

1. From the HyperTerminal *Transfer* menu select *Capture Text*.
2. On the dsPICDEM 1.1 board, press switch SW2. When the EFP detects the switch action it reads the lower 64 Kwords (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* menu select *Capture Text>Stop* to stop 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.

7.8 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 7-5.

TABLE 7-5: 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

7.9 OTHER EXTERNAL SOLUTIONS

The dsPIC30F Speech Encoding/Decoding Library includes drivers for interfacing with an AMD29F200B Flash memory. However, you may use any external memory that satisfies your application's requirements. Serial EEPROMs, byte-wide non-volatile memories or other 16-bit non-volatile 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. Functions comparable to those identified in Table 7-2 and Table 7-3 must be written and called from your application. For information on real-time interfacing with the library, review the guidelines in **Chapter 4. "Incorporating Speech Encoding"** and **Chapter 5. "Incorporating Speech Decoding"**.

Chapter 8. Speech Encoding Sample Application

8.1 INTRODUCTION

The speech encoding sample application demonstrates stand-alone speech encoding and playback from on-chip data EEPROM memory.

8.2 HIGHLIGHTS

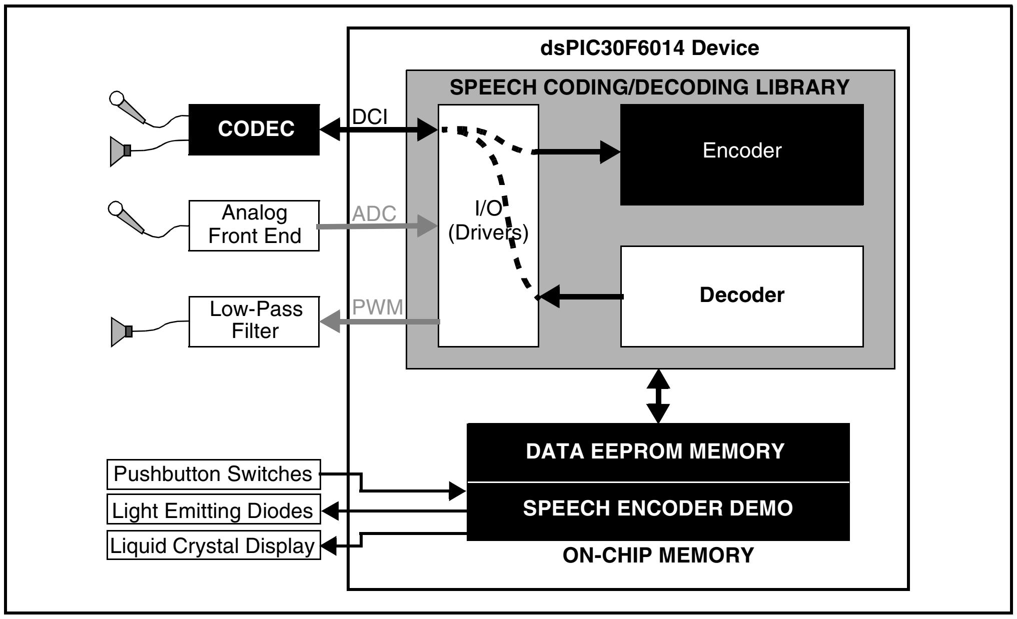
Items discussed in this chapter are:

- Overview
- Demo Setup
- Demo Operation
- Modifying the Sample Application

8.3 OVERVIEW

The speech encoding sample application demonstrates on-chip encoding of a speech sample, storing it to on-chip data EEPROM and then decoding the encoded speech for playback. This demonstration runs on the dsPICDEM 1.1 Development Board with a dsPIC30F6014 device. The dsPIC30F6014 device provides stand-alone processing for the sample application program, while the dsPICDEM 1.1 board supports the peripheral devices (codec, LCD, LEDs, and switches) used with the program. The speech sample is played through the Si3000 codec to allow you to listen to the audio quality of the encoded sample. Figure 8-1 is a simplified overview of the speech encoding sample application.

FIGURE 8-1: SPEECH ENCODER DEMO



The audio signal from the microphone is captured from the codec by the Data Converter Interface (DCI) and encoded to on-chip data EEPROM memory. The program is initiated and controlled from the pushbutton switches. The reset switch initializes the program. Switch S1 triggers an encoding session. Switch S2 triggers playback of the stored sample. The LEDs turn on and off based on the current operating mode while the LCD displays a menu and status message.

The speech encoding sample application is targeted to run on the dsPIC30F6014 processor with 4 Kbytes of data EEPROM, which provides storage support for approximately four seconds of speech. The sample encoder application automatically stops encoding speech as the data EEPROM approaches capacity.

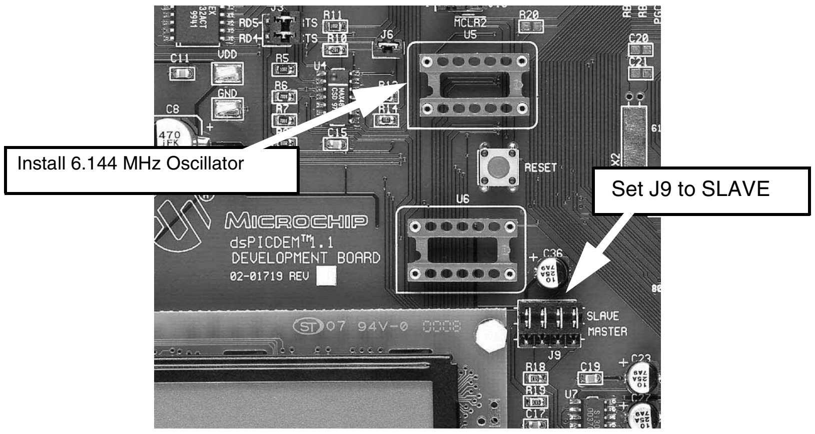
8.4 DEMO SETUP

Follow these steps to set up the demonstration.

8.4.1 Configure dsPICDEM 1.1 Board

1. Insert a 6.144 MHz oscillator into oscillator socket U5 on the dsPICDEM 1.1 board, as shown in Figure 8-2. You can use the oscillator included in the dsPICDEM 1.1 Accessory Kit or supply your own.
2. Set the jumper marked J9 to the 'Slave' position. This jumper allows the on-board Si3000 codec to function as a slave device to the dsPIC30F.

FIGURE 8-2: DEMO BOARD SETUP



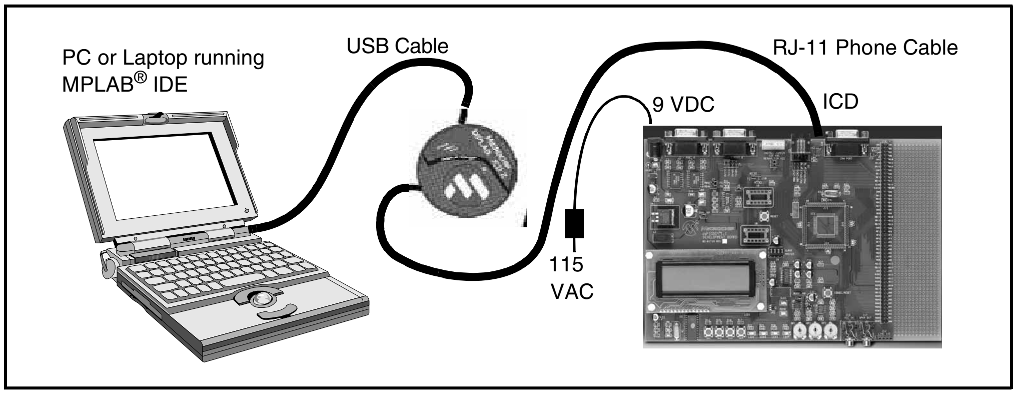
3. Apply power to the board.

8.4.2 Program dsPIC30F Device

After the board is configured, a project must be built in MPLAB IDE to program the demo application into the dsPIC30F device on the dsPICDEM 1.1 board. Use these steps:

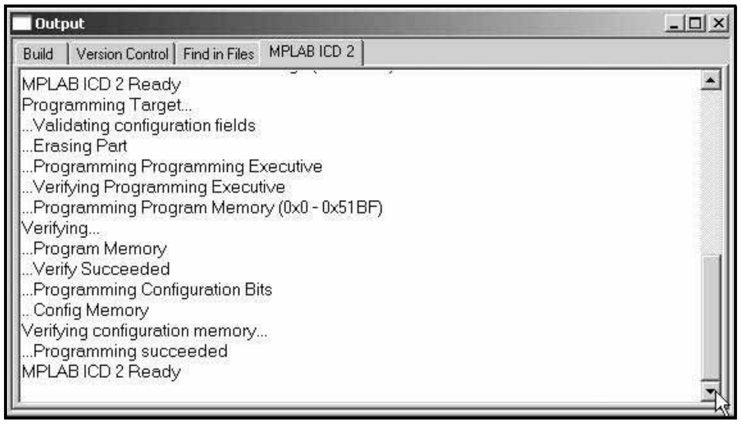
1. Launch MPLAB IDE and open the `encoder_demo.mcp` project located in the `Demo\Encoder` folder.
2. From the *Project>Build All* menu, build the project.
3. Plug the MPLAB ICD2 into the dsPICDEM 1.1 Board (see Figure 8-3).
4. From the *Programmer>Connect* menu, select the dsPIC30F6014 device.
5. From the *Programmer>Program* menu, program the dsPIC30F6014 device.

FIGURE 8-3: PROGRAMMING SETUP



6. When the project output window shows that programming is complete, as shown in Figure 8-4, unplug the MPLAB ICD2.

FIGURE 8-4: PROGRAMMING STATUS WINDOW

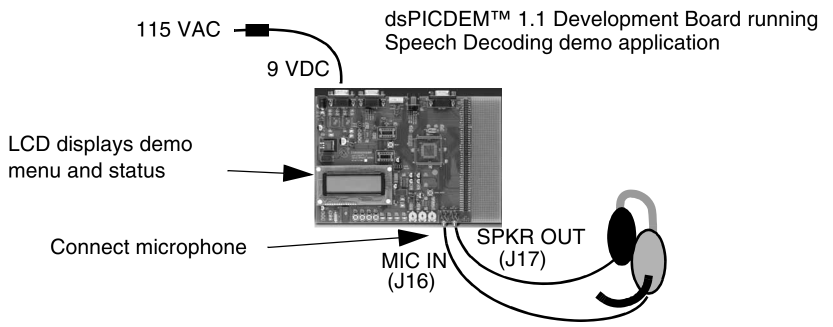


```
MPLAB ICD 2 Ready
Programming Target...
...Validating configuration fields
...Erasing Part
...Programming Programming Executive
...Verifying Programming Executive
...Programming Program Memory (0x0 - 0x51BF)
Verifying...
...Program Memory
...Verify Succeeded
...Programming Configuration Bits
.. Config Memory
Verifying configuration memory...
...Programming succeeded
MPLAB ICD 2 Ready
```

8.4.3 Set Up Demo

After the dsPIC30F6014 is programmed, attach the headset, as shown in Figure 8-5.

FIGURE 8-5: DEMO SETUP

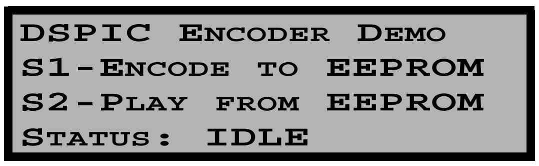

115 VAC
9 VDC
dsPICDEM™ 1.1 Development Board running Speech Decoding demo application
LCD displays demo menu and status
Connect microphone
MIC IN (J16)
SPKR OUT (J17)

8.5 DEMO OPERATION

To run the demo application:

1. Place the headset on your head with the microphone 1-2 inches away from your mouth.
2. Press the Reset button on the dsPICDEM 1.1 board to initialize the program. The LCD indicates that the program is idle, as shown in Figure 8-6.

FIGURE 8-6: DEMO STATUS ON LCD

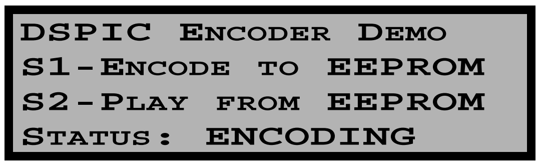


```
DSPIC ENCODER DEMO
S1-ENCODE TO EEPROM
S2-PLAY FROM EEPROM
STATUS: IDLE
```

3. Press switch S1. As soon as the status message changes to "**Status: ENCODING**", as shown in Figure 8-7, record your voice.

> **NOTE:** You only have four seconds of record time. After four seconds the demo automatically returns to the idle state ("**Status: IDLE**")

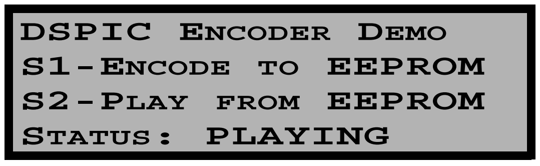
FIGURE 8-7: ENCODING STATUS DISPLAY



```
DSPIC ENCODER DEMO
S1-ENCODE TO EEPROM
S2-PLAY FROM EEPROM
STATUS: ENCODING
```

4. To listen to your stored speech sample, push switch S2.
   As the sample is decoded and played back from data EEPROM, the LCD displays "**Status: PLAYING**", as shown in Figure 8-8.
5. To make another recording, push switch S1. Your previous message is automatically erased and you can record another 4-second speech sample.

FIGURE 8-8: PLAYBACK STATUS DISPLAY



```
DSPIC ENCODER DEMO
S1-ENCODE TO EEPROM
S2-PLAY FROM EEPROM
STATUS: PLAYING
```

8.6 OPTIMIZING THE SAMPLE APPLICATION

The speech encoding sample application is optimized to run with a LabTec Axis-301 headset microphone. This headset has demonstrated good performance across a wide cross section of speakers. If you are using a different headset model and you are not satisfied with the performance of the demo, it is likely that the microphone gain, input volume control and/or output volume control level of the Si3000 codec needs to be modified. These parameters are defined in the `spxlib_Si3000.h` file and can be modified for your own headset. For information on tuning the Si3000 codec settings for your application, see **APPENDIX A. "SI3000 CODEC CONFIGURATION"**

8.7 MODIFYING THE SAMPLE APPLICATION

The speech encoding sample application performs encoding with Voice Activity Detection (VAD) disabled. If you want to use the encoder with Voice Activity Detection enabled, change the value of the VAD symbol from '`0`' to '`1`'. The VAD symbol is defined in the `spxlib_common.h` file.

```
#define VAD             0x01     //VAD enabled
```

After you change the VAD symbol, you must rebuild the demo and reprogram the dsPIC30F6014 device on the development board.

NOTES:

Chapter 9. Speech Decoding Sample Application

9.1 INTRODUCTION

The Speech Decoder sample application demonstrates stand-alone speech playback from on-chip program memory.

9.2 HIGHLIGHTS

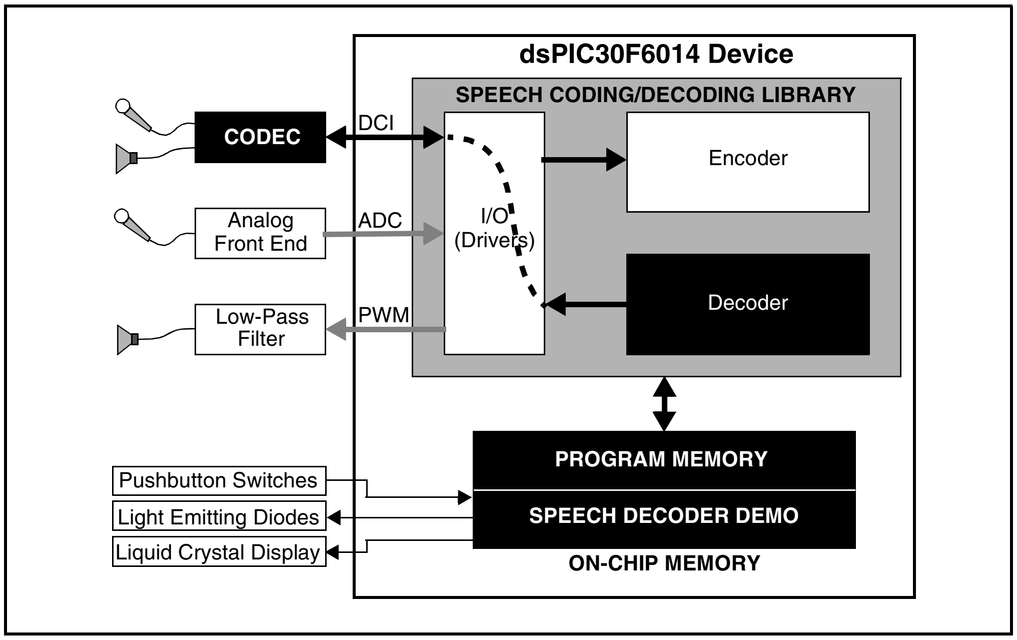
This section describes the Speech Decoding Library demonstration. Items discussed in this chapter include:

- Overview
- Demo Setup
- Demonstration Application
- Optimizing the Demonstration Application
- Modifying the Demonstration Application

9.3 OVERVIEW

The speech decoding sample application demonstrates on-chip decoding and play-back of a speech sample stored in dsPIC30F program memory. This demonstration runs on the dsPICDEM 1.1 Development Board with a dsPIC30F6014 device. The dsPIC30F6014 device provides stand-alone processing for the sample application program, while the dsPICDEM 1.1 board supports the peripheral devices (codec, LCD, LEDs, and switches) used with the program. The Si3000 codec is used for playback to allow you to listen to the audio quality of the encoder and decoder. Figure 9-1 is a simplified overview of the speech encoding sample application.

FIGURE 9-1: SPEECH DECODER DEMO



The program is initiated and controlled from the pushbutton switches. The reset switch initializes the program. Switch S1 triggers playback of the stored sample. Switch S2 halts the playback. Switch S3 triggers playback of the stored sample in a continuous loop. Switch S4 "*rewinds*" the speech sample and triggers playback. The LEDs turn on and off based on the current operating mode, while the LCD displays a menu and status message.

The speech decoding sample application is targeted to run on the dsPIC30F6014 processor. This device features 144 Kbytes of memory and can store approximately two minutes of speech data with an application that includes the library.

The encoded speech sample which is stored in program memory and played back was generated using the speech encoding utility (see **Chapter 6. "Speech Encoding Utility"**). The demonstration can be easily modified to play your own files created by the speech encoding utility. To perform this procedure, see **Section 9.7 "Modifying the Demonstration Application"**.

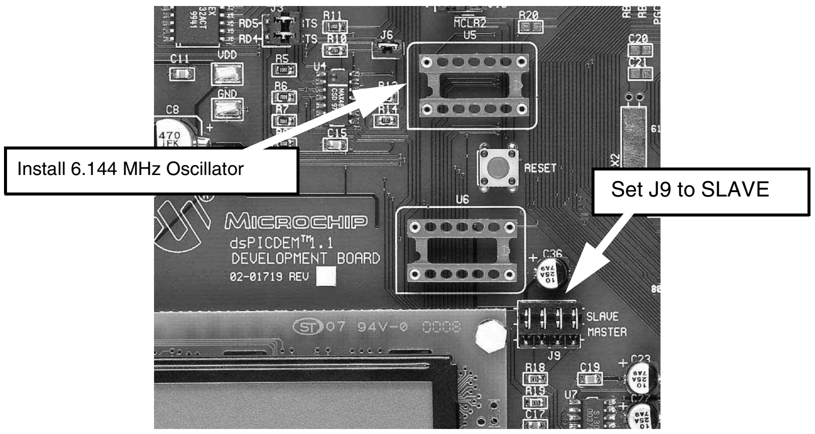
9.4 DEMO SETUP

Follow these steps to set up the demonstration.

9.4.1 Configure dsPICDEM 1.1 Board

1. Insert a 6.144 MHz oscillator into oscillator socket U5 on the dsPICDEM 1.1 board, as shown in Figure 9-2. You can use the oscillator included in the dsPICDEM 1.1 Accessory Kit or supply your own.
2. Set the jumper marked J9 to the 'Slave' position. This jumper allows the on-board Si3000 codec to function as slave device to the dsPIC30F.
3. Apply power to the board.

FIGURE 9-2: DEMO BOARD SETUP

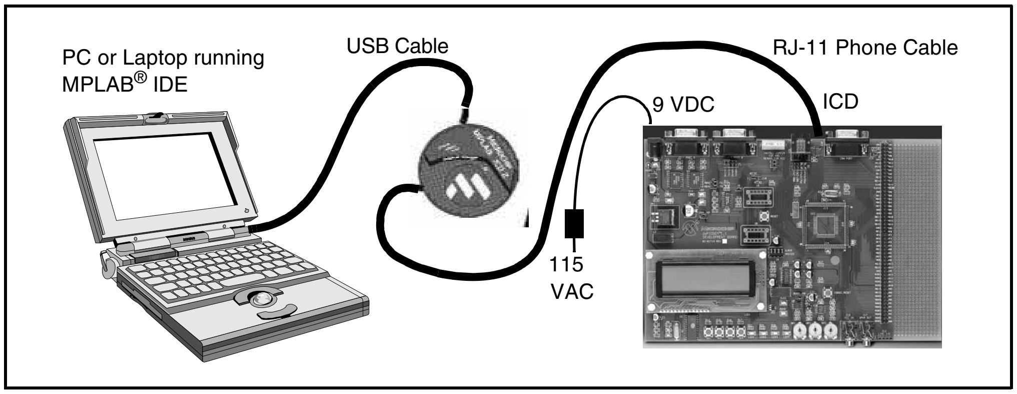


9.4.2 Program dsPIC30F Device

After the board is configured, a project must be built in MPLAB IDE to program the demo application into the dsPIC30F device on the dsPICDEM 1.1 board. Follow these steps:

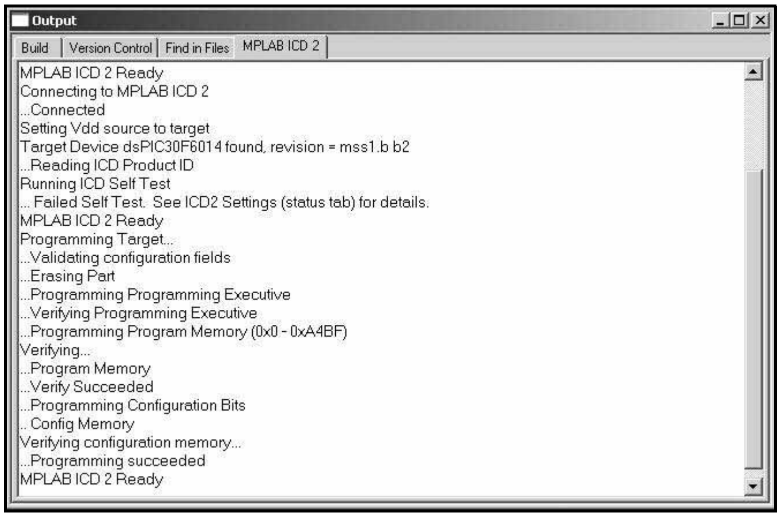
1. Launch MPLAB IDE.
2. From the *Project>Open* menu, open the `decoder_demo.mcp` project (located in the `Demo\Decoder` folder).
3. From the *Project>Build All* menu, build the project.
4. From the *Programmer>Connect* menu, select the dsPIC30F6014 device.
5. Plug the MPLAB ICD2 into the dsPICDEM 1.1 Board (see Figure 9-3).
6. From the *Programmer>Program* menu, program the dsPIC30F6014 device on the board.

FIGURE 9-3: PROGRAMMING SETUP



7. When the project output window shows that programming is complete, as shown in Figure 9-4, unplug the MPLAB ICD2.

FIGURE 9-4: PROGRAMMING STATUS WINDOW

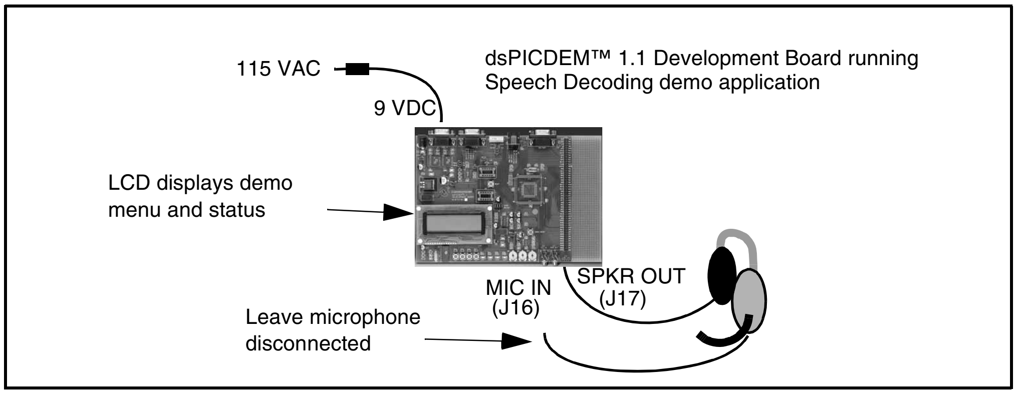


```
MPLAB ICD 2 Ready
Connecting to MPLAB ICD 2
...Connected
Setting Vdd source to target
Target Device dsPIC30F6014 found, revision = mss1.b b2
...Reading ICD Product ID
Running ICD Self Test
... Failed Self Test.  See ICD2 Settings (status tab) for details.
MPLAB ICD 2 Ready
Programming Target...
...Validating configuration fields
...Erasing Part
...Programming Programming Executive
...Verifying Programming Executive
...Programming Program Memory (0x0 - 0xA4BF)
Verifying...
...Program Memory
...Verify Succeeded
...Programming Configuration Bits
.. Config Memory
Verifying configuration memory...
...Programming succeeded
MPLAB ICD 2 Ready
```

9.4.3 Set Up Demo

After the dsPIC30F6014 is programmed, attach the headset, as shown in Figure 9-5. Leave the microphone disconnected for this demo set up.

FIGURE 9-5: DEMO SETUP

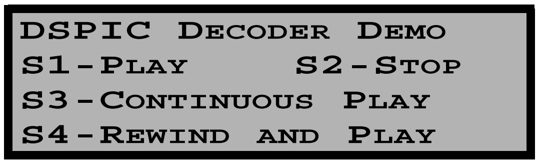

115 VAC
9 VDC
dsPICDEM™ 1.1 Development Board running Speech Decoding demo application
LCD displays demo menu and status
MIC IN (J16)
SPKR OUT (J17)
Leave microphone disconnected

9.5 DEMONSTRATION APPLICATION

After the demo application has been programmed it is ready to run. Follow these steps:

1. Place the headset on your head.
2. Press the Reset button on the dsPICDEM 1.1 board to initialize the program. The LCD indicates that the program is running, as shown in Figure 9-6.

FIGURE 9-6: DECODER DEMO STATUS ON LCD



```
DSPIC Decoder Demo
S1-Play      S2-Stop
S3-Continuous Play
S4-Rewind and Play
```

3. Press switch S1. The speech sample stored in program memory plays back through the Si3000 codec and can be heard in the headset. Observe that LED1 lights up and LEDs 2-4 turn off.
4. To stop the play back, press switch S2. Observe that LED2 lights up and LEDs 1, 3 and 4 turn off and the speech sample is no longer heard.
5. To play the stored speech sample back continuously, press switch S3. Observe that the speech sample repeats in the headset, while LED3 lights up and LEDs 1, 2 and 4 turn off.
6. To rewind the speech file and play it from the beginning, push switch S4. Observe that the speech sample jumps to the beginning in the headset, while LED4 lights up and LEDs 1-3 turn off.

Note: During this demo you can press any switch at any time regardless of the state of the program.

9.6 OPTIMIZING THE DEMONSTRATION APPLICATION

The Speech Decoding demo is optimized to run with a LabTec Axis-301 headset, which has demonstrated good performance across a wide cross section of speakers. If you are using a different headset model and you are not satisfied with the performance of the demo, it is likely that the output volume control level of the Si3000 codec needs to be modified. The value of the parameters is defined in the `spxlib_Si3000.h` file and can be modified for your headset. For information on tuning the Si3000 codec settings for your application, see **Appendix A. "Si3000 Codec Configuration"**.

9.7 MODIFYING THE DEMONSTRATION APPLICATION

To help you evaluate how the library will perform in your end application, the Speech Decoding demo can be modified to play your own speech samples encoded with the speech encoding utility.

To modify the demo speech sample:

1. Launch MPLAB IDE and open the `decoder_demo.mcp` project, located in the `Demo\Decoder` folder.
2. Delete the `f_English.s` file from the project.

To add your own encoded speech file:

1. Use the speech encoding utility to record a new sample.
2. Select program memory as the target memory to generate an assembly source file (*.s) and store it in program memory.
3. Use the default array name (`speex_data`) provided by the speech encoding utility.
   Since this array is defined in an assembly source file, the array symbol name should be `_speex_data`, as shown in Example 9-1.

   If you create your file with a different array name, change the global declaration and symbol definition to `_speex_data`.

EXAMPLE 9-1: ARRAY NAME

```
.global _speex_data         - - - > global declaration

.section .speex, "x"
_speex_data:                - - - > array symbol definition
.pword 0x70AE9C, 0xCE0500, 0x028073, 0xC039E7, 0x905201, 0x802D80
.pword 0x9C0000, 0x0180AC, 0x0000E4, 0xB08E00, 0x130000, 0x000000
```

4. Open your encoded speech source file and determine the size of the array in bytes. The array size is shown in the header of the source file generated by the speech encoding utility (see Example 9-2).

EXAMPLE 9-2: ARRAY SIZE

```
*   Speech Encoder Utility settings:
*       Input Source:    my_sample.wav
*       Output Array:    speex_data
*       Array Size:      59961 bytes      - - - > array size in bytes
*       Target Memory:   Program Memory
*       VAD:             Disabled
```

5. Open the `spxlib_common.h` header file and change the definition of the symbol `ARRAYSIZEINBYTES` to the size of your array. You may want to comment out the original definition and make a new definition for your own file as shown in Example 9-3:

EXAMPLE 9-3: ARRAY DEFINITION

```
#define ARRAYSIZEINBYTES 59961   // for my_sample.s
//#define ARRAYSIZEINBYTES 49146   // for f_english.s
```

6. Select *Project>Build All* to build the project.
7. Program and Run the demo as before.

Appendix A. Si3000 Codec Configuration

A.1 INTRODUCTION

The dsPIC30F Speech Encoding/Decoding Library provides native support for the Silicon Labs Si3000 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 is 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	dsPIC30F Master Setting	dsPIC30F 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 10dB Handset input muted
ADC Volume Control	0x5C	0x5C	RX gain 0dB Line out muted Handset out muted
DAC Volume Control	0x5F	0x5F	TX gain 0dB Speaker left active Speaker right active
Status Report	0x0	0x0	Read only register
Analog Attenuation	0x0	0x0	Line out 0dB attenuation Speaker out 0dB attenuation

By default, the dsPIC30F is the codec clock master and this 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 dsPIC30F 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 3-1 for valid system operating frequencies when the dsPIC30F 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 DSPIC30F AS CLOCK SLAVE

If you want to operate your application using the Si3000 codec at an operating frequency different than those shown in Table 3-1, 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 dsPIC30F is the clock master, the dsPIC30F 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 dsPIC30F.

To use the dsPIC30F 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 board, don't forget to move jumper J9 to the MASTER setting. This indicates that the Si3000 is now the clock master.

A.4 MODIFYING THE CODEC GAIN AND VOLUME CONTROLS

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

The following set of `#defines` 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 3dB:

```
#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 these symbols:

```
#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 3dB, you can define these symbols:

```
#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    /* -6dB 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 Development Board. The required I/O lines for this interface are shown in Table B-1.

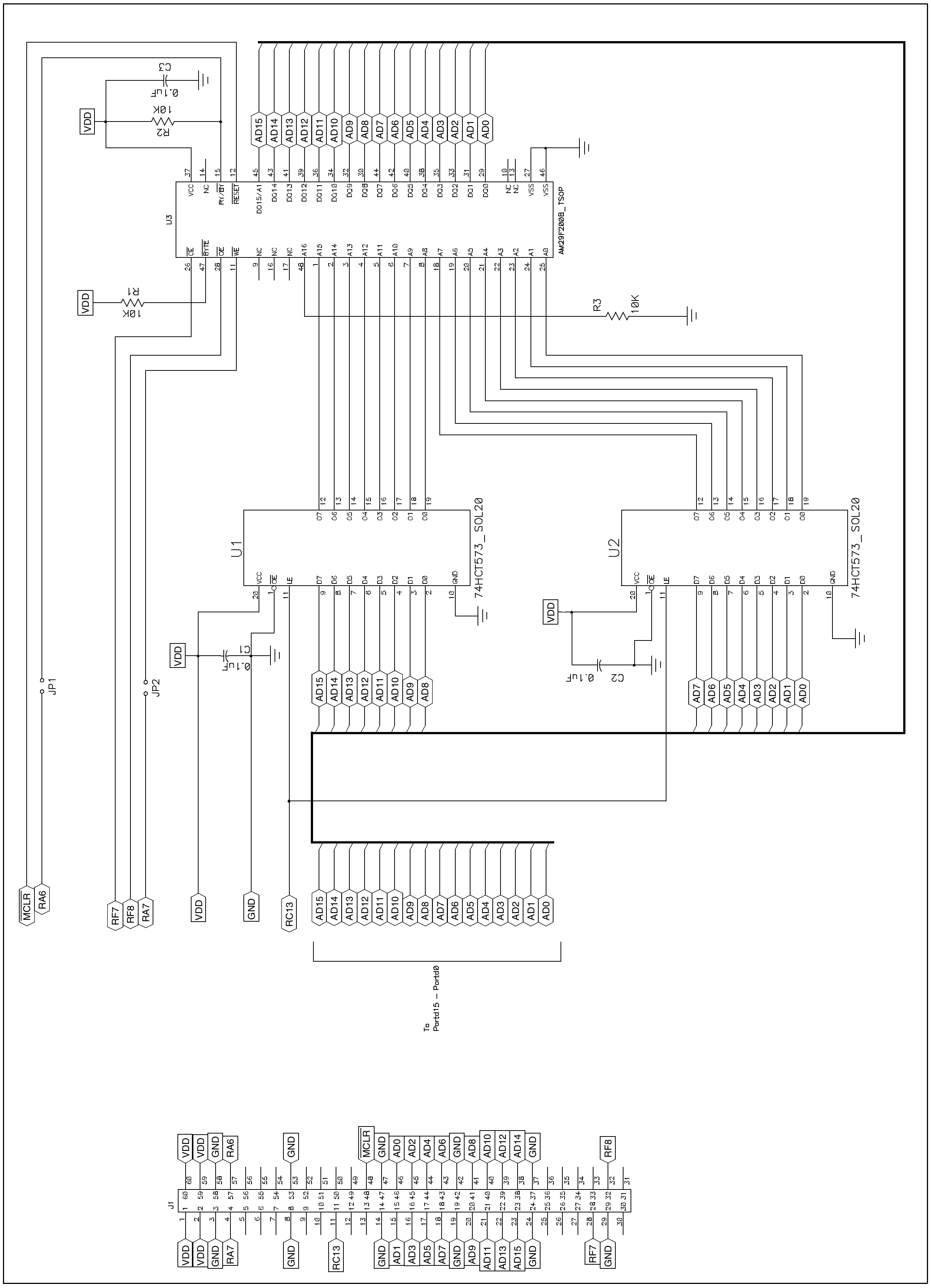
TABLE B-1: PINS USED FOR EXTERNAL MEMORY INTERFACE

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 that is included with the dsPIC30F 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 64-Kword interface can store approximately two minutes of compressed speech. If 128 Kwords 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.

FIGURE B-1: EXTERNAL MEMORY INTERFACE SCHEMATIC



Appendix C. ADC/PWM Interface Reference Design

C.1 OVERVIEW

This appendix provides a sample circuit that consists of an analog front-end circuit for a microphone, and an analog output circuit for a PWM signal. The microphone interface circuit consists of an 8th order low-pass, switched-capacitor filter with 3.8 kHz cut-off frequency, tunable preamplifier, tunable level shifter and tunable gain control.

The PWM interface circuit consists of an 8th order low-pass, switched-capacitor filter with 3.8 kHz cut-off frequency and tunable gain control for left and right audio channels.

FIGURE C-1: ADC/PWM INTERFACE (SHEET 1 OF 4)

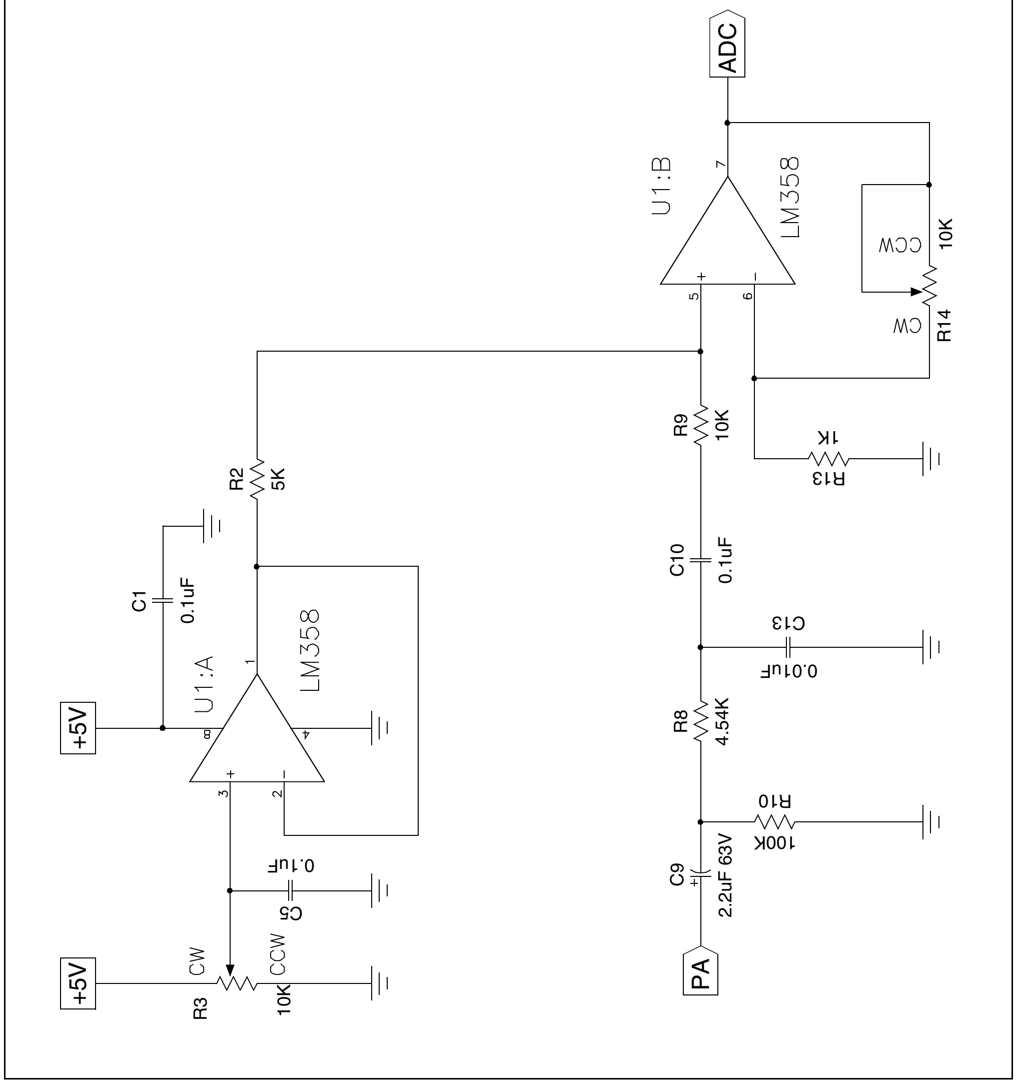


FIGURE C-2: ADC/PWM INTERFACE (SHEET 2 OF 4)

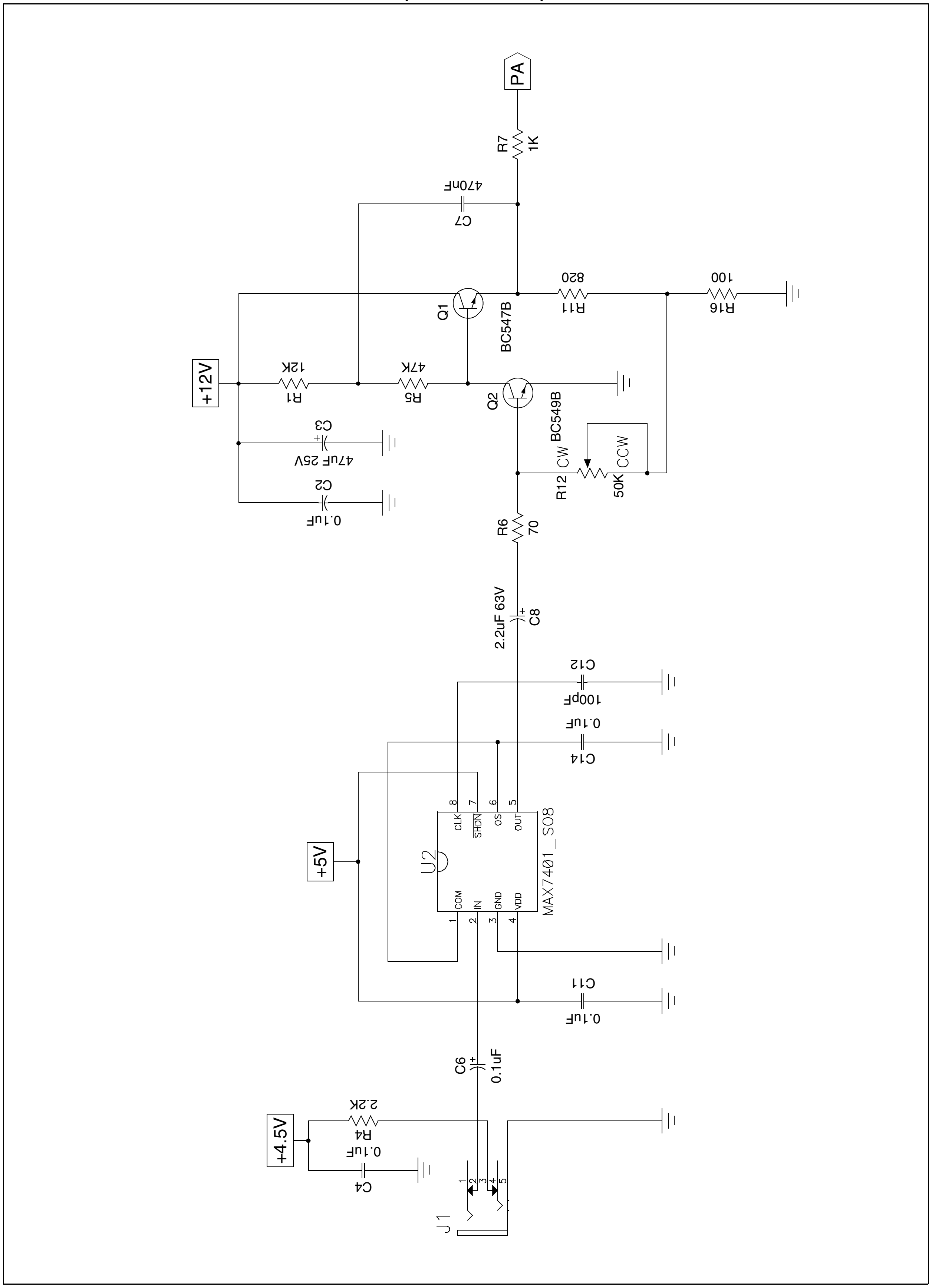


FIGURE C-3: ADC/PWM INTERFACE (SHEET 3 OF 4)

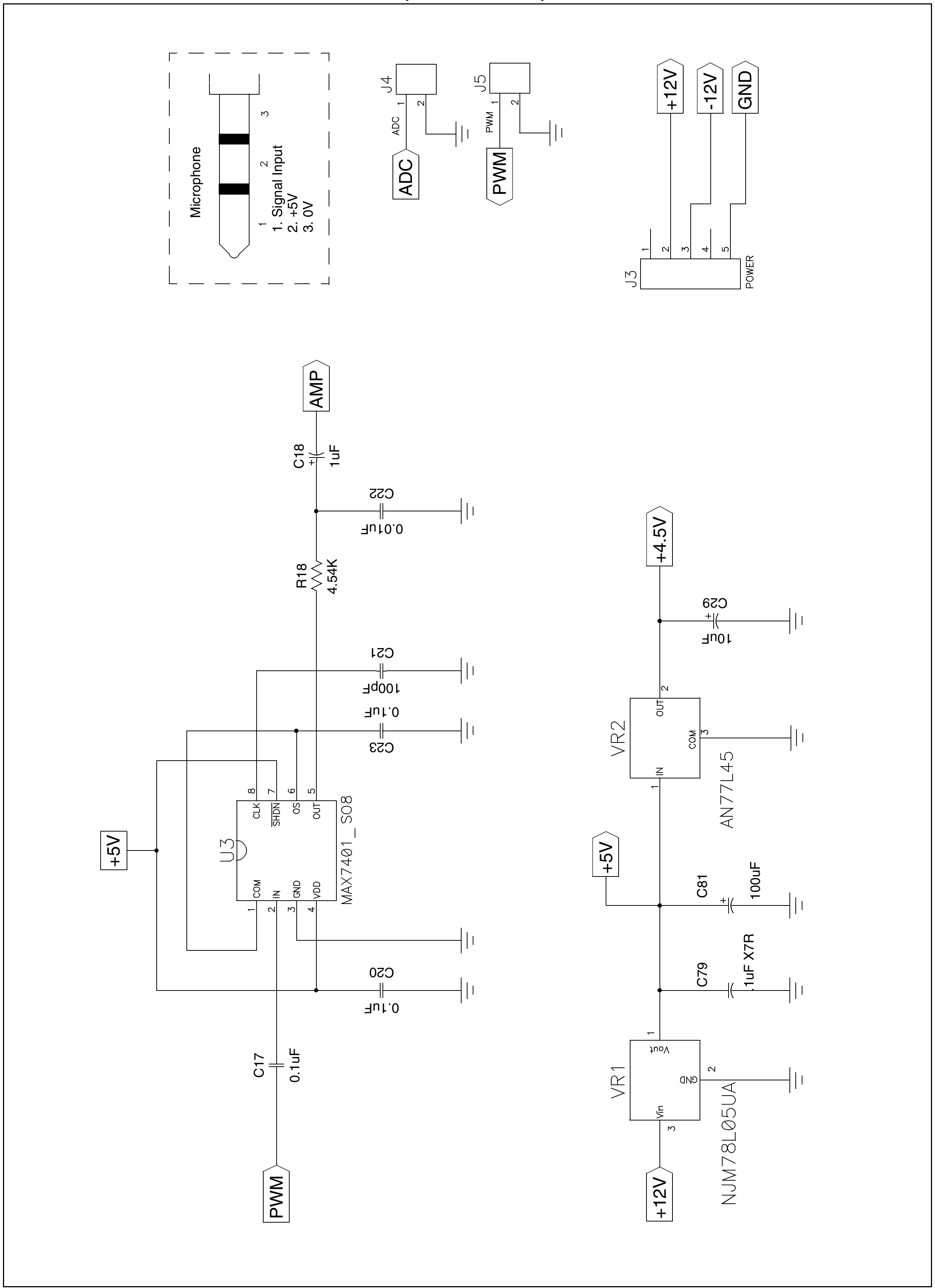
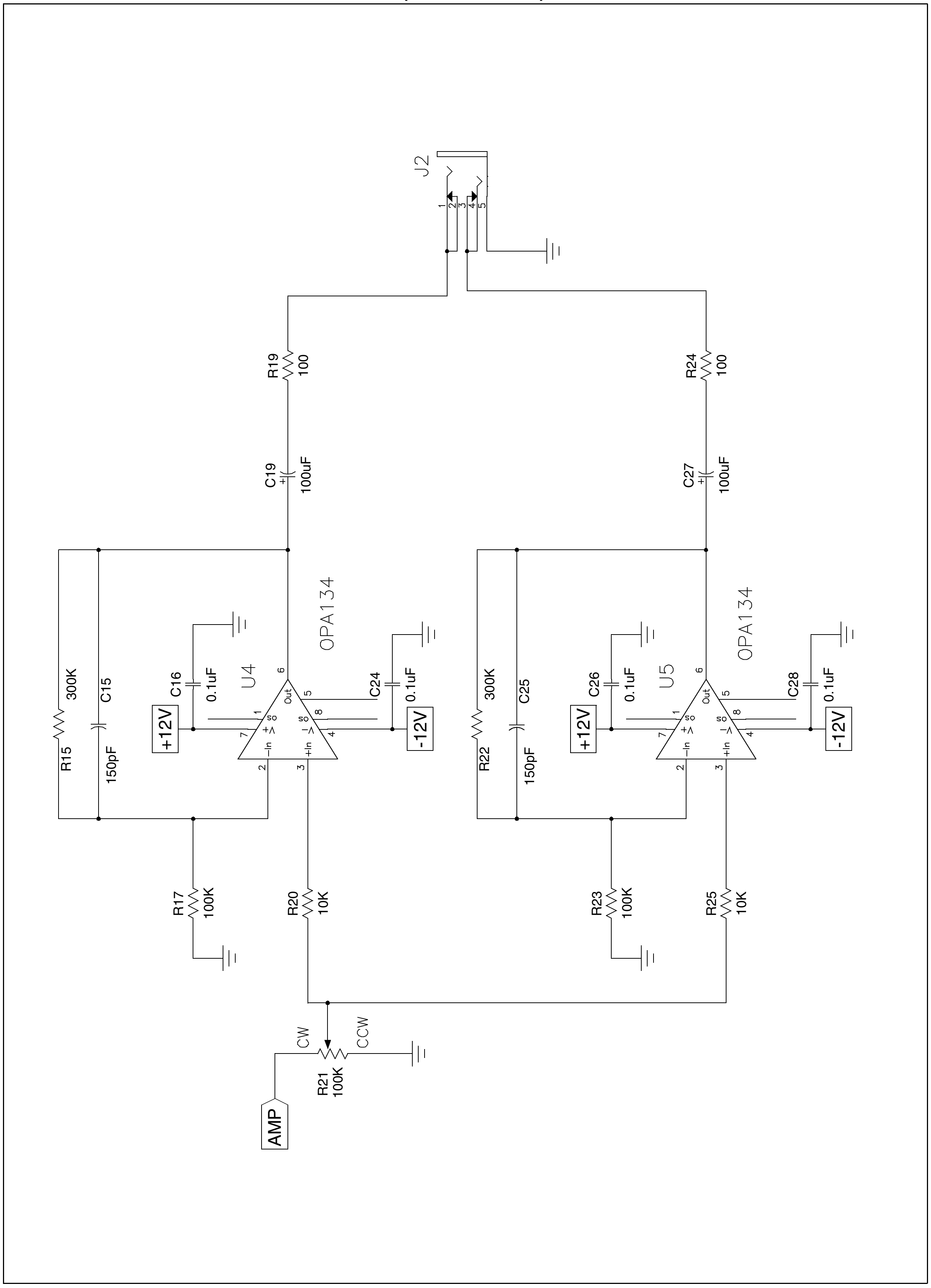


FIGURE C-4: ADC/PWM INTERFACE (SHEET 1 OF 4)



Appendix D. Sample Applications

D.1 LIBRARY DECODER SAMPLE APPLICATION

```
//-------------------------------------------------------------
// Decoder Sample Application
//-------------------------------------------------------------
#include "spxlib_common.h"
#include "spxlib_Si3000.h"
#include "spxlib_pwm.h"
short in1[160], in2[160];
char out1[20], out2[20];
char EncoderState = 0;             // DCI ISR flag to choose playback

CODECSETUP                         // Instantiate CODECSETUP structure

#ifdef CODEC_INTERFACE
  SPXSI3000INIT                    // Instantiating SI-3000 structure
#else
  SPXPWMINIT                       // Instantiate PWM structure
#endif

int main (void)
{
  libExtFlashInit ();              // Initialize external flash
  libDecoderInit ();               // Initialize decoder

      /* decode first frame of speech */
  libarrayFillDecoderInputPM ();
  libDecoder ();
  libBufManagerDecoder ();         // Rewind the sample buffer pointer

#ifdef CODEC_INTERFACE
  libSi3000Init ();                //Initialize Si3000 and DCI registers.
  libSi3000StartSampling(&out1[0], &out2[0], &in1[0], &in2[0]);
  libStartPlay();
#else
  libPWMInit ();
        libPWMStartSampling (&out1[0], &out2[0], &in1[0], &in2[0]);
   #endif

while (1) {

  /* determine if it's time to read another frame of data */
  if ( (!codecdata.fFramedone) && (codecdata.fFrameplayed) &&
       (codecdata.recordsize - 1) != codecdata.framecount ) {
         libarrayFillDecoderExternalFlash ();
      }

  /* wait to run the decoder */
  if (codecdata.fFramedone) {
      libDecoder ();
      while (codecdata.fFrameplayed);  // wait for frame to played
      libBufManagerDecoder ();              // manage buffers
    }

  /* determine if the entire record has played */
  if ( ( codecdata. recordSize - 1 )  == codecdata.frameCount ) {
#ifdef CODEC_INTERFACE
       libSi3000StopSampling ();
#else
   libStopPWM ();
#endif
   libStopPlay ();
       libDecoderKill ();
       codecdata.fFramedone  = 0x0;
       codecdata.fFrameplayed = 0x0;
       codecdata.framecount  = 0x0;
       codecdata.sampleCount = 0x0;
       break;
       }
  }
while (1);
}
```

D.2 LIBRARY ENCODER SAMPLE APPLICATION

```
//--------------------------------------------------------------
// Encoder Sample Application
//--------------------------------------------------------------
#include "spxlib_common.h"
#include "spxlib_Si3000.h"
#include "spxlib_adc.h"
short in1[160], in2[160];
char out1[20], out2[20];
char EncoderState = 1;        //Flag used in DCI ISR to choose sampling

CODECSETUP                    //Macro for instantiating CODECSETUP structure.

#ifdef CODEC_INTERFACE
SPXSI3000INIT                 //Macro for instantiating SI3000 structure.
#else
SPXADC12INIT                  //Macro for instantiating ADC12 structure.
#endif

int main (void)
{
   /* initialize encoder */
libEncoderInit ( codecdata.vad );

/* Initialize sampling interface and and begin sampling */
   #ifdef CODEC_INTERFACE
libADC12Init ();
   libADC12StartSampling (&in1[0], &in2[0], &out1[0], &out2[0]);
#else
   libADC12Init ();
   libADC12StartSampling (&in1[0], &in2[0], &out1[0], &out2[0]);
#endif

while (1) {
   if (codecdata.fFramedone ) {
      libEncoder ();                /* Encode the filled buffer */
      libBufManagerEncoder ();      /* Manage the ping-pong buffers */

      /* Here you do something with the encoded data… */
      YourCall (codecdata.sampleEncdOpBuffer,
            codecdata.numOfencSamplesPerFrame);
      }

    /* determine if all frames are encoded */
   if ( codecdata.frameCount == codecdata.recordSize ) {
#ifdef CODEC_INTERFACE
   libSi3000StopSampline();
#else
    libADC12StopSampling();           /*Disables sampling from ADC12*/
#endif
      libEncoderKill ();              /*Destroys Encoder state*/
      codecdata.fFramedone = 0x00;    /*Clear flags*/
      codecdata.framecount = 0x00;
      }
   }                                  /*end while(1)*/
}                                     /*end main()*/
```

NOTES:

Index

NOTES:

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
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

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

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

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

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

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

ASIA/PACIFIC

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: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

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

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 - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Qingdao
Tel: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

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

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

Malaysia - Penang
Tel:011-604-646-8870
Fax:011-604-646-5086

Philippines - Manila
Tel: 011-632-634-9065
Fax: 011-632-634-9069

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

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

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

Taiwan - Hsinchu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria - Weis
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

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

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

Germany - Ismaning
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

England - Berkshire
Tel: 44-118-921-5869
Fax: 44-118-921-5820

04/20/05